Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Evaluación y Refinamiento del Sistema de Gestión del Ciclo de Vida de Titulaciones Universitarias

Autor: María Cantón González

Tutor(a): Angélica de Antonio Jiménez

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Evaluación y Refinamiento del Sistema de Gestión del Ciclo de Vida de Titulaciones Universitarias
Junio 2025

Autor: María Cantón González

Tutor:
Angélica de Antonio Jiménez
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado se enmarca en el proceso de modernización de los sistemas de gestión académica de la Universidad Politécnica de Madrid, y tiene como objetivo el diseño y desarrollo de un sistema web para la gestión del ciclo de vida de titulaciones universitarias. El presente trabajo parte de un proyecto previo desarrollado en el marco de un TFG anterior, sobre el cual se han identificado diversos errores funcionales, limitaciones técnicas y áreas susceptibles de mejora. A partir de dicho análisis, este proyecto se plantea como una evolución que corrige errores, refina la experiencia de usuario e incrementa la fiabilidad y coherencia del sistema original.

El sistema desarrollado implementa una arquitectura cliente-servidor compuesta por una interfaz *frontend* basada en React y un *backend* construido con Node.js (Express), que se conecta a una base de datos relacional para persistencia de la información. La aplicación permite consultar y gestionar titulaciones tanto en fase de diseño como ya implantadas, ofreciendo funcionalidades adaptadas al estado del título.

Desde el punto de vista técnico, el sistema incorpora una lógica basada en llamadas a servicios RESTful y manejo asincrónico de datos para garantizar fluidez, integridad y coherencia. Se han aplicado mejoras significativas en usabilidad, validación de datos y arquitectura del código, con el objetivo de hacer el sistema más robusto, mantenible y preparado para su evolución futura.

Este trabajo representa únicamente una parte del sistema global concebido para la gestión integral del ciclo de vida de titulaciones en la UPM. Si bien se han logrado avances sustanciales en la consolidación de funcionalidades clave, se prevé que el sistema continúe expandiéndose en futuras iteraciones, incorporando nuevos módulos, funcionalidades avanzadas y mecanismos de interoperabilidad con otros sistemas institucionales. De esta manera, este proyecto constituye una base sólida sobre la cual seguir construyendo una herramienta estratégica para la transformación digital de la universidad.

Abstract

This Final Degree Project is part of the broader modernization process of the academic management systems at the Universidad Politécnica de Madrid (UPM), and its main objective is the design and development of a web-based system for managing the life cycle of university degree programs. The project builds upon a previous Bachelor's Thesis in which several functional issues, technical limitations, and areas for improvement were identified. Based on that analysis, the present work proposes an enhanced version that corrects existing errors, refines the user experience, and increases the reliability and consistency of the original system.

The developed system implements a client-server architecture, with a *frontend* interface built using React and a *backend* developed with Node.js (Express), connected to a relational database for data persistence. The application allows users to view and manage degree programs both in the design phase and already implemented, offering features tailored to the current state of each program.

From a technical standpoint, the system incorporates logic based on RESTful service calls and asynchronous data handling to ensure smooth performance, data integrity, and consistency. Significant improvements have been made in usability, data validation, and code architecture to make the system more robust, maintainable, and ready for future evolution.

This work represents only a part of a broader system conceived for the comprehensive management of the degree program life cycle at UPM. While considerable progress has been made in consolidating key functionalities, the system is expected to continue expanding in future iterations with the inclusion of new modules, advanced features, and interoperability mechanisms with other institutional systems. Thus, this project lays a solid foundation for the ongoing development of a strategic tool aimed at the university’s digital transformation.

Tabla de contenidos

1 Introducción

1.1 Contexto

La Universidad Politécnica de Madrid, a través de diferentes Trabajos de Fin de Grado, ha puesto en marcha un proyecto para informatizar la gestión del ciclo de vida de las titulaciones universitarias. Este proceso, tradicionalmente basado en documentación dispersa y herramientas manuales como hojas de cálculo y correo electrónico, requiere una transformación digital que permita mejorar la eficiencia, trazabilidad y coordinación entre los distintos actores implicados en la gestión académica.

En este contexto, se definió un sistema modular orientado a cubrir todas las etapas del ciclo de vida de un título universitario: desde su diseño inicial, verificación y puesta en marcha, hasta su seguimiento, renovación de acreditación y, en su caso, extinción. Para llevarlo a cabo, se ha adoptado un enfoque iterativo e incremental. En una primera iteración desarrollada en un Trabajo de Fin de Grado previo [1], se elaboró un diseño de alto nivel del sistema, incluyendo los principales procesos y actores. En una segunda fase, también desarrollada en otro TFG [2], se implementaron las funcionalidades básicas del sistema y se diseñó un prototipo para la gestión de la oferta académica. La presente iteración, correspondiente a este TFG, representa la tercera fase del proyecto y tiene como objetivo principal evaluar, refinar y ampliar el sistema desarrollado anteriormente, integrando nuevas funcionalidades clave y preparando el sistema para su despliegue en un entorno real de pruebas.

1.2 Objetivos del trabajo y alcance

El objetivo general de este Trabajo Fin de Grado es avanzar hacia la consolidación de un sistema funcional para la gestión del ciclo de vida de las titulaciones universitarias, centrándose en un subconjunto significativo de casos de uso. A diferencia de fases anteriores centradas en el diseño conceptual o en prototipos funcionales parciales, en esta iteración se busca entregar una versión estable y operativa del sistema, capaz de ser transferida a sus usuarios

finales en un entorno de pruebas reales. Esta aproximación permite no solo validar el diseño y la implementación, sino también recoger retroalimentación de los agentes implicados con vistas a futuras mejoras.

Como primer paso, se ha realizado una evaluación técnica detallada del sistema previamente desarrollado, mediante pruebas exhaustivas orientadas a detectar y corregir posibles defectos.

Posteriormente, el trabajo se ha centrado en la ampliación del sistema mediante la incorporación de nuevas funcionalidades como las etapas correspondientes al diseño de un título o un módulo para la gestión de títulos vigentes, que facilita el seguimiento y mantenimiento de las titulaciones activas en la institución.

Además, se ha desarrollado un subsistema de gestión de usuarios y roles que permite establecer un control de accesos flexible, adaptado a la diversidad de perfiles que intervienen en los procesos de gestión (como responsables de título, jefatura de estudios, o personal de calidad).

A nivel de infraestructura, se ha llevado a cabo una reestructuración del modelo de datos con el objetivo de mejorar la eficiencia, consistencia y escalabilidad del sistema. En paralelo, se ha integrado un subsistema de gestión documental que permite almacenar y organizar de forma centralizada los documentos que se generan a lo largo del ciclo de vida de cada título, asegurando su trazabilidad y accesibilidad.

El alcance de este trabajo se centra en consolidar una base sólida y operativa del sistema, sobre la cual se podrán desarrollar futuras extensiones funcionales en próximas iteraciones del proyecto.

1.3 Estructura de la memoria

La memoria se estructura en varios capítulos que abordan de forma progresiva y detallada los distintos aspectos del proyecto de desarrollo y refinamiento del sistema de gestión del ciclo de vida de titulaciones universitarias. La organización del documento es la siguiente:

- **Capítulo 2. Antecedentes:** Este capítulo contextualiza el proyecto dentro del marco institucional de la Universidad Politécnica de Madrid, describiendo el ciclo de vida de las titulaciones y las dos iteraciones previas realizadas en Trabajos Fin de Grado anteriores. Se analizan los avances alcanzados en cada fase, así como sus limitaciones, sentando las bases para el trabajo desarrollado en esta tercera iteración.
- **Capítulo 3. Tecnologías:** Se describen las tecnologías empleadas en el desarrollo del sistema, tanto en el lado del cliente como del servidor. Se justifica su elección en función de los objetivos del proyecto y se analiza su papel dentro de la arquitectura general de la aplicación.
- **Capítulo 4. Desarrollo del sistema:** Este capítulo constituye el núcleo de la memoria y presenta el trabajo técnico realizado. Se detallan el rediseño e implementación de diversos subsistemas (como el diseño de títulos, gestión de usuarios y roles, documentación asociada, etc.), la reestructuración de la base de datos, las pruebas funcionales, así como las instrucciones para la instalación y despliegue del sistema.
- **Capítulo 5. Resultados y conclusiones:** En este capítulo se recogen los resultados obtenidos tras el desarrollo e integración de los distintos módulos del sistema. Se reflexiona sobre el cumplimiento de los objetivos planteados y se identifican posibles líneas de trabajo futuro.
- **Capítulo 6. Análisis de impacto:** Se evalúa el impacto potencial del sistema en los procesos de gestión académica de la UPM, considerando tanto aspectos técnicos como organizativos, y se analiza su viabilidad de adopción institucional.
- **Capítulo 7. Bibliografía:** Se enumeran las fuentes consultadas para la elaboración de la memoria y el desarrollo del sistema, incluyendo documentación técnica, normativa institucional y referencias bibliográficas relevantes.

2 Antecedentes

2.1 La gestión del ciclo de vida de las titulaciones universitarias en la UPM

La Universidad Politécnica de Madrid (UPM) gestiona sus titulaciones oficiales a través de un ciclo estructurado que garantiza su calidad y adecuación a lo largo del tiempo. Este ciclo de vida abarca desde el diseño inicial del título hasta su eventual extinción, pasando por fases de verificación, seguimiento y actualización. Cada fase está asociada a un estado concreto del título, lo que permite identificar claramente su situación dentro del proceso académico y administrativo.

2.1.1 Diseño del Título – Estado: *En Diseño*

El proceso de diseño es el punto de partida del ciclo de vida de una titulación y se corresponde con el estado *En Diseño*. Su objetivo es crear un proyecto académico sólido, alineado con las necesidades sociales y del mercado, que garantice una formación de calidad al estudiante. Para ello, se deben seguir tanto las directrices nacionales como las establecidas por la propia UPM. Este diseño implica la participación de diversos órganos de gobierno universitario como el Consejo de Gobierno, la Junta de Escuela, el Consejo de Departamento y el Consejo Social.

Durante esta fase, se definen los objetivos generales del título, sus competencias específicas, el plan de estudios, y otros aspectos clave como la modalidad de enseñanza o los criterios de acceso. El diseño debe contemplar también la viabilidad de implantación y la sostenibilidad académica del título. Los responsables de su elaboración, revisión y aprobación dentro del Sistema de Aseguramiento Interno de la Calidad (SAIC) son el Subdirector de Ordenación Académica, el Subdirector de Calidad y el Director del Centro.

2.1.2 Verificación – Estado: En Verificación/Verificado

Una vez aprobado el diseño por el Consejo de Gobierno, el título pasa al estado *En Verificación*. Una vez diseñado, el título debe pasar por un proceso de verificación externa que garantice su calidad y validez académica. Este proceso, llevado a cabo por una agencia de evaluación, tiene como finalidad certificar que

el programa cumple con los estándares establecidos y puede ser impartido de forma oficial. La verificación es condición indispensable para que el título obtenga la autorización del Ministerio correspondiente y su inclusión en el Registro de Universidades, Centros y Títulos (RUCT).

Este procedimiento incluye la evaluación del plan de estudios, los recursos disponibles, los mecanismos de calidad internos, la adecuación de las competencias propuestas y la alineación con los objetivos formativos. El proceso es gestionado por el Subdirector de Ordenación Académica, el Subdirector de Calidad y el Director del Centro.

Tras la verificación, el título alcanza el estado de *Verificado*, pero aún no ha comenzado a impartirse. Si no se inicia en un plazo de dos años, pasaría automáticamente a *En Extinción*, por lo que es crucial planificar su implantación eficazmente.

2.1.3 Implantación - Estados: En Implantación /Implantado

Durante la fase de implantación, el título pasa al estado *En Implantación*, donde el centro organiza los recursos académicos, docentes y materiales necesarios para poner en marcha el programa.

Una vez el título está presente en todos los cursos y en funcionamiento pleno, adquiere el estado de *Implantado*. Desde el comienzo de la implantación, se le aplican mecanismos de seguimiento y evaluación para garantizar su calidad y su capacidad de adaptación a las necesidades sociales y del mercado.

2.1.4 Modificación – Estado: Implantado

En la fase de vida activa de las titulaciones (*Implantado*), es habitual que estas requieran ajustes para adaptarse a nuevas realidades académicas, científicas o profesionales. El proceso de modificación permite actualizar el plan de estudios, manteniendo su calidad y adecuación. Estas modificaciones pueden ser de dos tipos: sustanciales (cambios significativos que requieren verificación externa) y no sustanciales (ajustes menores que pueden gestionarse internamente dentro de la universidad).

La normativa distingue también entre títulos adscritos a centros con acreditación institucional y aquellos sin ella, ya que los procedimientos y niveles de autonomía varían según el caso. Las modificaciones deben respetar el

calendario establecido, y no pueden solicitarse antes de un año desde el último informe emitido, salvo en el caso de modificaciones no sustanciales. Nuevamente, la responsabilidad recae en el Subdirector de Ordenación Académica, el Subdirector de Calidad y el Director del Centro.

2.1.5 Seguimiento – Estado: Implantado

El seguimiento es un proceso clave para evaluar de forma continua el desarrollo de las titulaciones, comprobar el cumplimiento de los objetivos previstos y detectar áreas de mejora. Puede ser ordinario (obligatorio o voluntario) o especial, en función de los resultados obtenidos en procesos previos de verificación, modificación o renovación de la acreditación.

El seguimiento ordinario obligatorio se realiza cuando una titulación está en su cuarto año de implantación o cuando en la renovación se han detectado criterios con calificaciones bajas (C o D). El voluntario, en cambio, puede ser solicitado por el centro para mejorar su evaluación en criterios concretos. El seguimiento especial es obligatorio cuando la agencia de calidad así lo determine por deficiencias observadas.

Este proceso implica la recopilación de indicadores, encuestas de satisfacción, tasas de rendimiento y resultados académicos. En función de los resultados obtenidos, se pueden proponer planes de mejora. En este caso, los responsables son el Técnico de Calidad del centro, el Subdirector de Calidad y el Director del Centro.

2.1.6 Renovación de la Acreditación – Estado: Implantado

Cada seis años, las titulaciones deben pasar de manera obligatoria por un proceso de renovación de la acreditación, que garantiza su vigencia y cumplimiento con los estándares de calidad actuales. Esta evaluación externa tiene como objetivo asegurar que el programa mantiene su adecuación a la normativa vigente, su pertinencia académica y su alineación con los objetivos y competencias declaradas.

La renovación implica una revisión detallada del título, incluyendo aspectos como la empleabilidad de los egresados, la actualización del profesorado, la suficiencia de los recursos y la evolución de los indicadores de calidad. El resultado puede permitir al título continuar en funcionamiento, someterse a

mejoras o, en algunos casos, iniciar su proceso de extinción. Los responsables son el Técnico de Calidad del centro, el Subdirector de Calidad y el Director del Centro.

Mientras este proceso se lleva a cabo, el título permanece en el estado *Implantado.*

2.1.7 Extinción del título– Estados: En Extinción/Extinguido

La extinción marca la fase final del ciclo de vida de un título universitario. Puede producirse por diversos motivos: decisión institucional, falta de implantación tras la verificación, resultados negativos en procesos de seguimiento o reacreditación, o por la necesidad de sustituir el programa por otro más adecuado a las necesidades actuales. Cuando se decide poner fin al ciclo de vida de un título, este entra en el estado En Extinción.

Este proceso debe realizarse de forma planificada y garantizando los derechos de los estudiantes ya matriculados, que podrán finalizar sus estudios. Durante esta fase, el título deja de admitir nuevos estudiantes, y se establece un calendario de extinción que asegure una transición ordenada. El proceso es responsabilidad del Subdirector de Ordenación Académica, el Subdirector de Calidad y el Director del Centro.

Finalmente, una vez completado este proceso, el título adquiere el estado *Extinguido*, lo que implica su eliminación definitiva del catálogo académico de la universidad.

2.2 Primera iteración: Análisis del contexto de uso y prototipo de alta fidelidad en Figma

Durante la primera iteración del trabajo realizado por Álvaro López Martínez [1], se centró el esfuerzo en comprender en profundidad el contexto de uso del sistema de gestión del ciclo de vida de las titulaciones universitarias en la UPM y en la elaboración de un prototipo de alta fidelidad que representara fielmente los principales requisitos identificados.

2.2.1 Análisis del contexto de uso

El análisis comenzó con una entrevista clave con el personal de la Unidad Técnica de Calidad de la Universidad Politécnica de Madrid. Esta entrevista permitió conocer de primera mano sus necesidades, problemas actuales y expectativas en relación con la gestión de titulaciones oficiales.

Además, se revisaron diversos documentos internos y externos de la universidad, entre los que destacan los que describen en detalle los procesos implicados en el ciclo de vida de un título: diseño de nuevos títulos, verificación, renovación de la acreditación, modificación, seguimiento externo y extinción. Esta documentación fue esencial para identificar los procedimientos normativos y las tareas específicas que deben ser gestionadas por la futura aplicación.

Asimismo, se mantuvieron reuniones periódicas con el Vicerrectorado de Calidad y Eficiencia, que permitieron obtener una visión más precisa de los procedimientos actuales y de los flujos de trabajo reales que debía contemplar el sistema. Gracias a este trabajo de análisis contextual, se pudieron definir los principales requisitos funcionales desde una perspectiva centrada en el usuario.

2.2.2 Identificación y priorización de casos de uso

A partir del análisis realizado, se definieron un total de 13 casos de uso, que representan las funcionalidades clave que debe ofrecer el sistema para cubrir adecuadamente los distintos procesos del ciclo de vida de un título. Estos casos de uso se representan en el diagrama de casos de uso de la *Ilustración 1.Diagrama de Casos de Uso* tomado del trabajo previo [1], conforme a la notación UML estándar. El diagrama permite visualizar de forma clara la relación entre los actores y las funcionalidades del sistema, facilitando la validación de requisitos y la planificación de futuras iteraciones.

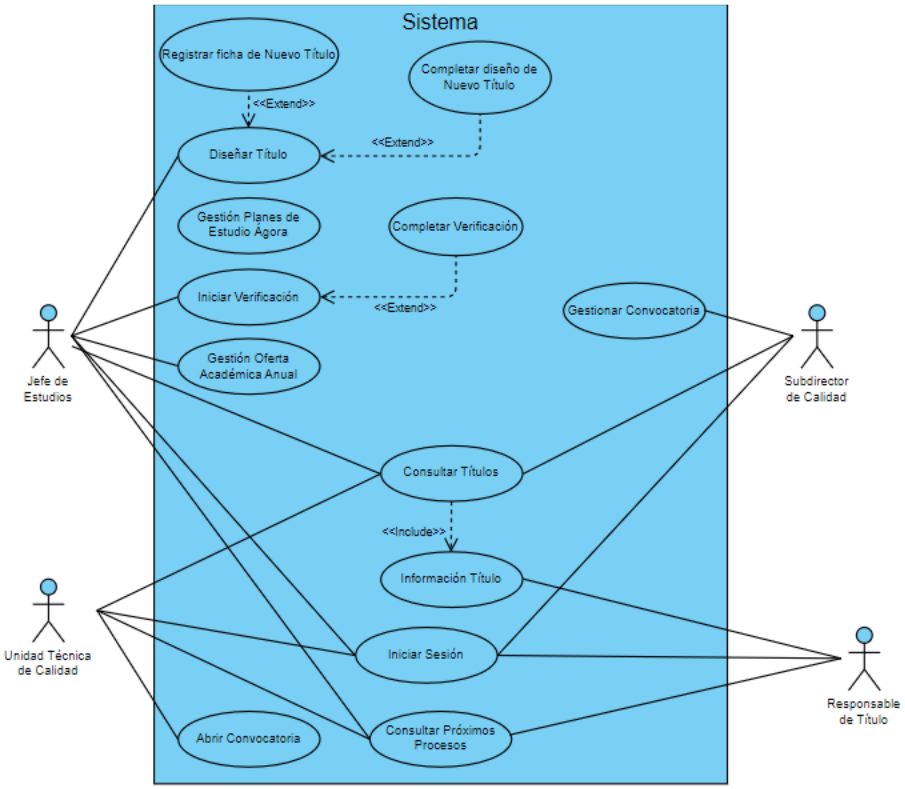


Ilustración 1.Diagrama de Casos de Uso

Dado el alcance del proyecto y los objetivos de esta primera iteración, se decidió dar prioridad al diseño y prototipado de los nueve primeros casos de uso, que constituyen el núcleo funcional del sistema. Estos son:

1. Iniciar Sesión
2. Diseñar Título
3. Consultar Títulos
4. Información Título
5. Iniciar Verificación
6. Abrir Convocatoria
7. Consultar Próximos Procesos
8. Gestionar Convocatoria de Seguimiento Externo
9. Gestionar Modificación

Los cuatro casos restantes se contemplaron como funcionalidades complementarias que podrían ser desarrolladas en iteraciones posteriores para ampliar las capacidades del sistema.

2.2.3 Prototipo de alta fidelidad en Figma

Con los casos de uso priorizados, se procedió al diseño de un prototipo de alta fidelidad utilizando la herramienta Figma. Este prototipo tiene como objetivo principal ofrecer una representación visual precisa del sistema, incluyendo tanto la estructura y disposición de la interfaz de usuario como la interacción entre pantallas.

El prototipo permite simular la navegación y el flujo de trabajo asociado a cada caso de uso, lo que resulta especialmente útil para identificar posibles problemas de usabilidad y para validar los requisitos antes de la implementación real.

Gracias a este enfoque iterativo basado en diseño centrado en el usuario, se logró una primera versión del sistema que recoge las necesidades reales de los distintos actores implicados en la gestión del ciclo de vida de las titulaciones universitarias en la UPM.

2.3 Segunda iteración: Selección de tecnologías, primer prototipo web y ampliación del prototipo en Figma

Durante la segunda iteración del proyecto realizada por Antonio Mendoza Carpintero [2] se avanzó significativamente en el desarrollo conceptual y visual del sistema. A partir de los trabajos realizados en la primera iteración, se diseñó un prototipo web de alta fidelidad, se realizó una primera selección de tecnologías para futuros desarrollos funcionales, y se amplió el prototipo en Figma incorporando nuevos casos de uso centrados en la gestión de la oferta académica.

2.3.1 Selección de tecnologías

En esta fase, se definió una arquitectura tecnológica preliminar basada en una estructura cliente-servidor. Se seleccionaron tecnologías ampliamente adoptadas en el desarrollo web moderno: React como biblioteca principal para el desarrollo del *frontend*, Node.js como entorno de ejecución para el *backend* y MySQL como sistema de gestión de base de datos relacional. Aunque en esta iteración no se desarrolló un sistema funcional, esta decisión tecnológica permitió establecer una base sólida para futuras implementaciones.

2.3.2 Primer prototipo web

El prototipo web creado durante esta fase se centró en representar de forma visual y estructurada algunos de los principales casos de uso identificados en la primera iteración. Se trataba de una interfaz de usuario navegable, pero sin conexión a lógica de negocio ni base de datos, es decir, un prototipo estático no funcional. Aun así, este entregable resultó clave para validar aspectos de usabilidad y diseño, y para facilitar la comprensión del sistema por parte de los diferentes actores implicados.

2.3.3 Ampliación del prototipo de Figma

En paralelo, se trabajó en la ampliación del prototipo en Figma centrado en la gestión de la oferta académica, una de las áreas clave del proyecto. Para ello, se mantuvieron reuniones con actores implicados (como el Vicerrectorado de Estrategia y Ordenación Académica (VREOA) y responsables de calidad) con el objetivo de comprender en profundidad los procesos actuales y detectar las necesidades específicas de mejora. Con esta información, se diseñaron nuevos flujos de trabajo y pantallas que daban soporte a tareas como la planificación anual de las titulaciones, la revisión y aprobación de propuestas académicas, y la coordinación entre distintos perfiles implicados.

En conjunto, esta segunda iteración supuso un progreso importante en la formalización de la interfaz y estructura del sistema, así como en la definición tecnológica que guiase futuras fases de desarrollo. Aunque lo producido aún no era funcional, se consolidaron las bases para una implementación técnica escalable y se amplió la visión del sistema hacia una solución más integral y alineada con las necesidades institucionales.

3 Tecnologías

3.1 Arquitectura general de la aplicación

La arquitectura de la aplicación sigue un modelo de arquitectura distribuida basada en la división entre cliente y servidor, estructurando el sistema en dos componentes funcionales claramente diferenciados: el *frontend* (interfaz de usuario) y el *backend* (lógica del servidor). Esta separación facilita no solo una mayor escalabilidad y mantenibilidad, sino también una implementación más flexible y modular, permitiendo abordar futuras ampliaciones con menor complejidad técnica.

La comunicación entre ambos componentes del sistema se lleva a cabo a través de una API REST, que establece un protocolo estándar para el intercambio de datos mediante peticiones HTTP. Esta interfaz actúa como nexo entre el cliente, que solicita recursos, y el servidor, que los procesa y responde en formato estructurado (JSON), garantizando un intercambio eficiente y en tiempo real.

3.2 Interfaz de usuario: Desarrollo frontend con React

El cliente ha sido desarrollado utilizando el framework React [3], una biblioteca de JavaScript orientada al desarrollo de interfaces dinámicas basadas en componentes. La estructura del *frontend* se organiza en componentes funcionales encapsulados, localizados en un árbol de directorios que facilita la reutilización y el mantenimiento del código. Cada componente representa una unidad funcional autónoma y se encarga de gestionar tanto la lógica de presentación como el estilo visual correspondiente.

La navegación dentro de la aplicación se gestiona mediante la librería React Router, lo que permite al usuario desplazarse entre las distintas funcionalidades sin provocar recargas completas de la página, optimizando así la experiencia interactiva. Esta funcionalidad proporciona una experiencia de usuario fluida, mejorando la eficiencia en la interacción con el sistema. Dentro de la aplicación este contenido se gestiona mediante el archivo App.js, que definirá las rutas de acceso a las distintas secciones.

Asimismo, React emplea internamente un mecanismo conocido como DOM virtual que permite actualizar solo aquellas partes de la interfaz que han cambiado tras una interacción del usuario. Esto desemboca en un rendimiento más eficiente, especialmente útil en aplicaciones donde las consultas, actualizaciones y cambios de estado se realizan de forma frecuente.

3.3 Lógica del servidor y persistencia de datos

La lógica de negocio de la aplicación se implementa en el *backend*, desarrollado sobre Node.js [4], un entorno de ejecución asincrónico que permite manejar múltiples conexiones de forma eficiente. Node.js se emplea para definir los *endpoints* de la API REST, validar los datos recibidos, ejecutar las operaciones necesarias sobre la base de datos y devolver las respuestas correspondientes al *frontend*.

En el sistema desarrollado, el archivo principal del servidor, *index.js*, actúa como punto de entrada del *backend*. En él se configura la conexión con la base de datos, se definen los *middleware* necesarios para procesar las peticiones, y se organiza el enrutamiento de los diferentes módulos funcionales. A nivel funcional, el *backend* define múltiples *endpoints* especializados, cada uno asociado a una tarea específica, como la obtención de titulaciones, la autenticación de usuarios o la modificación del estado de un título en concreto. Cada uno de estos puntos de acceso está diseñado para validar la entrada, ejecutar la lógica correspondiente y retornar los datos requeridos por el cliente.

Para la gestión de datos estructurados, se ha optado por el sistema gestor de bases de datos relacional MySQL [5], que permite organizar la información mediante tablas interrelacionadas, garantizando la integridad referencial mediante claves foráneas. El modelo de datos refleja fielmente el ciclo de vida de las titulaciones universitarias, abarcando entidades como Centros y Títulos entre otras.

Las operaciones con la base de datos mencionada anteriormente se realizan a través de sentencias SQL directas. Esta decisión responde a criterios de rendimiento y control, ya que las consultas manuales permiten una

optimización más precisa en operaciones complejas y reducen la sobrecarga añadida que implican las abstracciones propias de los ORM(*Object-Relational Mapping*).

Además, el diseño de la base de datos se ha llevado a cabo con un enfoque orientado a la escalabilidad, de manera que el sistema pueda adaptarse y ampliarse con facilidad en el futuro. Esta perspectiva arquitectónica permite la incorporación de nuevas tablas o la adición de campos a las estructuras existentes sin que ello implique una reestructuración significativa del modelo original. Tal característica resulta esencial en el contexto de un proyecto con perspectiva de crecimiento progresivo y previsiones de evolución funcional a medio y largo plazo.

3.4 Mecanismos de seguridad y gestión de archivos

Con el fin de garantizar la protección de los datos gestionados por la aplicación, se han incorporado mecanismos orientados a la seguridad y confidencialidad de la información, especialmente en lo que respecta a la gestión de usuarios y al manejo de archivos.

En primer lugar, se utiliza la biblioteca *bcrypt* [6], una herramienta especializada en la protección de credenciales. *Bcrypt* aplica un algoritmo de *hash* criptográfico diseñado para dificultar la recuperación de la contraseña original a partir del valor almacenado. Cada contraseña introducida por el usuario es transformada mediante este algoritmo antes de ser almacenada en la base de datos. Este enfoque impide el almacenamiento de contraseñas en texto plano y protege al sistema frente a ataques de tipo *rainbow table* o *brute force*.

Por otro lado, se ha incorporado la herramienta *Multer* [7], un *middleware* específico para Node.js que facilita la gestión de archivos enviados mediante formularios web. Esta biblioteca permite definir con precisión los criterios de validación de los archivos (como su tamaño máximo, tipo MIME o nombre), establecer su ruta de almacenamiento temporal o permanente, y capturar sus metadatos de forma segura. *Multer* es especialmente útil en funcionalidades

donde se requiere que los usuarios puedan subir documentos adjuntos, como convocatorias, resoluciones u otro tipo de información académica.

Ambas herramientas contribuyen de forma decisiva a la robustez del sistema, reforzando tanto la privacidad de los datos como la estabilidad del proceso de carga y almacenamiento de archivos.

3.5 Interacción y flujo de datos entre frontend y backend

El flujo de información entre la interfaz de usuario y el servidor sigue un modelo de comunicación síncrona mediante peticiones HTTP. Cuando el usuario realiza una acción en el *frontend* (por ejemplo, solicitar los títulos activos en un centro), se genera una petición que es enviada a través de la API al *backend*, el cual procesa la lógica asociada, accede a la base de datos y responde con los datos solicitados.

Este patrón de funcionamiento, basado en el paradigma *request-response*, garantiza que los datos presentados en la interfaz estén siempre actualizados, proporcionando una experiencia de usuario interactiva, coherente y en tiempo real. Las operaciones más comunes se canalizan a través de los métodos GET (para recuperación de información), POST (para creación de recursos), PUT (para actualizaciones) y DELETE (para eliminación), siguiendo los principios de la arquitectura REST.

La estructura modular del sistema y su diseño desacoplado permiten realizar modificaciones en cualquiera de los componentes (por ejemplo, cambios en la base de datos o mejoras en la interfaz) sin que estas alteraciones afecten negativamente al resto del sistema.

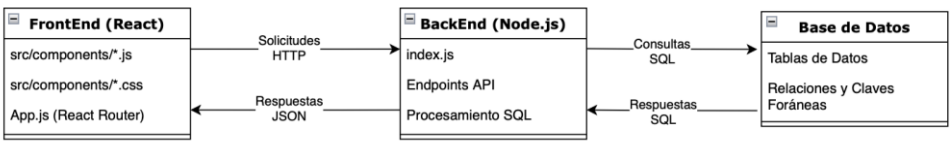


Ilustración 2. Diagrama de Integración entre Frontend, Backend y Base de Datos

4 Desarrollo

4.1 Diseño e implementación de un subsistema de gestión de usuarios y roles

El subsistema de gestión de usuarios y roles constituye un componente esencial de la aplicación, ya que permite controlar el acceso a las distintas funcionalidades del sistema en función del perfil del usuario. Este diseño se fundamenta en la necesidad de garantizar tanto la seguridad de la información como una asignación clara de responsabilidades en el entorno académico.

4.1.1 Definición de roles en la aplicación

La aplicación implementa un sistema de control de acceso basado en roles, diseñado para garantizar que cada usuario interactúe únicamente con las funcionalidades correspondientes a sus competencias dentro de la estructura organizativa de la universidad. Este enfoque, facilita una distribución clara de responsabilidades y tareas, mejorando la trazabilidad de las acciones realizadas por cada perfil.

A continuación, se describen los distintos roles definidos en el sistema, así como las capacidades asociadas a cada uno de ellos:

- **Jefe de Estudios:** Este rol posee amplias competencias en materia de diseño, consulta y administración de títulos universitarios dentro de su centro. En primer lugar, tiene la capacidad de iniciar el proceso de creación de un nuevo título, introduciendo todos los datos requeridos para su diseño inicial. Asimismo, puede completar la información de títulos ya existentes que se encuentren en fase de diseño.

  En segundo lugar, este usuario puede consultar en detalle tanto los títulos vigentes como aquellos ya extinguidos, limitándose su alcance a los programas implantados en su propio centro. Dicha consulta incluye la totalidad de la información asociada a cada título, lo que resulta esencial para una gestión académica informada.

  Además, el jefe de estudios tiene acceso a la planificación de los procesos de calidad a los que se someterán los títulos implantados, tales como el seguimiento externo o la renovación de acreditaciones.

Finalmente, este rol dispone de funcionalidades específicas para la administración de los responsables de título: puede consultar los actuales responsables asignados, acceder a un histórico detallado de todos aquellos que hayan desempeñado dicha función en el pasado —junto con las fechas asociadas—, y registrar nuevos responsables para los títulos vigentes en su centro.

- **Subdirector de Calidad:** Este rol está orientado a funciones de supervisión y gestión de los procesos de calidad dentro del centro correspondiente. Tiene acceso a la consulta de títulos vigentes y extinguidos correspondientes a su centro, pudiendo también visualizar toda la información asociada a los mismos.

  De igual modo, tiene la posibilidad de consultar y gestionar las convocatorias abiertas vinculadas a procesos de seguimiento externo, renovación de acreditación o renovación de sellos de calidad. Asimismo, puede consultar la planificación futura de dichos procesos para los títulos implantados en su centro, lo que le permite anticipar y preparar adecuadamente las acciones institucionales pertinentes.
- **Unidad Técnica de Calidad:** Este perfil tiene un alcance transversal a nivel institucional. Puede consultar la lista de títulos, tanto vigentes como extinguidos, de todos los centros de la universidad, accediendo a sus respectivos detalles.

  Asimismo, está capacitado para iniciar nuevas convocatorias de evaluación —ya sea de seguimiento externo, renovación de acreditación o renovación de sello—, garantizando con ello la continuidad de los procesos de aseguramiento de la calidad.

  Además, puede consultar los próximos procesos previstos para todos los títulos implantados en la institución, lo que le otorga una visión global y estratégica del estado de la calidad académica en la universidad.
- **Responsable de Título:** Este perfil está limitado a los títulos específicos sobre los que ejerce responsabilidad. Puede consultar tanto los títulos vigentes como extinguidos bajo su supervisión, accediendo a la información detallada de cada uno. Además, tiene la posibilidad de consultar los próximos procesos de calidad que afecten a dichos títulos.

En síntesis, el diseño del sistema de roles no solo se articula en función de los distintos niveles jerárquicos y funcionales dentro de la universidad, sino que también responde a una lógica de especialización y delimitación de responsabilidades. Esta arquitectura permite preservar la integridad del sistema, mejorar su escalabilidad organizativa y asegurar que cada usuario actúe conforme a sus atribuciones, favoreciendo así una gestión académica más eficiente, segura y transparente.

4.1.2 Proceso de autenticación de usuarios

El proceso de autenticación constituye un componente fundamental del sistema, ya que garantiza que el acceso a la aplicación esté limitado únicamente a usuarios válidos y autorizados según su rol. Este mecanismo se ha desarrollado en dos capas diferenciadas: el cliente (*frontend)*, implementado en React, y el servidor (*backend*), construido sobre Node.js con una base de datos MySQL como sistema de almacenamiento persistente.

4.1.2.1 Lógica del cliente (Frontend)

Desde el punto de vista del usuario, el acceso al sistema se realiza a través de un formulario de inicio de sesión*(Ilustración 3. Formulario inicio de sesión*). Este formulario incluye tres elementos clave: un campo para introducir el correo electrónico, otro para la contraseña y un desplegable para elegir el rol correspondiente (jefe de estudios, subdirector de calidad, unidad técnica de calidad o responsable de título). Esta estructura garantiza que el proceso de autenticación esté condicionado no solo por las credenciales del usuario, sino también por el tipo de acceso requerido.

Ilustración 3. Formulario inicio de sesión

El componente "*Login*", implementado como componente funcional de React, mantiene el estado de estos tres campos mediante los *hooks* "*useState*". Al hacer clic en el botón de inicio de sesión, se ejecuta la función "*handleLogin*", que encapsula la lógica principal del proceso:

1. **Validación Inicial:** Se comprueba que ninguno de los campos esté vacío. En caso contrario, se lanza una alerta informativa al usuario.

```
const handleLogin = async () => {
  if (!perfil || !correo || !contrasena) {
    Create Jira Issue
    alert('Por favor, complete todos los campos.');
    return;
  }
```

Ilustración 4. Código validación inicial

2. **Envío de Solicitud:** Si los datos están completos, se realiza una solicitud POST al *endpoint* "*/api/login*", enviando como cuerpo un objeto JSON con el correo electrónico, la contraseña y el rol seleccionados.

```
try {
  const response = await fetch('http://localhost:3001/api/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email: correo, password: contrasena, rol: perfil }),
  });

  const data = await response.json();
```

Ilustración 5. Código envío de solicitud login

3. **Gestión de Respuesta:** Si la autenticación es correcta (es decir, el servidor responde con un estado 200 OK), se almacena en el *localStorage* del navegador la información relevante del usuario (correo, centro y rol).

```
// Guardamos en localStorage
localStorage.setItem('userEmail', data.user.email);
localStorage.setItem('userCentro', data.user.centro);
localStorage.setItem('userRole', data.user.rol);
```

Ilustración 6. Código gestión de respuesta login

4. **Redirección:** A continuación, el sistema redirige al usuario a la ruta correspondiente según su rol mediante el *hook useNavigate*.

```
  switch (data.user.rol) {
    case 'jefe-estudios':
      navigate('/jefe-estudios');
      break;
    case 'subdirector-calidad':
      navigate('/subdirector-calidad');
      break;
    case 'utc':
      navigate('/utc');
      break;
    case 'responsable-titulo':
      navigate('/responsable-titulo');
      break;
    default:
      alert('Rol no reconocido');
  }
} else {
  alert(data.error || 'Error de autenticación');
}
```

Ilustración 7. Código redirección login

Una vez completado el proceso de autenticación y realizada la redirección a la ruta correspondiente en función del rol seleccionado —tal como se detalla en la *Ilustración 7. Código redirección login*—, se presenta al usuario una interfaz inicial adaptada a sus privilegios. Dicha interfaz varía en función del rol, mostrando únicamente las funcionalidades que le han sido habilitadas, tal como se ilustra en las imágenes expuestas a continuación.

Ilustración 8. Interfaz de inicio para el rol Jefe de Estudios

En la imagen superior, -véase *Ilustración 8. Interfaz de inicio para el rol Jefe de Estudios*- se muestra el panel principal accesible a los usuarios con el rol de jefe de estudios. Desde esta vista, pueden gestionar el diseño de nuevos títulos, consultar títulos vigentes o extinguidos, consultar los próximos procesos, así como administrar los responsables de título asociados a su centro.

Ilustración 9.Interfaz de inicio para el rol Subdirector de Calidad

En esta pantalla – *véase Ilustración 9.Interfaz de inicio para el rol Subdirector de Calidad* - se presentan las funcionalidades disponibles para el subdirector de calidad, centradas en la supervisión de títulos y convocatorias de su centro y la

gestión de procesos de calidad como seguimiento externo o renovación de acreditaciones.

Ilustración 10. Interfaz de inicio para el rol Unidad Técnica de Calidad.

En la pantalla superior -*véase Ilustración 10. Interfaz de inicio para el rol Unidad Técnica de Calidad.*- representa el entorno de trabajo de los usuarios pertenecientes a la Unidad Técnica de Calidad de la Universidad Politécnica de Madrid (UPM), con acceso centralizado a la información de todos los títulos implantados en la universidad, así como a la apertura y seguimiento de convocatorias de evaluación de calidad.

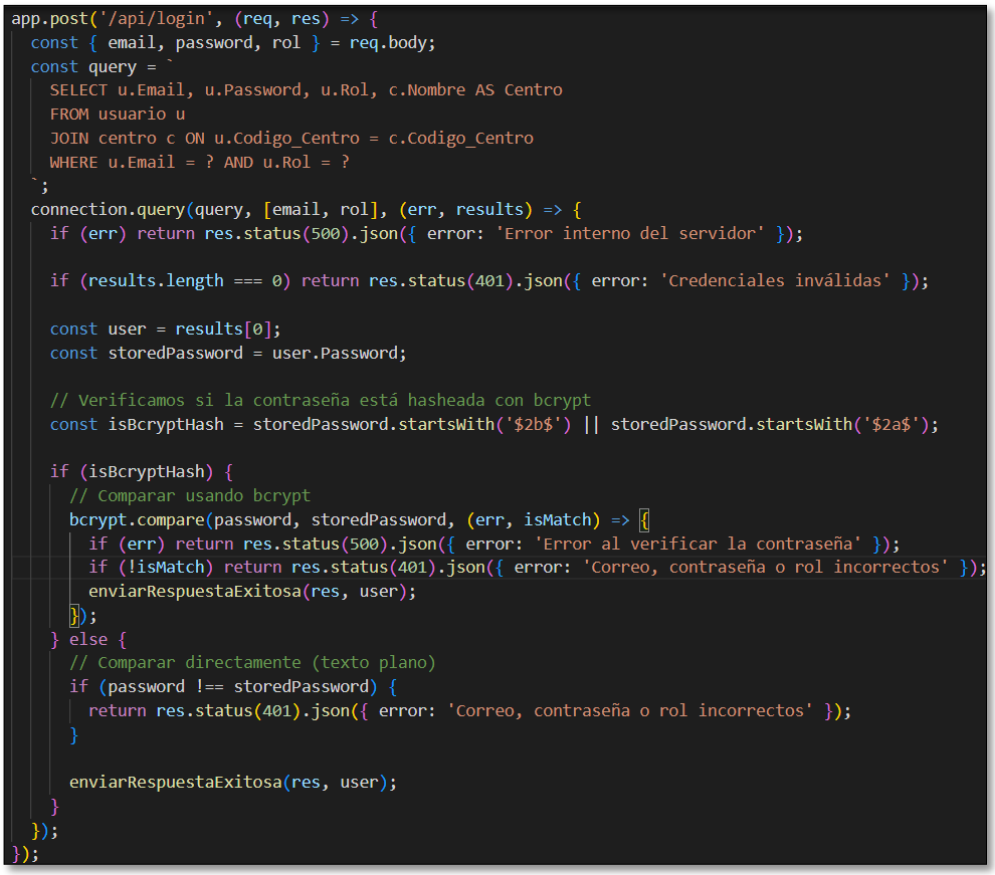
Ilustración 11. Interfaz de inicio para el rol Responsable de Título.

Esta vista -*véase Ilustración 11. Interfaz de inicio para el rol Responsable de Título.*-permite al responsable de título acceder a los datos relevantes de los títulos a su cargo, incluyendo detalles académicos y próximos procesos de calidad en los que se verán involucrados.

4.1.2.2 Lógica del backend

En el servidor, se ha implementado una ruta POST en */api/login* - *véase Ilustración 12. Código backend login-* utilizando *Express.js*. Esta ruta realiza las siguientes operaciones:

1. **Extracción de Datos:** Obtiene el correo electrónico, la contraseña y el rol del cuerpo de la solicitud.
2. **Consulta a la Base de Datos:** Ejecuta una consulta SQL para recuperar el usuario que coincida con el correo electrónico y el rol proporcionados.
3. **Verificación de Contraseña:**
   - Si la contraseña almacenada está *hasheada* con *bcrypt* (identificada por los prefijos $2b$, $2a$ o $2y$), utiliza *bcrypt.compare* para verificar la coincidencia.
   - Si la contraseña está en texto plano (casuística que se da la primera vez que se le asigna una contraseña a un usuario haciéndola coincidir con su email), se realiza una comparación directa porque aún no ha sido cifrada. Esto es debido a que la contraseña no se *hashea* inicialmente para que el usuario pueda acceder una primera vez al sistema y modificarla por una definitiva. Tras el primer acceso, al cambiar la contraseña, esta sí se cifra y desde entonces todas las comparaciones se hacen mediante *hash*.
4. **Respuesta al Cliente:** Si la autenticación es exitosa, responde con un objeto JSON que incluye el correo electrónico, el centro y el rol del usuario.



```
app.post('/api/login', (req, res) => {
  const { email, password, rol } = req.body;
  const query = `
    SELECT u.Email, u.Password, u.Rol, c.Nombre AS Centro
    FROM usuario u
    JOIN centro c ON u.Codigo_Centro = c.Codigo_Centro
    WHERE u.Email = ? AND u.Rol = ?
  `;
  connection.query(query, [email, rol], (err, results) => {
    if (err) return res.status(500).json({ error: 'Error interno del servidor' });

    if (results.length === 0) return res.status(401).json({ error: 'Credenciales inválidas' });

    const user = results[0];
    const storedPassword = user.Password;

    // Verificamos si la contraseña está hasheada con bcrypt
    const isBcryptHash = storedPassword.startsWith('$2b$') || storedPassword.startsWith('$2a$');

    if (isBcryptHash) {
      // Comparar usando bcrypt
      bcrypt.compare(password, storedPassword, (err, isMatch) => {
        if (err) return res.status(500).json({ error: 'Error al verificar la contraseña' });
        if (!isMatch) return res.status(401).json({ error: 'Correo, contraseña o rol incorrectos' });
        enviarRespuestaExitosa(res, user);
      });
    } else {
      // Comparar directamente (texto plano)
      if (password !== storedPassword) {
        return res.status(401).json({ error: 'Correo, contraseña o rol incorrectos' });
      }

      enviarRespuestaExitosa(res, user);
    }
  });
});
```

Ilustración 12. Código backend login

Este enfoque combina prácticas de seguridad modernas, como el uso de *bcrypt* para el hash de contraseñas, con una lógica clara y estructurada para la autenticación de usuarios.

4.1.3 Gestión del cambio de contraseña

Con el objetivo de garantizar la seguridad y permitir la autogestión de credenciales por parte del usuario, la aplicación incorpora una funcionalidad específica para el cambio de contraseña. Esta opción está accesible desde la pantalla de inicio de cada rol – véase *Ilustración 8. Interfaz de inicio para el rol Jefe de Estudios*-, mediante un botón ubicado en la barra lateral con el título "Cambiar contraseña". Al seleccionar dicha opción, se despliega una nueva vista que contiene un formulario compuesto por dos campos de entrada: uno para la contraseña actual y otro para la nueva contraseña que el usuario desea establecer, tal como se muestra en *Ilustración 13.Interfaz de formulario para el cambio de contraseña.*

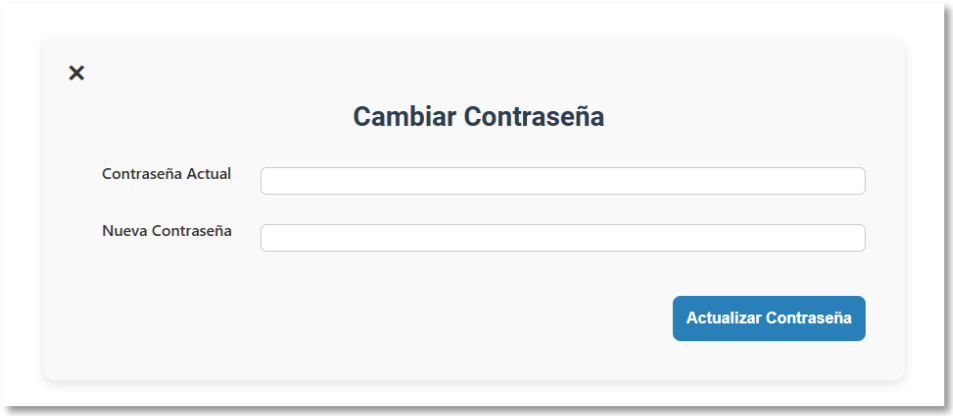


Ilustración 13.Interfaz de formulario para el cambio de contraseña

Desde el punto de vista funcional, la lógica del cliente está implementada en el componente *CambiarContrasena.js*. En este se gestionan tres estados locales: *contrasenaActual*, *nuevaContrasena* y *mensaje*, a través del *hook useState* de *React*. Cuando el usuario completa el formulario y pulsa el botón "Actualizar Contraseña", se ejecuta la función *handleSubmit*, la cual evita el comportamiento por defecto del formulario (*e.preventDefault()*) y realiza una petición POST al *endpoint /api/cambiar-contrasena*.

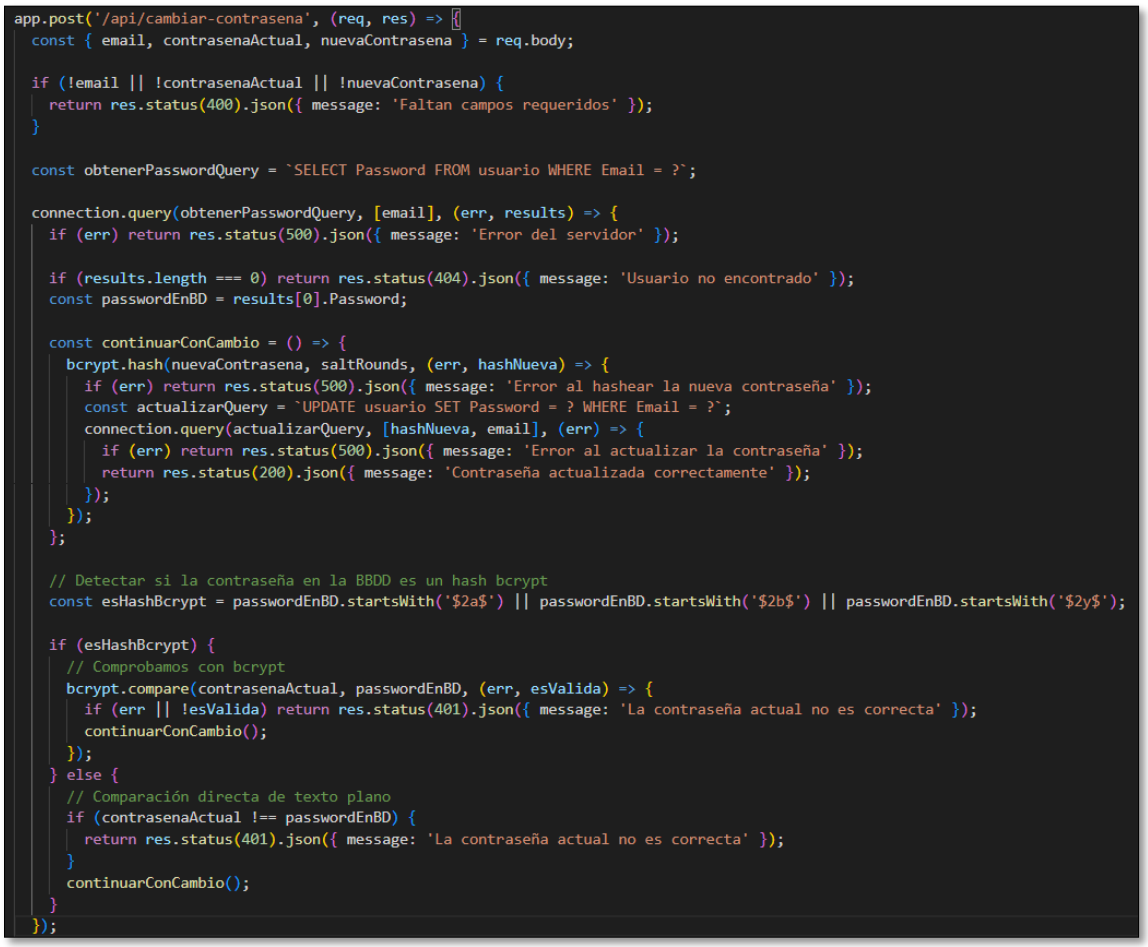
Dicha petición incluye en el cuerpo los siguientes parámetros: el correo electrónico del usuario autenticado (obtenido del almacenamiento local del navegador) y las contraseñas actual y nueva. En función de la respuesta recibida desde el servidor, se actualiza el estado *mensaje*, informando al usuario del resultado de la operación mediante un mensaje contextual.

En el lado del servidor, el *backend* implementa el *endpoint* correspondiente mediante *Express.js*. La ruta /api/cambiar-contrasena – *véase Ilustración 14. Código backend Cambio de contraseña*- recibe los datos enviados por el cliente y, en primer lugar, valida que no falte ninguno de los campos requeridos. Posteriormente, se recupera de la base de datos la contraseña asociada al usuario utilizando la dirección de correo electrónico proporcionada.

El sistema contempla dos posibles escenarios respecto al formato de la contraseña almacenada: si esta se encuentra en texto plano, se compara directamente con el valor ingresado por el usuario; si, por el contrario, está

cifrada con el algoritmo *bcrypt* (identificable por los prefijos $2a$, $2b$, o $2y$), se utiliza la función *bcrypt.compare()* para verificar su validez.

Una vez validada correctamente la contraseña actual, el servidor procede a cifrar la nueva contraseña mediante *bcrypt.hash()* con un número de rondas de sal (*saltRounds*) definido como 10. Finalmente, se actualiza la contraseña cifrada en la base de datos mediante una sentencia UPDATE, y se devuelve una respuesta al cliente indicando el éxito de la operación.



```
app.post('/api/cambiar-contrasena', (req, res) => {
  const { email, contrasenaActual, nuevaContrasena } = req.body;

  if (!email || !contrasenaActual || !nuevaContrasena) {
    return res.status(400).json({ message: 'Faltan campos requeridos' });
  }

  const obtenerPasswordQuery = `SELECT Password FROM usuario WHERE Email = ?`;

  connection.query(obtenerPasswordQuery, [email], (err, results) => {
    if (err) return res.status(500).json({ message: 'Error del servidor' });

    if (results.length === 0) return res.status(404).json({ message: 'Usuario no encontrado' });
    const passwordEnBD = results[0].Password;

    const continuarConCambio = () => {
      bcrypt.hash(nuevaContrasena, saltRounds, (err, hashNueva) => {
        if (err) return res.status(500).json({ message: 'Error al hashear la nueva contraseña' });
        const actualizarQuery = `UPDATE usuario SET Password = ? WHERE Email = ?`;
        connection.query(actualizarQuery, [hashNueva, email], (err) => {
          if (err) return res.status(500).json({ message: 'Error al actualizar la contraseña' });
          return res.status(200).json({ message: 'Contraseña actualizada correctamente' });
        });
      });
    };

    // Detectar si la contraseña en la BBDD es un hash bcrypt
    const esHashBcrypt = passwordEnBD.startsWith('$2a$') || passwordEnBD.startsWith('$2b$') || passwordEnBD.startsWith('$2y$');

    if (esHashBcrypt) {
      // Comprobamos con bcrypt
      bcrypt.compare(contrasenaActual, passwordEnBD, (err, esValida) => {
        if (err || !esValida) return res.status(401).json({ message: 'La contraseña actual no es correcta' });
        continuarConCambio();
      });
    } else {
      // Comparación directa de texto plano
      if (contrasenaActual !== passwordEnBD) {
        return res.status(401).json({ message: 'La contraseña actual no es correcta' });
      }
      continuarConCambio();
    }
  });
```

Ilustración 14. Código backend Cambio de contraseña

4.1.4 Registro de Nuevos Responsables de Título (Funcionalidad del Jefe de Estudios)

En el presente apartado se describe detalladamente el proceso de registro de nuevos responsables de título, funcionalidad clave dentro del sistema de gestión académica desarrollado. Esta funcionalidad permite al Jefe de Estudios asignar o modificar los responsables de las titulaciones impartidas en el centro. La implementación de esta característica implica tanto lógica de presentación (cliente) como lógica de negocio y persistencia de datos (servidor). A continuación, se describe cada una de estas capas.

4.1.4.1 Lógica del cliente (Frontend)

Desde el punto de vista del usuario final, el acceso a la funcionalidad de registro de responsables se realiza a través de la interfaz correspondiente al perfil del Jefe de Estudios. Dentro de dicha vista, se habilita una sección específica denominada Administración, que centraliza la gestión de los responsables de título. Esta vista se compone principalmente de dos elementos: una tabla de responsables actuales y un histórico de responsables anteriores -véase *Ilustración 15. Interfaz vista Responsables de Titulo-.*

- **Responsables de Título Actuales:** Tabla que muestra todos aquellos usuarios que, en el momento de la consulta, están asignados como responsables de alguna titulación. Para cada registro se visualizan campos como el nombre, apellidos, correo electrónico institucional y la denominación del título coordinado.
- **Histórico de Responsables de Título:** Tabla que permite consultar los usuarios que han desempeñado este rol con anterioridad, incluyendo la fecha de inicio y de finalización de su responsabilidad. Además, se proporciona una barra de búsqueda que permite filtrar los registros del histórico por título, lo que facilita la navegación por registros antiguos.

Ilustración 15. Interfaz vista Responsables de Titulo

Desde esta misma pantalla, el jefe de estudios puede pulsar el botón "Registrar Responsable de Título", lo que le llevará a un formulario -*véase Ilustración 16. Interfaz formulario Registro de Responsable de Título* - donde podrá registrar un nuevo coordinador. Este formulario solicita los datos necesarios para registrar un nuevo responsable: nombre, apellidos, dirección de correo electrónico institucional y título que coordinará. La lista de títulos disponibles se carga dinámicamente a partir de una consulta al servidor, filtrando únicamente aquellos títulos asociados al centro del Jefe de Estudios.

Ilustración 16. Interfaz formulario Registro de Responsable de Título

Desde el punto de vista técnico, la lógica de esta funcionalidad está desarrollada en React, utilizando el *hook useEffect* – *véase Ilustración 17. Código lógica frontend tablas Responsable Titulo-* para la carga inicial de datos mediante peticiones *fetch* al servidor. Estas peticiones recuperan tanto los responsables activos como el histórico, ejecutándose de forma paralela para optimizar el rendimiento. El filtrado por título del histórico se implementa en el cliente mediante una operación de filtrado sobre el estado local.

La acción de envío del formulario genera una solicitud HTTP de tipo POST al *backend*, en la que se encapsulan los datos introducidos por el usuario. Si la operación concluye correctamente, se presenta una ventana emergente de confirmación que notifica al usuario que el nuevo responsable ha sido registrado con éxito.

```
const AdministracionJefeEstudios = () => {
    const [responsables, setResponsables] = useState([]); // Lista de responsables
    const [historicoResponsables, setHistoricoResponsables] = useState([]); // Lista de historico de responsables
    const [filtroTitulo, setFiltroTitulo] = useState('');

    const navigate = useNavigate();

    useEffect(() => {
        const fetchData = async () => {
          try {
            // Hacer ambas peticiones en paralelo
            const [responseResponsables, responseHistoricoResponsables] = await Promise.all([
              fetch('http://localhost:3001/api/responsables'), // Obtener responsables actuales con rol 'responsable-titulo'
              fetch('http://localhost:3001/api/historico-responsables'), // Obtener responsables antiguos con rol 'responsable-titulo'
            ]);

            // Verificar que ambas respuestas sean correctas
            if (!responseResponsables.ok) throw new Error('Error al obtener responsables');
            if (!responseHistoricoResponsables.ok) throw new Error('Error al obtener el historico de responsables');

            // Convertir las respuestas a JSON
            const responsablesData = await responseResponsables.json();
            const historicoResponsablesData = await responseHistoricoResponsables.json();

            // Guardar los datos en el estado
            setResponsables(responsablesData);
            setHistoricoResponsables(historicoResponsablesData);
          } catch (error) {
            console.error('Error al obtener los datos:', error);
          }
        };

        fetchData();
      }, []); // Se ejecuta solo al montar el componente
```

Ilustración 17. Código lógica frontend tablas Responsable Titulo

4.1.4.2 Lógica del backend

La lógica del servidor asociada al registro de nuevos responsables se ha diseñado para garantizar la consistencia de los datos y el cumplimiento de las reglas de negocio propias del modelo académico. Se han implementado varios

endpoints que permiten recuperar información y registrar nuevos responsables de forma segura y trazable.

En primer lugar, el servidor expone un conjunto de rutas GET que permiten recuperar los datos necesarios para la visualización inicial de la interfaz -véase *Ilustración 15. Interfaz vista Responsables de Titulo*-. Estas rutas incluyen:

- **GET /api/responsables:** Devuelve los responsables de título actuales, aquellos cuya relación de coordinación no ha sido finalizada (campo Fecha_Fin nulo).
- **GET /api/historico-responsables:** Devuelve un listado de antiguos responsables, es decir, registros cuya coordinación ya ha concluido (campo Fecha_Fin no nulo).
- **GET /api/titulos/por-centro:** Devuelve la lista de titulaciones impartidas por el centro del usuario autenticado, permitiendo así mostrar un desplegable contextualizado en el formulario de registro.

```
//Ruta para obtener los titulos asociados a un centro
app.get('/api/titulos/por-centro', (req, res) => {
  const nombreCentro = req.query.nombre;

  const query = `
    SELECT t.Codigo_Propuesto, t.Nombre
    FROM titulo t
    JOIN centro_imparte ci ON t.Codigo_Propuesto = ci.Codigo_Propuesto
    JOIN centro c ON ci.Codigo_Centro = c.Codigo_Centro
    WHERE c.Nombre = ?
  `;

  connection.query(query, [nombreCentro], (err, results) => {
    if (err) {
      console.error('Error al obtener los títulos por centro:', err);
      return res.status(500).send('Error al obtener los títulos');
    }
    res.json(results);
  });
});
```

Ilustración 18. Código ruta GET /api/titulos/por-centro

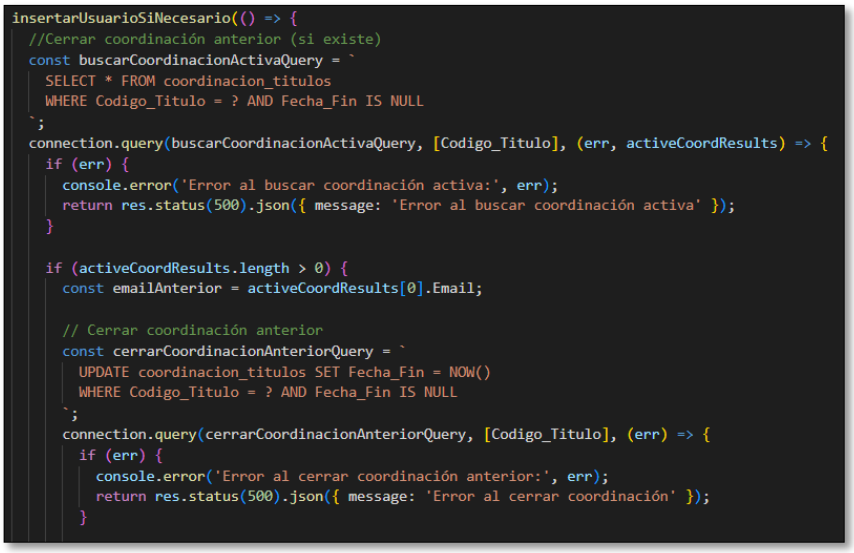
El procesamiento del registro de un nuevo responsable se realiza a través de un *endpoint* de tipo **POST** identificado como **/api/responsables-titulo**. Este *endpoint* lleva a cabo una serie de operaciones encadenadas que aseguran la correcta actualización del sistema. La secuencia lógica es la siguiente:

1. **Verificación de la existencia del usuario:** El sistema comprueba si el correo electrónico introducido ya está registrado en las tablas *usuario* y *personal*. En caso de no existir, se crean nuevas entradas correspondientes.

```
//Insertar en "personal" si no existe
const verificarPersonalQuery = 'SELECT * FROM personal WHERE Email = ?';
connection.query(verificarPersonalQuery, [Email], (err, personalResults) => {
  if (err) {
    console.error('Error al verificar personal:', err);
    return res.status(500).json({ message: 'Error del servidor' });
  }

  const insertarUsuarioYCoordinacion = () => {
    //Insertar en "usuario" si no existe con ese rol
    const verificarUsuarioQuery = 'SELECT * FROM usuario WHERE Email = ? AND Rol = ?';
    connection.query(verificarUsuarioQuery, [Email, Rol], (err, usuarioResults) => {
      if (err) {
        console.error('Error al verificar usuario:', err);
        return res.status(500).json({ message: 'Error del servidor' });
      }

      const insertarUsuarioSiNecesario = (callback) => {
        if (usuarioResults.length === 0) {
          const insertarUsuarioQuery = `
            INSERT INTO usuario (Email, Password, Rol, Codigo_Centro)
            VALUES (?, ?, ?, ?)
          `;
          connection.query(insertarUsuarioQuery, [Email, Password, Rol, Codigo_Centro], (err) => {
            if (err) {
              console.error('Error al insertar en usuario:', err);
              return res.status(500).json({ message: 'Error al insertar en usuario' });
            }
            callback();
          });
        } else {
          callback(); // Ya existe, continuar
        }
      };
```

Ilustración 19.Código verificar existencia usuario

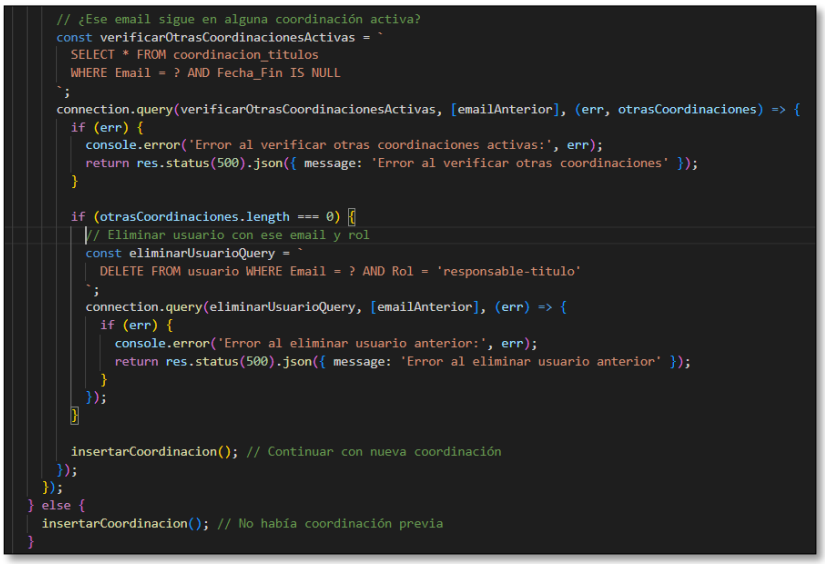
2. **Finalización de la coordinación anterior:** Si existe un responsable actualmente asignado a la titulación seleccionada, se actualiza su entrada en la tabla *coordinacion_titulos*, fijando la fecha de finalización al día actual.



```
insertarUsuarioSiNecesario(() => {
  //Cerrar coordinación anterior (si existe)
  const buscarCoordinacionActivaQuery = `
    SELECT * FROM coordinacion_titulos
    WHERE Codigo_Titulo = ? AND Fecha_Fin IS NULL
  `;
  connection.query(buscarCoordinacionActivaQuery, [Codigo_Titulo], (err, activeCoordResults) => {
    if (err) {
      console.error('Error al buscar coordinación activa:', err);
      return res.status(500).json({ message: 'Error al buscar coordinación activa' });
    }

    if (activeCoordResults.length > 0) {
      const emailAnterior = activeCoordResults[0].Email;

      // Cerrar coordinación anterior
      const cerrarCoordinacionAnteriorQuery = `
        UPDATE coordinacion_titulos SET Fecha_Fin = NOW()
        WHERE Codigo_Titulo = ? AND Fecha_Fin IS NULL
      `;
      connection.query(cerrarCoordinacionAnteriorQuery, [Codigo_Titulo], (err) => {
        if (err) {
          console.error('Error al cerrar coordinación anterior:', err);
          return res.status(500).json({ message: 'Error al cerrar coordinación' });
        }
```

Ilustración 20. Código finalización coordinación anterior

3. **Depuración de usuarios inactivos:** Si el responsable saliente no tiene ninguna otra coordinación activa, se procede a eliminar su cuenta de usuario de la tabla usuario con el rol responsable-titulo, ya que deja de formar parte del sistema con este rol (un usuario puede estar de manera activa en el sistema con distintos roles simultáneamente).



```
        // ¿Ese email sigue en alguna coordinación activa?
        const verificarOtrasCoordinacionesActivas = `
          SELECT * FROM coordinacion_titulos
          WHERE Email = ? AND Fecha_Fin IS NULL
        `;
        connection.query(verificarOtrasCoordinacionesActivas, [emailAnterior], (err, otrasCoordinaciones) => {
          if (err) {
            console.error('Error al verificar otras coordinaciones activas:', err);
            return res.status(500).json({ message: 'Error al verificar otras coordinaciones' });
          }

          if (otrasCoordinaciones.length === 0) {
            // Eliminar usuario con ese email y rol
            const eliminarUsuarioQuery = `
              DELETE FROM usuario WHERE Email = ? AND Rol = 'responsable-titulo'
            `;
            connection.query(eliminarUsuarioQuery, [emailAnterior], (err) => {
              if (err) {
                console.error('Error al eliminar usuario anterior:', err);
                return res.status(500).json({ message: 'Error al eliminar usuario anterior' });
              }
            });
          }

          insertarCoordinacion(); // Continuar con nueva coordinación
        });
      });
    } else {
      insertarCoordinacion(); // No había coordinación previa
    }
```

Ilustración 21. Código depuración de usuarios inactivos

4. **Registro del nuevo responsable:** Finalmente, se inserta una nueva entrada en la tabla *coordinacion_titulos*, con el correo electrónico del

nuevo responsable, el código de la titulación y la fecha de inicio establecida en la fecha actual del sistema.

```
function insertarCoordinacion() {
  const verificarCoordinacionQuery = `
    SELECT * FROM coordinacion_titulos
    WHERE Email = ? AND Codigo_Titulo = ? AND Fecha_Inicio = CURDATE()
  `;

  connection.query(verificarCoordinacionQuery, [Email, Codigo_Titulo], (err, coordinacionResults) => {
    if (err) {
      console.error('Error al verificar coordinación:', err);
      return res.status(500).json({ message: 'Error al verificar coordinación' });
    }

    if (coordinacionResults.length > 0) {
      return res.status(400).json({ message: 'Ya existe una coordinación para ese usuario y título con la fecha de inicio actual' });
    }

    const insertarCoordinacionQuery = `
      INSERT INTO coordinacion_titulos (Email, Fecha_Inicio, Codigo_Titulo)
      VALUES (?, NOW(), ?)
    `;

    connection.query(insertarCoordinacionQuery, [Email, Codigo_Titulo], (err) => {
      if (err) {
        console.error('Error al insertar en coordinacion_titulos:', err);
        return res.status(500).json({ message: 'Error al insertar en coordinación' });
      }

      return res.status(201).json({ message: 'Responsable registrado correctamente y asignado al título' });
    });
  });
```

Ilustración 22. Código registro nuevo responsable

Este flujo garantiza que en todo momento exista un único responsable activo por titulación y que los históricos se mantengan actualizados, permitiendo así consultas posteriores con total trazabilidad.

4.2 Diseño e implementación del subsistema de diseño de un título nuevo

4.2.1 Visualización de títulos en proceso de diseño

Como parte del subsistema dedicado a la creación de nuevos títulos universitarios, se ha implementado una funcionalidad que permite consultar de forma centralizada aquellos títulos cuyo estado actual corresponde a las fases de “En Diseño” o “En Verificación”. Esta vista resulta esencial para que los usuarios con funciones de planificación académica (jefes de estudios) puedan revisar el estado y los metadatos básicos de las titulaciones en desarrollo, facilitando así la supervisión y la toma de decisiones.

Desde el punto de vista del usuario, al acceder a esta sección, se presenta una tabla interactiva que muestra todos los títulos en fase de diseño – véase *Ilustración 23. Interfaz visualización de títulos en proceso de diseño*-. Cada fila de la tabla representa un título individual e incluye las siguientes columnas:

- Nombre del título
- Tipo de título (Grado, Máster o Doctorado)
- Centro responsable de la titulación
- Estado actual de la propuesta (En Diseño o En Verificación)

Además, el usuario dispone de controles desplegables (filtros) que le permiten refinar la consulta según el tipo de título, el centro responsable y el estado. Estos filtros se aplican en tiempo real, y la interfaz se actualiza dinámicamente según los criterios seleccionados. Asimismo, si el usuario hace clic sobre cualquier fila, es redirigido a una vista detallada del título correspondiente. Finalmente, un botón destacado permite iniciar el proceso de diseño de una nueva titulación.

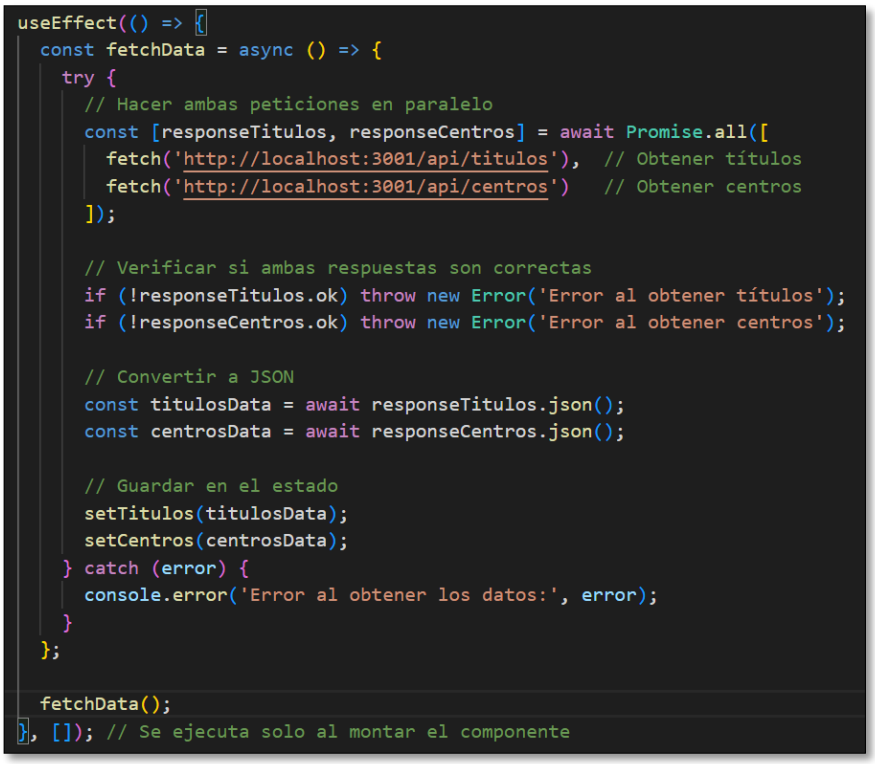
Ilustración 23. Interfaz visualización de títulos en proceso de diseño

4.2.1.1 Lógica del cliente (Frontend)

El componente principal encargado de la funcionalidad de la interfaz de usuario para la visualización de títulos en estado de diseño es ConsultarTitulosDiseno, que encapsula tanto la lógica de obtención y filtrado de datos como la presentación visual de los mismos.

Desde el momento en que el componente es montado, se lanza un efecto (*useEffect*) – véase *Ilustración 24. Código tabla títulos en diseño*- que ejecuta de forma asíncrona la recuperación de los datos necesarios para renderizar la tabla. Concretamente, se realizan dos peticiones HTTP paralelas: una para obtener la lista completa de títulos y otra para recuperar los centros universitarios disponibles en la base de datos. Estos datos se almacenan en los estados locales

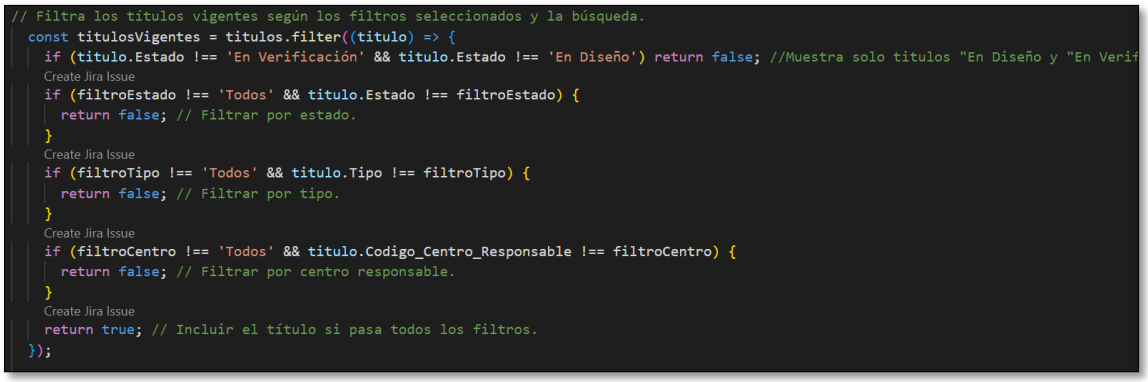
del componente (*useState*) y se utilizan tanto para mostrar la información como para poblar los menús desplegables de filtrado.



```
useEffect(() => {
  const fetchData = async () => {
    try {
      // Hacer ambas peticiones en paralelo
      const [responseTitulos, responseCentros] = await Promise.all([
        fetch('http://localhost:3001/api/titulos'),  // Obtener títulos
        fetch('http://localhost:3001/api/centros')   // Obtener centros
      ]);

      // Verificar si ambas respuestas son correctas
      if (!responseTitulos.ok) throw new Error('Error al obtener títulos');
      if (!responseCentros.ok) throw new Error('Error al obtener centros');

      // Convertir a JSON
      const titulosData = await responseTitulos.json();
      const centrosData = await responseCentros.json();

      // Guardar en el estado
      setTitulos(titulosData);
      setCentros(centrosData);
    } catch (error) {
      console.error('Error al obtener los datos:', error);
    }
  };

  fetchData();
}, []); // Se ejecuta solo al montar el componente
```

Ilustración 24. Código tabla títulos en diseño

Una vez que los datos están disponibles, se aplica un proceso de filtrado en el lado cliente que permite al usuario restringir los resultados según tres criterios: el tipo de título (Grado, Máster o Doctorado), el centro responsable y el estado del título (En Diseño o En Verificación). Este filtrado se realiza en tiempo real, de forma reactiva, de modo que la interfaz se actualiza automáticamente al modificar cualquiera de los filtros -véase *Ilustración 25. Código filtrado tabla títulos en diseño*-.



```
// Filtra los títulos vigentes según los filtros seleccionados y la búsqueda.
const titulosVigentes = titulos.filter((titulo) => {
  if (titulo.Estado !== 'En Verificación' && titulo.Estado !== 'En Diseño') return false; //Muestra solo titulos "En Diseño y "En Verif
  Create Jira Issue
  if (filtroEstado !== 'Todos' && titulo.Estado !== filtroEstado) {
    return false; // Filtrar por estado.
  }
  Create Jira Issue
  if (filtroTipo !== 'Todos' && titulo.Tipo !== filtroTipo) {
    return false; // Filtrar por tipo.
  }
  Create Jira Issue
  if (filtroCentro !== 'Todos' && titulo.Codigo_Centro_Responsable !== filtroCentro) {
    return false; // Filtrar por centro responsable.
  }
  Create Jira Issue
  return true; // Incluir el título si pasa todos los filtros.
});
```

Ilustración 25. Código filtrado tabla títulos en diseño

La representación visual de los títulos se materializa en una tabla HTML estilizada mediante una hoja de estilos propia (ConsultarTitulosDiseno.css). Cada fila de la tabla corresponde a un título concreto y muestra, de manera clara y estructurada, su nombre, tipo, centro responsable y estado actual. Para mejorar la experiencia visual, el estado del título se colorea con estilos específicos según su valor, facilitando así una interpretación rápida por parte del usuario.

Además de la visualización pasiva, el componente ofrece interactividad. Cada fila de la tabla es clicable y actúa como enlace hacia la vista detallada del título correspondiente. Esta funcionalidad se implementa mediante el *hook useNavigate* de React Router, que permite redirigir programáticamente al usuario hacia la ruta */titulos-diseno/:codigo*, donde se mostrará la información ampliada del título seleccionado.

Como parte complementaria, se incluye un botón situado en la parte inferior de la vista que permite iniciar el proceso de diseño de un nuevo título. Al hacer clic en este botón, el usuario es redirigido al formulario de creación de una nueva titulación mediante la ruta */disenar-titulo*.

Por último, en caso de que no existan títulos que cumplan con los criterios de búsqueda, se muestra un mensaje claro e informativo dentro de la propia tabla, indicando al usuario que actualmente no hay títulos en proceso de diseño que coincidan con los filtros seleccionados. Esta decisión de diseño busca evitar vacíos visuales y mejorar la comunicación con el usuario final.

4.2.1.2 Lógica del servidor (Backend)

Para respaldar esta funcionalidad en el servidor, se han definido dos *endpoints* REST que proporcionan los datos requeridos:

- **GET /api/titulos:** Se utiliza una consulta SQL que recupera todos los títulos desde la base de datos, incluyendo el nombre del centro responsable, utilizando una cláusula JOIN. Esta consulta es intencionadamente amplia, ya que el filtrado por estado se realiza en el *frontend*.

```
// Ruta para obtener todos los títulos (general)
app.get('/api/titulos', (req, res) => {

  const query = `
  SELECT t.*, c.Nombre AS Nombre_Centro
  FROM titulo t
  JOIN centro c ON t.Codigo_Centro_Responsable = c.Codigo_Centro;
  `;
  connection.query(query, (err, results) => {
    if (err) {
      console.error('Error al obtener los títulos:', err);
      return res.status(500).send('Error al obtener los títulos');
    }
    res.json(results);
  });
});
```

Ilustración 26. Código ruta /api/titulos

- **GET/api/centros:** Se define una segunda ruta que devuelve todos los centros académicos registrados, cuya información es empleada para poblar el filtro de centros en el *frontend*.

```
// Ruta para obtener los centros
app.get('/api/centros', (req, res) => {
  connection.query('SELECT * FROM Centro', (err, results) => {
    if (err) {
      console.error('Error al obtener los centros:', err);
      return res.status(500).send('Error al obtener los centros');
    }
    res.json(results);
  });
});
```

Ilustración 27. Código ruta /api/centros

4.2.2 Diseño de un nuevo título

Con el fin de facilitar a los jefes de estudios el inicio del proceso de creación de nuevas titulaciones, la aplicación incorpora una funcionalidad específica

accesible desde la vista principal de títulos en diseño. A través del botón "Diseñar nuevo título", situado en la parte inferior de dicha vista – véase *Ilustración 23. Interfaz visualización de títulos en proceso de diseño*-, se habilita el acceso a un formulario estructurado que permite introducir los datos generales de una propuesta de titulación. Esta operación está restringida al ámbito del centro al que pertenece el jefe de estudios autenticado, garantizando así la integridad organizativa del proceso.

Desde el punto de vista de la experiencia de usuario, el formulario presenta un diseño claro e intuitivo, con campos bien diferenciados y validaciones básicas que guían al usuario durante el proceso de entrada de datos -véase *Ilustración 28. Formulario diseño nuevo título*.-. El jefe de estudios deberá cumplimentar obligatoriamente el nombre del título, seleccionar su nivel académico (Grado, Máster o Doctorado), e introducir el número de créditos ECTS que conformarán la titulación (en el caso del Doctorado se introducirá el valor "0" debido a que este tipo de titulaciones carece de créditos ECTS). Por su parte, el campo correspondiente al centro responsable se presenta de forma no editable, ya que se rellena automáticamente a partir de la información almacenada en el sistema de autenticación (obtenida de *localStorage*) y está asociada al perfil del usuario en sesión.

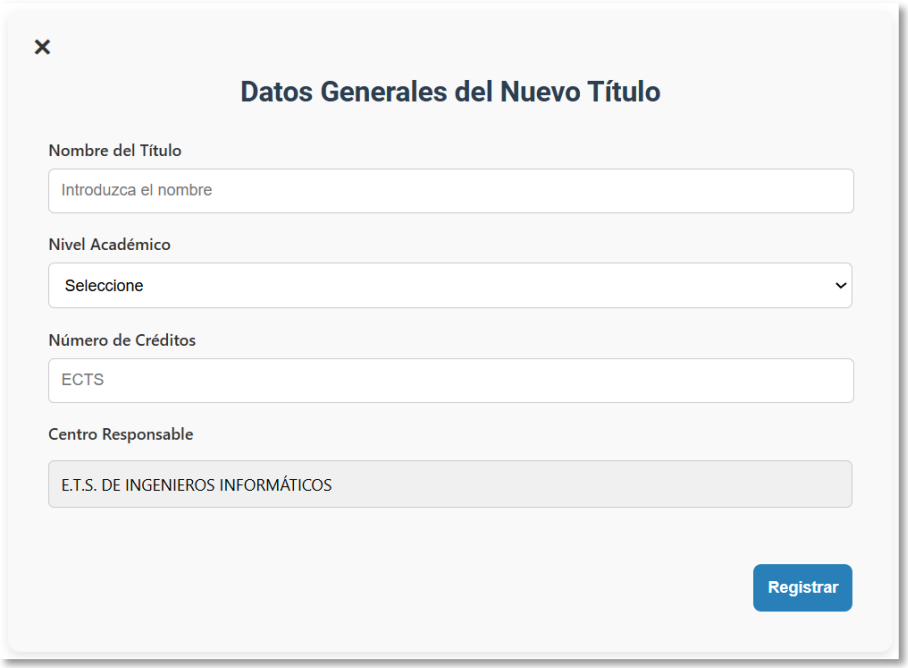
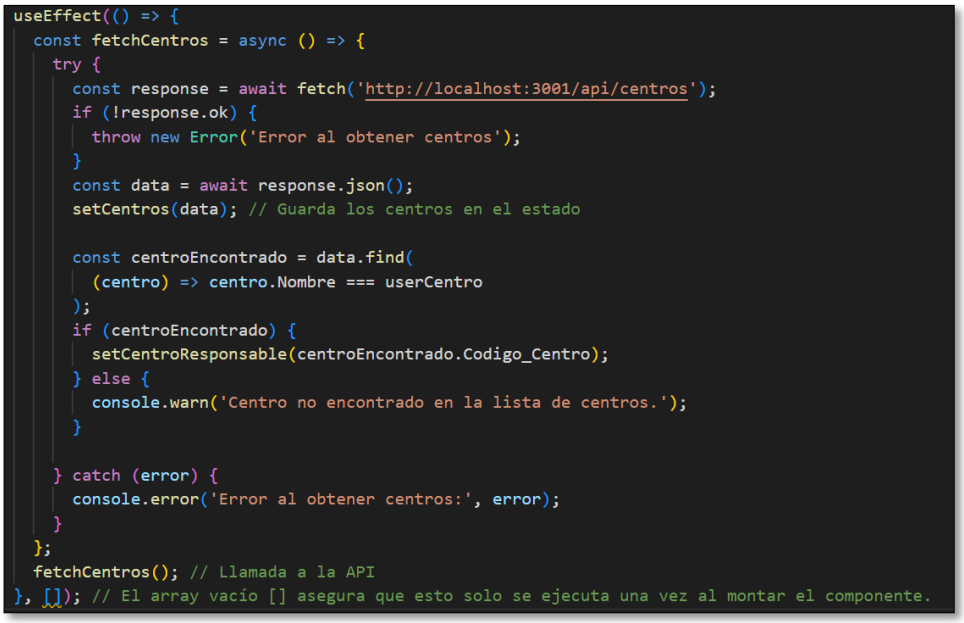


Ilustración 28. Formulario diseño nuevo título.

4.2.2.1 Lógica del cliente (frontend)

La vista de diseño del título está implementada como un componente funcional de React denominado DisenarTitulo. En este componente se gestionan los distintos estados del formulario mediante *useState*, mientras que el *hook useEffect* -véase *Ilustración 29.Código useEffect de DisenarTitulo*- se emplea para realizar una carga inicial de los centros universitarios desde la API y validar que el centro correspondiente al jefe de estudios se encuentra registrado.

Durante la inicialización, el componente recupera de forma asíncrona la lista completa de centros mediante una llamada GET a la ruta /api/centros. Esta información permite validar que el nombre del centro guardado localmente (obtenido tras el inicio de sesión) corresponde con un centro existente en la base de datos. Si se encuentra una coincidencia, se extrae y almacena su código, que será incluido en la petición de creación del título.



```
useEffect(() => {
  const fetchCentros = async () => {
    try {
      const response = await fetch('http://localhost:3001/api/centros');
      if (!response.ok) {
        throw new Error('Error al obtener centros');
      }
      const data = await response.json();
      setCentros(data); // Guarda los centros en el estado

      const centroEncontrado = data.find(
        (centro) => centro.Nombre === userCentro
      );
      if (centroEncontrado) {
        setCentroResponsable(centroEncontrado.Codigo_Centro);
      } else {
        console.warn('Centro no encontrado en la lista de centros.');
      }

    } catch (error) {
      console.error('Error al obtener centros:', error);
    }
  };
  fetchCentros(); // Llamada a la API
}, []); // El array vacío [] asegura que esto solo se ejecuta una vez al montar el componente.
```

Ilustración 29.Código useEffect de DisenarTitulo

Al pulsar el botón *"Registrar"*, se ejecuta la función *handleSubmit* -véase *Ilustración 30. Fragmento código función handleSubmit en DisenoTitulo*-, que previene el comportamiento por defecto del formulario y construye un objeto que representa el nuevo título. Entre los datos generados dinámicamente se encuentra un código de identificación único (codigo_propuesto), así como la fecha actual (fecha_propuesta), generada en formato YYYY-MM-DD. Además, el título se crea con estado inicial "En Diseño" y subestado "Fase Inicial".

Una vez conformado el objeto, se realiza una petición HTTP de tipo POST a la ruta /api/titulos, enviando la información en formato *JSON*. Si la operación es exitosa, se notificará al usuario mediante un popup de confirmación visual y se procederá a limpiar los campos del formulario. Además, si el componente padre ha proporcionado una función de actualización de la lista de títulos, esta será invocada para reflejar inmediatamente el nuevo registro en la tabla "Títulos en Diseño".

```
// Función para manejar el envío del formulario
const handleSubmit = async (e) => {
  e.preventDefault(); // Previene el comportamiento por defecto del formulario

  // Generación de un código único para el título propuesto
  const codigoPropuesto = `P${Math.random().toString(36).substring(2, 8).toUpperCase()}`;

  // Creación del objeto con los datos del nuevo título propuesto
  const nuevoTituloPropuesto = {
    codigo_propuesto: codigoPropuesto,
    nombre: nombreTitulo,
    tipo: nivelAcademico,
    codigo_centro_responsable: centroResponsable,
    fecha_propuesta: new Date().toISOString().split('T')[0], // Fecha actual en formato YYYY-MM-DD
    creditos: Number(creditos), // Convierte los créditos a un número
    estado: 'En Diseño', // Estado inicial del título
    subestado: 'Fase Inicial',
  };

  console.log("Datos que se enviarán:", JSON.stringify(nuevoTituloPropuesto, null, 2));
  try {
    // Llamada a la API para enviar los datos del título
    const response = await fetch('http://localhost:3001/api/titulos', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(nuevoTituloPropuesto), // Convierte el objeto a JSON
    });
```

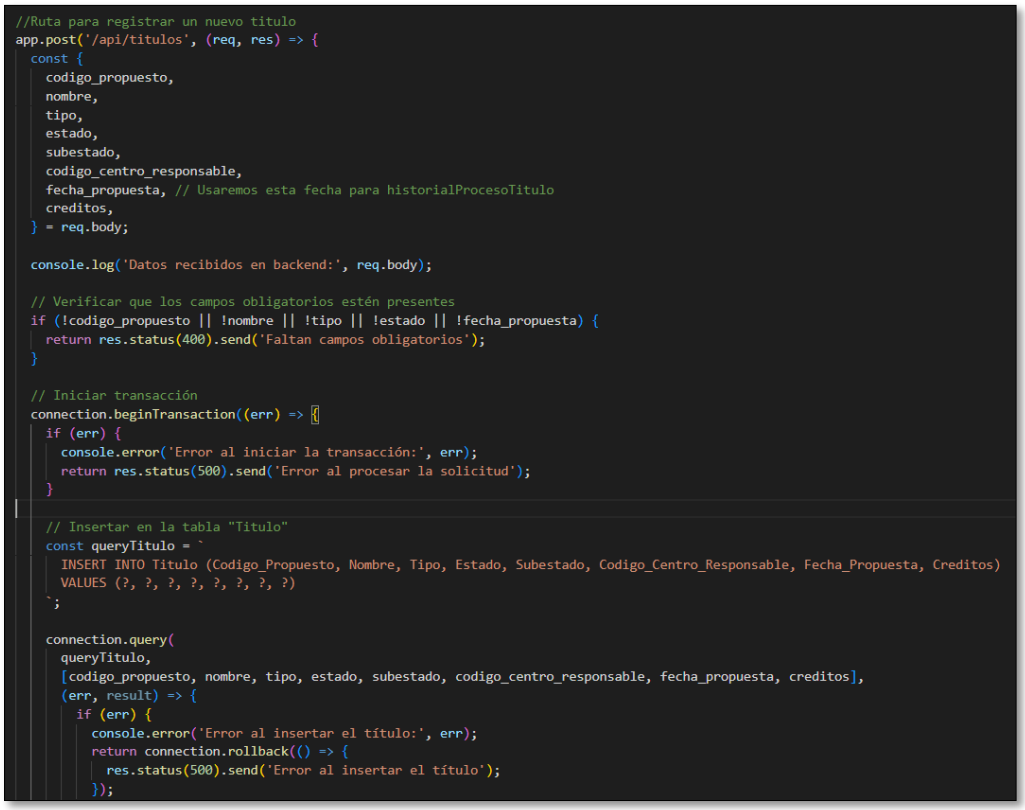
Ilustración 30. Fragmento código función handleSubmit en DisenoTitulo

4.2.2.2 Lógica del servidor (backend)

El servidor, implementado con *Node.js* y MySQL, expone dos rutas relevantes para esta funcionalidad. La primera, **GET /api/centros** -véase *Ilustración 27. Código ruta /api/centros*-, se encarga de retornar la lista completa de centros registrados en la base de datos. Esta información es utilizada por el *frontend* para validar y asociar correctamente el código del centro al nuevo título.

La segunda ruta, **POST /api/titulos**, permite registrar un nuevo título en el sistema. A través de esta ruta, el servidor recibe los datos enviados por el formulario, los valida y los inserta en la base de datos utilizando una transacción SQL para asegurar la consistencia de la operación. La transacción consta de dos etapas:

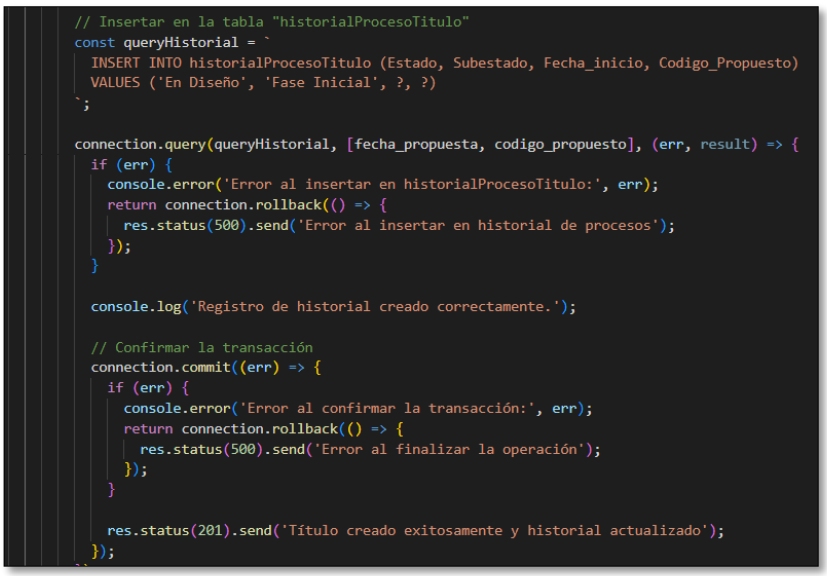
1. **Inserción del título en la tabla Titulo**, utilizando los campos proporcionados (nombre, tipo, código de centro, fecha de propuesta, créditos, estado y subestado).



```
//Ruta para registrar un nuevo titulo
app.post('/api/titulos', (req, res) => {
  const {
    codigo_propuesto,
    nombre,
    tipo,
    estado,
    subestado,
    codigo_centro_responsable,
    fecha_propuesta, // Usaremos esta fecha para historialProcesoTitulo
    creditos,
  } = req.body;

  console.log('Datos recibidos en backend:', req.body);

  // Verificar que los campos obligatorios estén presentes
  if (!codigo_propuesto || !nombre || !tipo || !estado || !fecha_propuesta) {
    return res.status(400).send('Faltan campos obligatorios');
  }

  // Iniciar transacción
  connection.beginTransaction((err) => {
    if (err) {
      console.error('Error al iniciar la transacción:', err);
      return res.status(500).send('Error al procesar la solicitud');
    }

    // Insertar en la tabla "Titulo"
    const queryTitulo = `
      INSERT INTO Titulo (Codigo_Propuesto, Nombre, Tipo, Estado, Subestado, Codigo_Centro_Responsable, Fecha_Propuesta, Creditos)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)
    `;

    connection.query(
      queryTitulo,
      [codigo_propuesto, nombre, tipo, estado, subestado, codigo_centro_responsable, fecha_propuesta, creditos],
      (err, result) => {
        if (err) {
          console.error('Error al insertar el título:', err);
          return connection.rollback(() => {
            res.status(500).send('Error al insertar el título');
          });
```

Ilustración 31. Código POST /api/titulos parte 1

2. **Registro histórico del inicio del proceso** en la tabla historialProcesoTitulo, donde se documenta el estado inicial del título ("En Diseño") y se asocia con la fecha de creación y su código.



```
        // Insertar en la tabla "historialProcesoTitulo"
        const queryHistorial = `
          INSERT INTO historialProcesoTitulo (Estado, Subestado, Fecha_inicio, Codigo_Propuesto)
          VALUES ('En Diseño', 'Fase Inicial', ?, ?)
        `;

        connection.query(queryHistorial, [fecha_propuesta, codigo_propuesto], (err, result) => {
          if (err) {
            console.error('Error al insertar en historialProcesoTitulo:', err);
            return connection.rollback(() => {
              res.status(500).send('Error al insertar en historial de procesos');
            });
          }

          console.log('Registro de historial creado correctamente.');

          // Confirmar la transacción
          connection.commit((err) => {
            if (err) {
              console.error('Error al confirmar la transacción:', err);
              return connection.rollback(() => {
                res.status(500).send('Error al finalizar la operación');
              });
            }

            res.status(201).send('Título creado exitosamente y historial actualizado');
          });
```

Ilustración 32. Código POST /api/titulos parte 2

Si ambas operaciones se completan correctamente, se confirma la transacción y se retorna una respuesta satisfactoria al cliente. En caso de error en alguna de las fases, la transacción se revierte para evitar inconsistencias en la base de datos, y se devuelve un mensaje de error apropiado.

4.2.3 Visualización del detalle del Título en Diseño

Esta vista constituye el núcleo del proceso de gestión y supervisión de títulos en fase de diseño. Permite al jefe de estudios consultar, editar y completar toda la información y documentación asociada a un nuevo título universitario propuesto. Su acceso se realiza desde la tabla general de títulos en diseño –véase *Ilustración 23. Interfaz visualización de títulos en proceso de diseño*-: al hacer clic sobre el nombre de un título, se redirige al usuario a esta interfaz detallada -véase *Ilustración 33. Interfaz detalle de título en diseño*-.

Ilustración 33. Interfaz detalle de título en diseño

Desde esta pantalla, se ofrece una representación estructurada del flujo completo de creación y aprobación del título, dividiendo el proceso en secciones bien definidas que corresponden con las distintas etapas del ciclo de vida del diseño: “Información general”, “Aprobación inicial por la escuela”, “Aprobación inicial por la universidad”, “Aprobación de memoria por la escuela”, “Aprobación de memoria por la universidad”, “Espera de verificación” y “Título verificado”.

4.2.3.1 Estructura general de la vista

La interfaz está diseñada para ser clara, guiada y progresiva. Se basa en una estructura de navegación por pestañas o secciones, controladas mediante un conjunto de botones visibles en la parte superior del componente. Cada botón permite al usuario acceder a una de las fases del proceso de aprobación del título. Estas secciones están implementadas como componentes "React" condicionalmente renderizados a partir del estado *seccionActiva*, lo que permite alternar entre vistas sin abandonar el contexto del título actual -véase *Ilustración 34.Código navegación botones DetallesTituloDiseno*-.

El flujo es secuencial y validado: las secciones solo pueden ser activadas si se ha completado la anterior. Esta restricción se gestiona mediante una serie de estados booleanos (*aprobacionEscuelaCompleta*, *aprobacionUniversidadCompleta*, *memoriaEscuelaCompleta*, etc.) que reflejan el progreso del título y el atributo *"disabled*". De este modo, se fuerza a que los pasos se completen de forma ordenada, evitando inconsistencias u omisiones.

La primera sección, "Información General", se encuentra siempre disponible inicialmente, y en ella se muestra un resumen de los datos fundamentales del título (nombre, código, centros responsables, estado, etc.). A partir de aquí, el usuario puede navegar por las siguientes etapas, siempre que cumplan las condiciones necesarias.

```jsx
{/* Botones de navegación para cambiar entre secciones */}
<div className="botones-secciones">
  {/* Botón: Información General */}
  <button
    onClick={() => cambiarSeccion('informacion')}
    className={seccionActiva === 'informacion' ? 'activo' : ''}
  >
    Información General
  </button>

  {/* Botón: Aprobación Inicial Escuela */}
  <button
    onClick={() => cambiarSeccion('aprobacionInicialEscuela')}
    className={seccionActiva === 'aprobacionInicialEscuela' ? 'activo' : ''}
  >
    Aprobación Inicial Escuela
  </button>

  {/* Botón: Aprobación Inicial Universidad */}
  <button
    onClick={() => cambiarSeccion('aprobacionInicialUniversidad')}
    className={seccionActiva === 'aprobacionInicialUniversidad' ? 'activo' : ''}
    disabled={!aprobacionEscuelaCompleta} // Se deshabilita hasta que la anterior esté completa
  >
```

Ilustración 34.Código navegación botones DetallesTituloDiseno

Cada sección de aprobación permite la visualización de documentos específicos, la subida de nuevos archivos, y en muchos casos, el registro de fechas clave, siempre en función del estado del título. Estas funcionalidades están respaldadas por diversos estados internos (documentos, actas, propuestas, etc.) que permiten tanto mostrar como almacenar información de manera reactiva y organizada.

4.2.3.2 Sección: Información General

La sección Información General proporciona una vista consolidada de los principales datos descriptivos del título en diseño – véase *Ilustración 33. Interfaz detalle de título en diseño*-. Esta es la primera pestaña accesible para el usuario al consultar los detalles de un título específico, y constituye la base sobre la cual se inicia el proceso de diseño del título.

El usuario accede a esta sección clicando en el titulo del cual desea obtener información de los disponibles en la tabla "Títulos en Diseño". En primer lugar, se le muestran los datos generales del título, tales como su denominación oficial, el nivel académico al que corresponde (grado, máster o doctorado), el número total de créditos ECTS, y el centro responsable designado. Estos datos se presentan en un formato de lectura clara, sin posibilidad de edición directa – véase *Ilustración 33. Interfaz detalle de título en diseño*-.

A continuación, se ofrece un apartado interactivo donde el usuario puede visualizar los distintos centros académicos que participan en la impartición del título. Junto a cada uno de ellos aparece una opción para eliminar su asociación. Además, mediante un desplegable que muestra todos los centros disponibles no asociados aún al título, se permite al usuario seleccionar y añadir nuevos centros a la propuesta actual.

Finalmente, se incluye una tabla cronológica que refleja el historial de estados administrativos por los que ha pasado el título desde su creación. Para cada entrada en el historial se especifica el estado alcanzado, un posible subestado y la fecha en la que dicho cambio se hizo efectivo. Esta información resulta fundamental para comprender la evolución y trazabilidad del expediente académico del título.

La lógica de presentación y comportamiento de esta sección se implementa utilizando React. La vista se activa cuando el estado seccionActiva toma el valor 'informacion', lo que permite encapsular toda la funcionalidad en un único bloque condicional. Dentro de este componente se accede al estado titulo, que contiene los datos generales, así como a los arrays centrosAsociados, centrosDisponibles y historial, todos ellos actualizados mediante llamadas asincrónicas a la API.

Para visualizar los datos generales del título los cuales se muestran de manera estática se utiliza la función fetchTitulo – véase *Ilustración 35. Código función fetchTitulo en DetallesTituloDiseño*-, invocada en un useEffect.

```
//Función para obtener la información del título
const fetchTitulo = async () => {
  try {
    const responseTitulo = await fetch(`http://localhost:3001/api/titulos/${codigo}`);
    if (!responseTitulo.ok) throw new Error("Error al obtener los detalles del título");

    const dataTitulo = await responseTitulo.json();
    console.log("Detalles del título:", dataTitulo);
    setTitulo(dataTitulo);
  } catch (error) {
    console.error("Error al cargar el título:", error);
    setError(error.message);
  } finally {
    setLoading(false); //Asegurar que la carga termina siempre
  }
};
```

Ilustración 35. Código función fetchTitulo en DetallesTituloDiseño

Por otro lado, para la visualización de los centros asociados, se recorre el array correspondiente y se renderizan los nombres de los centros con la opción de eliminación. El botón de eliminación invoca la función *handleEliminarCentro* - véase *Ilustración 36. Código función handleEliminarCentro en DetallesTituloDiseno*-, que ejecuta una petición DELETE al servidor y, en caso de éxito, actualiza el estado local para reflejar los cambios en la interfaz. Por su parte, el formulario de incorporación de centros aprovecha el estado nuevoCentro y un menú select para añadir nuevos registros mediante la función *handleAgregarCentro* - véase *Ilustración 37.Código función handleAgregarCentro en DetallesTituloDiseno* -.

```
//Funcion para eliminar un centro asociado a un titulo
const handleEliminarCentro = async (codigoCentro) => {
  const confirmacion = window.confirm("¿Estás seguro de que quieres eliminar este centro?");
  if (!confirmacion) return;

  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/centros/${codigoCentro}`, {
      method: "DELETE",
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || "Error al eliminar el centro");
    }

    // Eliminar el centro de la lista en el frontend
    setCentrosAsociados((prev) => prev.filter((c) => c.Codigo_Centro !== codigoCentro));

    alert("Centro eliminado correctamente");
  } catch (error) {
    console.error("Error al eliminar el centro:", error);
    alert(error.message);
  }
};
```

Ilustración 36. Código función handleEliminarCentro en DetallesTituloDiseno

```
//Función para añadir un nuevo centro al título
const handleAgregarCentro = async () => {
 if (!nuevoCentro) return;

 try {
   const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/centros`, {
     method: "POST",
     headers: { "Content-Type": "application/json" },
     body: JSON.stringify({ codigoCentro: nuevoCentro }),
   });

   if (!response.ok) {
     const errorData = await response.json();
     throw new Error(errorData.error || "Error al añadir el centro");
   }

   // Obtener el centro recién añadido
   const centroAgregado = centrosDisponibles.find((c) => c.Codigo_Centro === nuevoCentro);

   // Actualizar la lista de centros en el frontend
   setCentrosAsociados((prev) => [...prev, centroAgregado]);

   // Limpiar el formulario
   setNuevoCentro("");
 } catch (error) {
   console.error("Error al añadir el centro:", error);
   alert(error.message);
 }
```

Ilustración 37.Código función handleAgregarCentro en DetallesTituloDiseno

Finalmente, la tabla de historial se construye dinámicamente a partir de los datos obtenidos mediante la función fetchHistorial – véase *Ilustración 38. Código función fetchHistorial en DetallesTituloDiseno*-, representando cada entrada en una fila que detalla el estado, subestado (si procede) y la fecha de inicio de este.

```
//Función para obtener el historial del título
const fetchHistorial = async () => {
  try {
    const responseHistorial = await fetch(`http://localhost:3001/api/titulos/${codigo}/historial`);
    if (!responseHistorial.ok) throw new Error("Error al obtener el historial del título");

    const dataHistorial = await responseHistorial.json();
    console.log("Historial del título:", dataHistorial);
    setHistorial(dataHistorial);
  } catch (error) {
    console.error("Error al cargar el historial:", error);
  }
};
```

Ilustración 38. Código función fetchHistorial en DetallesTituloDiseno

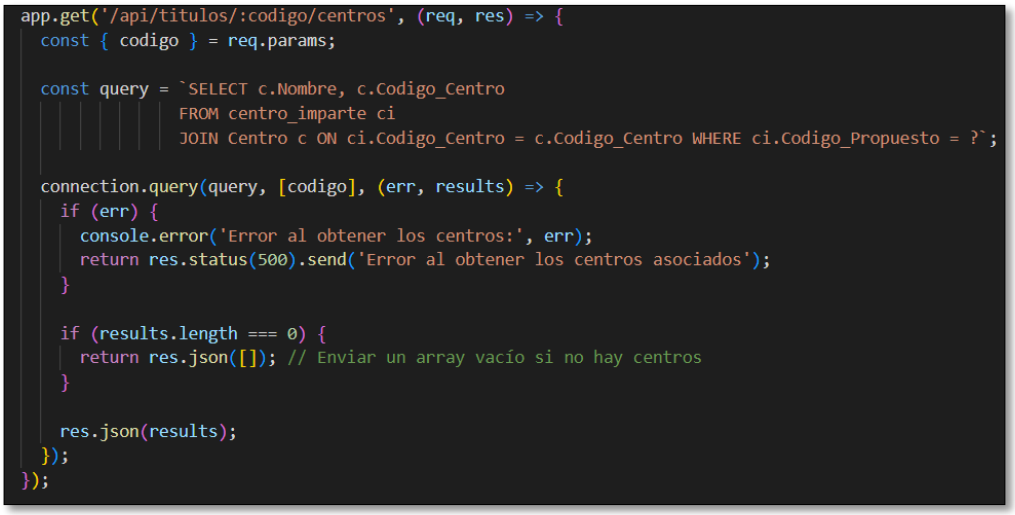
Para respaldar esta funcionalidad en el servidor, se han definido varios *endpoints* REST que proporcionan los datos requeridos:

- **GET /api/titulos/:código:** La información general del título se obtiene a través de una consulta SELECT que une la tabla principal de títulos con la de centros responsables, permitiendo así recuperar el nombre del centro además de los atributos propios del título.

```
app.get('/api/titulos/:codigo', (req, res) => {
  const { codigo } = req.params;

  connection.query(`SELECT t.*, c.Nombre AS Nombre_Centro
                    FROM titulo t
                    JOIN centro c ON t.Codigo_Centro_Responsable = c.Codigo_Centro WHERE Codigo_Propuesto = ?`, [codigo], (err, results) => {
    if (err) {
      console.error('Error al obtener los detalles del título:', err);
      return res.status(500).send('Error al obtener los detalles del título');
    }

    if (results.length === 0) {
      return res.status(404).send('Título no encontrado');
    }

    res.json(results[0]);
  });

});
```

Ilustración 39. Código ruta GET /api/titulos/:codigo

- **GET /api/titulos/:codigo/centros:** La lista de centros actualmente asociados a la impartición del título se gestiona a través de esta ruta que realiza una consulta con JOIN entre las tablas centro_imparte y centro obteniendo de este modo todos los centros asociados al título en cuestión.



```
app.get('/api/titulos/:codigo/centros', (req, res) => {
  const { codigo } = req.params;

  const query = `SELECT c.Nombre, c.Codigo_Centro
                 FROM centro_imparte ci
                 JOIN Centro c ON ci.Codigo_Centro = c.Codigo_Centro WHERE ci.Codigo_Propuesto = ?`;

  connection.query(query, [codigo], (err, results) => {
    if (err) {
      console.error('Error al obtener los centros:', err);
      return res.status(500).send('Error al obtener los centros asociados');
    }

    if (results.length === 0) {
      return res.json([]); // Enviar un array vacío si no hay centros
    }

    res.json(results);
  });
});
```

Ilustración 40. Código ruta GET /api/titulos/:codigo/centros

- **GET /api/centros:** La carga de todos los centros disponibles se efectúa mediante una consulta directa sobre la tabla Centro a través de esta ruta – véase *Ilustración 27. Código ruta /api/centros*-.

- **POST /api/titulos/:codigo/centros:** La incorporación de nuevos centros participantes se implementa mediante una operación POST, que primero verifica la no duplicidad de la relación antes de realizar una inserción en la tabla centro_imparte. Esta lógica garantiza la integridad referencial y evita asociaciones redundantes.

```
//Ruta para añadir un centro participante a un titulo
app.post('/api/titulos/:codigo/centros', (req, res) => {
  const { codigo } = req.params;
  const { codigoCentro } = req.body;

  if (!codigoCentro) {
    return res.status(400).json({ error: "Debe proporcionar un código de centro" });
  }

  console.log(`Añadiendo centro ${codigoCentro} al título ${codigo}`);

  // Verificar si el centro ya está asociado
  const checkQuery = `
    SELECT * FROM centro_imparte
    WHERE Codigo_Propuesto = ? AND Codigo_Centro = ?`;

  connection.query(checkQuery, [codigo, codigoCentro], (err, results) => {
    if (err) {
      console.error("Error al verificar centro:", err);
      return res.status(500).json({ error: "Error al verificar el centro" });
    }

    if (results.length > 0) {
      return res.status(400).json({ error: "Este centro ya está asociado al título" });
    }

    // Si no está asociado, insertarlo
    const insertQuery = `
      INSERT INTO centro_imparte (Codigo_Propuesto, Codigo_Centro) VALUES (?, ?)`;

    connection.query(insertQuery, [codigo, codigoCentro], (err) => {
      if (err) {
        console.error("Error al añadir centro:", err);
        return res.status(500).json({ error: "Error al añadir el centro" });
      }
      res.status(201).json({ message: "Centro añadido correctamente" });
```

Ilustración 41. Código ruta POST /api/titulos/:codigo/centros

- **DELETE /api/titulos/:codigo/centros/:codigoCentro:** De manera análoga, la eliminación de un centro participante se realiza con una consulta DELETE que elimina la entrada correspondiente de la tabla centro_imparte, siempre que la asociación exista.

```
//Ruta para eliminar un centro participante de un titulo
app.delete('/api/titulos/:codigo/centros/:codigoCentro', (req, res) => {
  const { codigo, codigoCentro } = req.params;

  console.log(`Intentando eliminar el centro ${codigoCentro} del título ${codigo}`);

  const query = `DELETE FROM centro_imparte WHERE Codigo_Propuesto = ? AND Codigo_Centro = ?`;

  connection.query(query, [codigo, codigoCentro], (err, results) => {
    if (err) {
      console.error("Error al eliminar el centro:", err);
      return res.status(500).json({ error: "Error al eliminar el centro" });
    }

    if (results.affectedRows === 0) {
      return res.status(404).json({ error: "El centro no está asociado a este título" });
    }

    res.json({ message: "Centro eliminado correctamente" });
  });
});
```

Ilustración 42. Código ruta DELETE /api/titulos/:codigo/centros/:codigoCentro

- **GET /api/titulos/:codigo/historial:** Finalmente, la obtención del historial de estados se gestiona mediante una consulta a la tabla HistorialProcesoTitulo, filtrada por el identificador del título. Esta información se envía como un array de objetos que posteriormente es interpretado y visualizado en el *frontend* en forma de tabla.

```
// Ruta para obtener el historial de un título específico
app.get('/api/titulos/:codigo/historial', (req, res) => {
  const { codigo } = req.params;

  const query = `
    SELECT * FROM HistorialProcesoTitulo WHERE Codigo_Propuesto = ?
  `;


  connection.query(query, [codigo], (err, results) => {
    if (err) {
      console.error('Error al obtener el historial del título:', err);
      return res.status(500).send('Error al obtener el historial del título');
    }


    res.json(results);
  });
});
```

Ilustración 43. Código ruta GET /api/titulos/:codigo/historial

4.2.3.3 Secciones de Aprobación (Estructura Común)

En el proceso de tramitación de un nuevo título universitario, diversas fases del procedimiento requieren una validación formal mediante la aportación de documentación específica. Para gestionar esta validación, se han implementado

secciones de aprobación estructuralmente homogéneas en la interfaz de usuario. Estas secciones permiten la recogida y posterior visualización de documentos clave, acompañados de su correspondiente fecha de aprobación.

Inicialmente, cada sección de aprobación se presenta al usuario como un formulario dinámico – véase *Ilustración 44. Interfaz formularios subida documentos de títulos en diseño*- . Este incluye un campo obligatorio para la introducción de la fecha de aprobación y uno o varios campos de carga de archivos, dependiendo de los documentos requeridos en dicha fase. Por ejemplo, en la etapa de "Aprobación Inicial por la Escuela", se solicitan tanto la propuesta de título como el acta correspondiente.

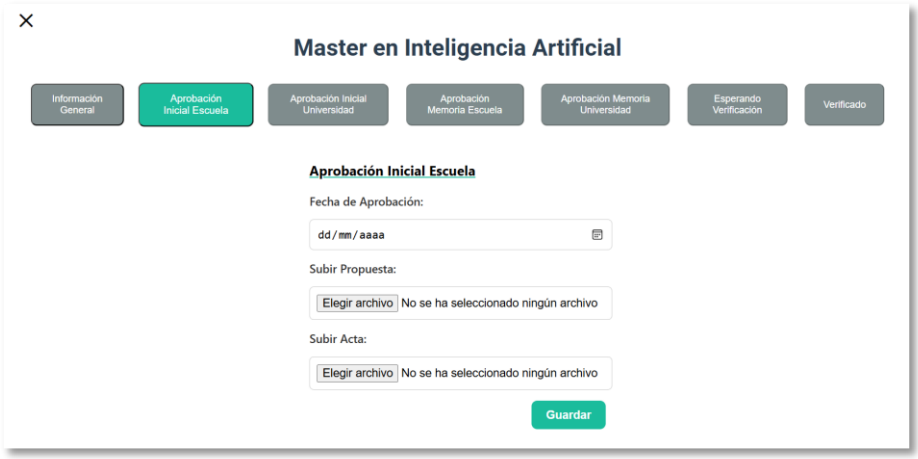


Ilustración 44. Interfaz formularios subida documentos de títulos en diseño

Una vez completado y enviado el formulario, el sistema reemplaza la vista editable por una representación estática – véase *Ilustración 45. Interfaz resultados formularios subida documentos de títulos en diseño* -. En esta, se muestra la fecha introducida junto con enlaces de descarga directa a los documentos almacenados en el servidor. Esta transición de formulario a vista estática no solo mejora la trazabilidad del proceso, sino que asegura la inmutabilidad de la documentación asociada a cada fase.

Ilustración 45. Interfaz resultados formularios subida documentos de títulos en diseño

Cada una de las etapas del proceso de aprobación de títulos presenta particularidades en cuanto a los documentos requeridos y a las validaciones necesarias en el *backend*. En la etapa de Aprobación Inicial Escuela, se exige la carga de la propuesta del título y el acta correspondiente, junto con la fecha de aprobación. Por su parte, la Aprobación Inicial Universidad requiere únicamente el acuerdo del Consejo de Gobierno. En la Aprobación de Memoria por parte de la Escuela, se solicitan tanto la memoria del título como el acta de aprobación de dicha memoria, mientras que la Aprobación de Memoria Universidad demanda exclusivamente el acuerdo del Consejo de Gobierno correspondiente.

Desde el punto de vista del cliente, cada una de las secciones de aprobación está compuesta por un componente *React* que sigue una estructura funcional común. Estas secciones presentan una lógica condicional que permite, en función de la existencia o no de documentos previamente cargados, alternar entre dos vistas: una de visualización documental y otra de carga de nueva documentación.

Para ello, se emplea la longitud del array documentos para determinar si ya se han registrado documentos anteriormente. Si el array contiene elementos, se presenta al usuario la fecha de aprobación (extraída de la propiedad Fecha_Subida del primer documento) y un listado con enlaces de descarga directa para cada uno de los documentos subidos.

En el caso contrario, es decir, cuando no existen documentos registrados para esta sección, se muestra un formulario de carga, compuesto por:

- Un campo de selección de fecha (type="date") para registrar la fecha oficial de aprobación.
- Campos de selección de archivos (type="file") para la subida de los documentos requeridos.
- Un botón de acción que activa el envío del formulario.

```
{/* Sección 2: Aprobación Inicial Escuela */}
{seccionActiva === 'aprobacionInicialEscuela' && (
  <div className="seccion">
    <h3>Aprobación Inicial Escuela</h3>
    <div className= "formulario-diseno">
    {documentos.length > 0 ? (
    <>
      {/* Mostrar la fecha de aprobación */}
      <p><span class="titulo">Fecha de Aprobación:</span> {new Date(documentos[0].Fecha_Subida).toLocaleDateString()}</p>
      {/*Mostrar los documentos con su botón de descarga */}
      {documentos.map((doc) => (
        <div key={doc.URL_Documento}>
          <p><span class="titulo">{doc.Nombre_Documento}: </span>
          <a href={`http://localhost:3001/api/documentos/${doc.URL_Documento}`} download>
            Descargar {doc.Nombre_Documento} 📄
          </a></p>
        </div>
      ))}
    </>
  ) : (
    <form onSubmit={handleSubirDocumento}>
      <label >Fecha de Aprobación:</label>
      <input type="date" value={fechaAprobacion} onChange={(e) => setFechaAprobacion(e.target.value)} required />

      <label>Subir Propuesta:</label>
      <input type="file" onChange={(e) => setPropuesta(e.target.files[0])} required />

      <label>Subir Acta:</label>
      <input type="file" onChange={(e) => setActa(e.target.files[0])} required />

      <button className="boton-guardar-cambio" type="submit">Guardar</button>
    </form>
  )}
  </div>
  </div>
)}
```

Ilustración 46. Código secciones aprobación en DetallesTituloDiseno

El envío de datos al servidor se gestiona mediante la función *handleSubirDocumento* –véase *Ilustración 47. Código función handleSubirDocumento en DetallesTituloDiseno*-, la cual previene el comportamiento por defecto del formulario y valida que todos los campos requeridos estén completos. A continuación, construye un objeto *FormData* con la fecha, el estado y los archivos adjuntos, y lo envía mediante una petición POST al *endpoint* del *backend* correspondiente al título seleccionado.

```
const handleSubirDocumento = async (e) => {
  e.preventDefault();
  console.log(" Enviando formulario de Aprobación Inicial Escuela...");

  if (!fechaAprobacion || !propuesta || !acta) {
    Create Jira Issue
    alert("Todos los campos son obligatorios.");
    return;
  }

  const formData = new FormData();
  formData.append("codigoEstado", "Aprobación Inicial Escuela");
  formData.append("fechaAprobacion", fechaAprobacion);
  formData.append("tipoAprobacion", "escuela");

  if (propuesta) {
    formData.append("propuesta", propuesta);
  }
  if (acta) {
    formData.append("acta", acta);
  }

  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/documentos`, {
      method: "POST",
      body: formData,
    });

    if (!response.ok) throw new Error("Error al subir los documentos");

    alert("Documentos subidos correctamente");
    setFechaAprobacion("");
    setPropuesta(null);
    setActa(null);

    fetchDocumentos();  //Actualizará la lista sin recargar la pagina
    fetchTitulo();
    fetchHistorial();
    verificarProgreso();
  } catch (error) {
    console.error("Error al subir los documentos:", error);
  }
};
```

Ilustración 47. Código función handleSubirDocumento en DetallesTituloDiseno

Por otra parte, la función *fetchDocumentos* – véase *Ilustración 48. Código función fetchDocumentos en DetallesTituloDiseno*- permite recuperar los documentos previamente cargados, filtrando únicamente aquellos correspondientes a esta sección específica. Para ello, se realiza una petición GET al servidor, y posteriormente se filtra el array recibido para incluir únicamente aquellos documentos cuyo nombre sea el indicado para cada una de las secciones, por ejemplo, en la sección “Aprobación Inicial Escuela” se filtrará por los nombres "Propuesta de Título" o "Acta".

```
const fetchDocumentos = async () => {
  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/documentos`);
    if (!response.ok) throw new Error("Error al obtener documentos");

    const data = await response.json();

    //Filtrar solo los documentos de Aprobación Inicial Escuela
    const documentosFiltrados = data.filter(doc =>
      doc.Nombre_Documento === "Propuesta de Título" || doc.Nombre_Documento === "Acta"
    );

    setDocumentos(documentosFiltrados);
  } catch (error) {
    console.error("Error al cargar documentos:", error);
  }
};
```

Ilustración 48. Código función fetchDocumentos en DetallesTituloDiseno

El servidor implementa una serie de rutas RESTful que permiten tanto la consulta como la carga de documentos, empleando para ello el framework Express.js junto con la librería *multer* para la gestión de archivos.

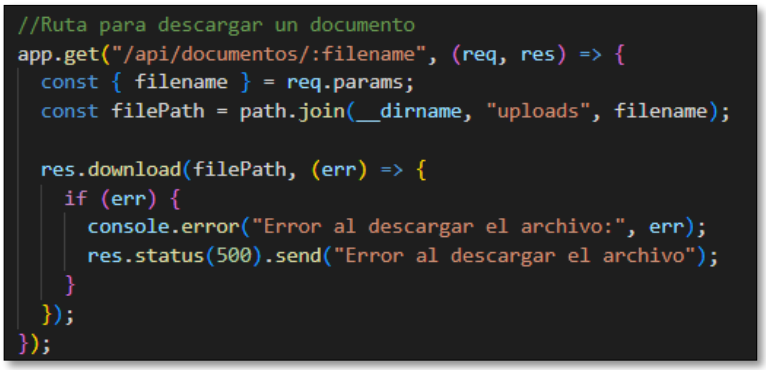
Las rutas más relevantes para la funcionalidad de las secciones de aprobación son las siguientes:

- **GET /api/titulos/:codigo/documentos:** Esta ruta permite obtener todos los documentos asociados a un título específico, identificados mediante el campo Codigo_Propuesto.

```
app.get("/api/titulos/:codigo/documentos", (req, res) => {
  const { codigo } = req.params;

  const query = `
    SELECT Nombre_Documento, URL_Documento, Fecha_Subida
    FROM documentaciontitulo
    WHERE Codigo_Propuesto = ?`;

  connection.query(query, [codigo], (err, results) => {
    if (err) {
      console.error("Error al obtener los documentos:", err);
      return res.status(500).json({ error: "Error al obtener los documentos" });
    }
    console.log("📂 Documentos obtenidos:", results);
    res.json(results);
  });
});
```

Ilustración 49. Código ruta GET/api/titulos/:codigo/documentos

- **GET /api/documentos/:filename:** Permite descargar un documento concreto, dado su nombre, accediendo al sistema de archivos del servidor (/uploads) y enviando el archivo como descarga directa mediante res.download().



```
//Ruta para descargar un documento
app.get("/api/documentos/:filename", (req, res) => {
  const { filename } = req.params;
  const filePath = path.join(__dirname, "uploads", filename);

  res.download(filePath, (err) => {
    if (err) {
      console.error("Error al descargar el archivo:", err);
      res.status(500).send("Error al descargar el archivo");
    }
  });
});
```

Ilustración 50. Código ruta GET /api/documentos/:filename

- **POST /api/titulos/:codigo/documentos:** Esta ruta maneja la subida de documentos, siendo utilizada por las distintas secciones de aprobación. El servidor recibe múltiples archivos, cada uno identificado por un nombre específico que representa distintos tipos de documentos (por ejemplo, "propuesta", "acta", "memoria_titulo", etc.). Junto con los archivos, también se procesan metadatos como la fecha de aprobación, el estado del proceso y el tipo de aprobación.

  A continuación, realiza validaciones específicas en función del tipo de aprobación recibido (tipoAprobacion), asegurando que se hayan enviado todos los documentos requeridos para cada fase del proceso (como "Aprobación Inicial Escuela" o "Verificado"). Una vez validados, los documentos son registrados en la base de datos, donde se almacena su nombre, ubicación y fecha de subida. Además, se actualiza el estado y subestado del título en la base de datos, así como el historial del proceso.

4.2.3.4 Sección: Esperando Verificación

En esta etapa del proceso, se habilita al usuario la posibilidad de solicitar la verificación oficial del título una vez completadas las fases previas de aprobación.

Al acceder a la sección "Esperando Verificación", se muestra un botón denominado "Solicitar Verificación" siempre que el título aún no haya sido enviado a verificación – véase *Ilustración 51. Interfaz Esperando Verificación (Botón)*-. Esta interfaz se adapta dinámicamente en función del estado del título: una vez que el usuario hace clic en dicho botón, este desaparece y en su lugar

se muestra la fecha exacta en la que se ha registrado la solicitud – véase *Ilustración 52. Interfaz Esperando Verificación (Fecha)*-.

Ilustración 51. Interfaz Esperando Verificación (Botón)

Ilustración 52. Interfaz Esperando Verificación (Fecha)

Desde el punto de vista del *frontend*, el componente React realiza varias operaciones clave. En primer lugar, mediante la función fetchEstadoVerificacion – véase *Ilustración 53. Código función fetchEstadoVerificación*-, se consulta la API para recuperar tanto el estado actual del título como su historial, con el fin de determinar si ya se ha registrado previamente una solicitud de verificación. Si no existe tal registro, se habilita el botón para iniciar el proceso.

```
const fetchEstadoVerificacion = async () => {
  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}`);
    if (!response.ok) throw new Error("Error al obtener el estado del título");

    const data = await response.json();
    setEstadoVerificacion(data.Estado);
    setSubestadoVerificacion(data.Subestado);

    //Buscar la fecha de solicitud en el historial
    const responseHistorial = await fetch(`http://localhost:3001/api/titulos/${codigo}/historial`);
    if (!responseHistorial.ok) throw new Error("Error al obtener el historial");

    const historial = await responseHistorial.json();
    const registroVerificacion = historial.find(entry => entry.Subestado === "En Verificación");

    if (registroVerificacion) {
      setFechaSolicitudVerificacion(new Date(registroVerificacion.Fecha_Inicio).toLocaleDateString("es-ES"));
    }
  } catch (error) {
    console.error("Error al cargar estado de verificación:", error);
  }
};
```

Ilustración 53. Código función fetchEstadoVerificación

Al pulsarlo, se ejecuta la función handleSolicitarVerificacion – véase *Ilustración 54. Código función handleSolicitarVerificación*-, que realiza una petición POST a la ruta /api/titulos/:codigo/solicitar-verificacion. Si la operación es exitosa, el *frontend* actualiza inmediatamente el estado de la interfaz, sustituyendo el botón por un mensaje que muestra la fecha en que se solicitó la verificación, obtenida directamente de la respuesta del *backend*.

```
const handleSolicitarVerificacion = async () => {
  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/solicitar-verificacion`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
    });

    if (!response.ok) throw new Error("Error al solicitar la verificación");

    const data = await response.json();
    alert("Verificación solicitada correctamente");

    setFechaSolicitudVerificacion(new Date(data.fechaSolicitud).toLocaleDateString("es-ES"));

    fetchEstadoVerificacion(); // ⚡ Actualizar la interfaz con los nuevos datos
    verificarProgreso();
  } catch (error) {
    console.error("Error al solicitar la verificación:", error);
  }
};
```

Ilustración 54. Código función handleSolicitarVerificación

En cuanto al *backend*, este responde a la solicitud de verificación mediante la ruta **POST /api/titulos/:codigo/solicitar-verificacion**. En esta operación, el sistema realiza tres acciones encadenadas:

1. Actualiza el estado y subestado del título a "En Verificación" en la base de datos.

```
app.post("/api/titulos/:codigo/solicitar-verificacion", (req, res) => {
  const { codigo } = req.params;
  const fechaSolicitud = new Date().toISOString().split("T")[0]; // Obtiene la fecha actual en formato YYYY-MM-DD

  console.log("Procesando solicitud de verificación para:", codigo);
  console.log("Fecha de solicitud:", fechaSolicitud);
  // Validación de datos
  if (!codigo) {
    return res.status(400).json({ error: "El código del título es obligatorio." });
  }

  let operations = [];
  //Actualizar el Estado y Subestado en la tabla `titulo`
  const updateTituloQuery = `UPDATE titulo SET Estado = 'En Verificación', Subestado = 'En Verificación' WHERE Codigo_Propuesto = ?`;
  operations.push(new Promise((resolve, reject) => {
    connection.query(updateTituloQuery, [codigo], (err) => {
      if (err) reject(err);
      else resolve();
    });
  }));
```

Ilustración 55.Código ruta POST /api/titulos/:codigo/solicitar-verificacion I

2. Cierra el subproceso anterior en la tabla historialprocesotitulo asignando la fecha actual como Fecha_Fin.

```
// Cerrar la fase anterior en `historialprocesotitulo`
const updateHistorialQuery = `
  UPDATE historialprocesotitulo
  SET Fecha_Fin = NOW()
  WHERE Codigo_Propuesto = ? AND Fecha_Fin IS NULL`;

operations.push(new Promise((resolve, reject) => {
  connection.query(updateHistorialQuery, [codigo], (err) => {
    if (err) reject(err);
    else resolve();
  });
}));
```

Ilustración 56. Código ruta POST /api/titulos/:codigo/solicitar-verificacion II

3. Inserta un nuevo registro en dicha tabla con el nuevo estado, subestado y la fecha de inicio del cambio.

```
//Insertar un nuevo registro en `historialprocesotitulo`
const insertHistorialQuery = `
  INSERT INTO historialprocesotitulo (Estado, Subestado, Fecha_Inicio, Codigo_Propuesto)
  VALUES ('En Verificación', 'En Verificación', ?, ?)`;
```

Ilustración 57. Código ruta POST /api/titulos/:codigo/solicitar-verificacion III

Todos estos cambios se ejecutan de manera atómica mediante promesas agrupadas con Promise.all, lo que garantiza la integridad y consistencia de los datos en la base de datos.

```
operations.push(new Promise((resolve, reject) => {
  connection.query(insertHistorialQuery, [fechaSolicitud, codigo], (err) => {
    if (err) reject(err);
    else resolve();
  });
}));
// Ejecutar todas las operaciones
Promise.all(operations)
  .then(() => res.status(201).json({ message: "Verificación solicitada correctamente", fechaSolicitud }))
  .catch((err) => {
    console.error(" Error al procesar la solicitud de verificación:", err);
    res.status(500).json({ error: "Error en el proceso" });
  });
});
```

Ilustración 58. Código ruta POST /api/titulos/:codigo/solicitar-verificacion IV

4.2.3.5 Sección: Verificado

La sección "Verificado" constituye la etapa final en el ciclo de verificación de un título universitario, una vez superadas las fases previas de diseño, aprobación interna, evaluación y aprobación externa. Desde el punto de vista del usuario, esta sección tiene como objetivo registrar los documentos oficiales emitidos por los órganos competentes, especialmente aquellos vinculados a los informes de evaluación y a la resolución del Consejo de Universidades, así como introducir el código RUCT que formaliza la inclusión del título en el registro oficial nacional.

Desde el *frontend*, la interfaz presenta al usuario un formulario estructurado -véase *Ilustración 59. Interfaz formulario sección "Verificado"* -que permite introducir tres fechas fundamentales: la del informe provisional, la del informe definitivo, y la de la resolución del Consejo de Universidades. A cada una de estas fechas se asocia la posibilidad de subir el correspondiente archivo. Asimismo, se habilita un campo específico para introducir el código RUCT, que constituye el identificador oficial del título en el sistema del Ministerio de Universidades.

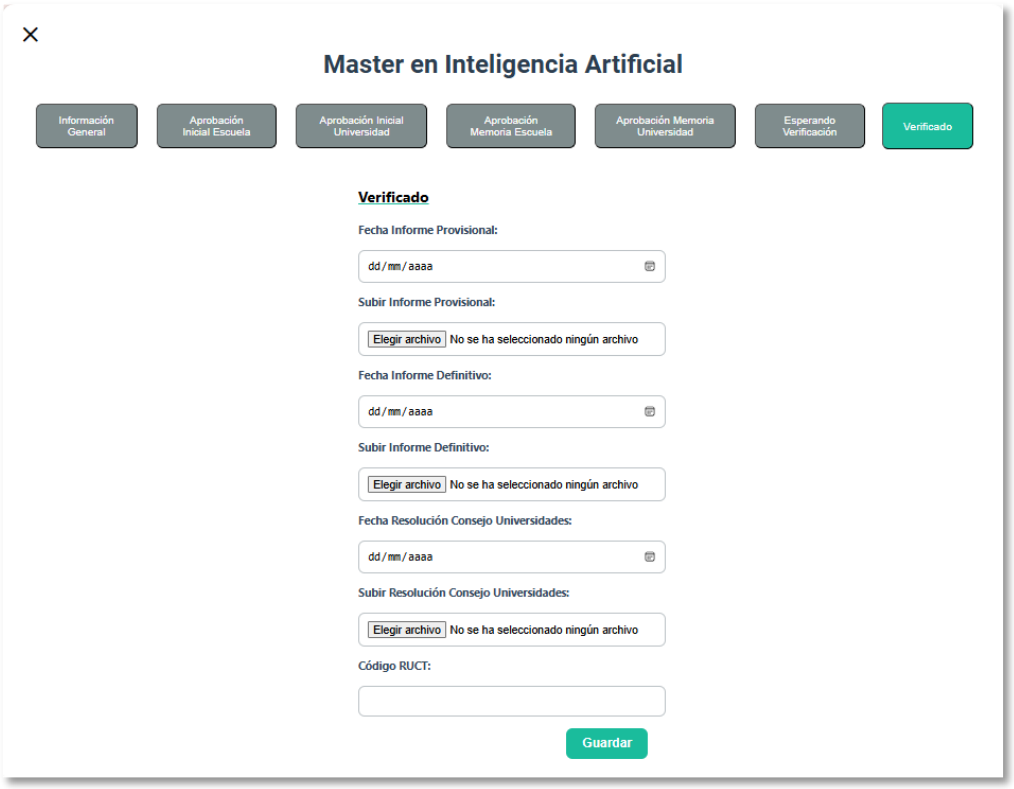


Ilustración 59. Interfaz formulario sección "Verificado"

El formulario está controlado por la función *handleSubirDocumentoVerificado* -véase *Ilustración 60. Código función handleSubirDocumentoVerificado en DetallesTituloDiseno*-, que se activa al pulsar el botón "Guardar". Esta función valida que todos los campos estén debidamente completados antes de proceder. En caso contrario, se interrumpe el envío y se notifica al usuario mediante una alerta.

```
const handleSubirDocumentoVerificado = async (e) => {
  e.preventDefault();
  if (
    !fechaInformeProvisional || !informeProvisional ||
    !fechaInformeDefinitivo || !informeDefinitivo ||
    !fechaResolucionCU || !resolucionCU || !codigoRuct
  ) {
    Create Jira Issue
    alert("Todos los campos son obligatorios.");
    return;
  }

  const formData = new FormData();
  formData.append("codigoEstado", "Verificado");
  formData.append("tipoAprobacion", "verificado");
  formData.append("fechaInformeProvisional", fechaInformeProvisional);
  formData.append("fechaInformeDefinitivo", fechaInformeDefinitivo);
  formData.append("fechaResolucionCU", fechaResolucionCU);
  formData.append("codigoRuct", codigoRuct);
  formData.append("informeProvisional", informeProvisional);
  formData.append("informeDefinitivo", informeDefinitivo);
  formData.append("resolucionCU", resolucionCU);

  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/documentos`, {
      method: "POST",
      body: formData,
    });

    if (!response.ok) throw new Error("Error al subir documentos");

    alert("Formulario Verificado completado correctamente");

    setFechaInformeProvisional(""); setInformeProvisional(null);
    setFechaInformeDefinitivo(""); setInformeDefinitivo(null);
    setFechaResolucionCU(""); setResolucionCU(null);
    setCodigoRuct("");

    fetchTitulo();
    fetchHistorial();
    verificarProgreso(); // si estás bloqueando las secciones
  } catch (error) {
```

Ilustración 60. Código función handleSubirDocumentoVerificado en DetallesTituloDiseno

Una vez validados los datos, se construye un objeto de tipo FormData, en el que se encapsulan las fechas, los documentos, el código RUCT, y los parámetros que indican el tipo de operación (en este caso "verificado") y el nuevo estado

deseado ("Verificado"). Esta información se envía mediante una solicitud HTTP POST a la ruta correspondiente del *backend*.

Tras recibir una respuesta exitosa por parte del servidor, se limpia el formulario y se actualiza el estado interno de la aplicación a través de funciones auxiliares (fetchTitulo – véase *Ilustración 35. Código función fetchTitulo en DetallesTituloDiseño*-, fetchHistorial – véase *Ilustración 38. Código función fetchHistorial en DetallesTituloDiseno*- y verificarProgreso – véase *Ilustración 61. Código función verificarProgreso en DetallesTituloDiseno*-). Esto asegura que tanto la información visualizada como las restricciones de acceso a otras secciones reflejen correctamente la nueva situación del título. De esta manera, el título desaparece de la tabla de "Títulos en Diseño" y pasa a formar parte de la tabla "Títulos Vigentes", lo que marca formalmente su reconocimiento definitivo.

```
const verificarProgreso = async () => {
  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/documentos`);
    if (!response.ok) throw new Error("Error al obtener documentos");

    const documentos = await response.json();

    // Verificar si existen documentos en cada sección
    setAprobacionEscuelaCompleta(documentos.some(doc => doc.Nombre_Documento === "Propuesta de Título" || doc.Nombre_Documento === "Acta"));
    setAprobacionUniversidadCompleta(documentos.some(doc => doc.Nombre_Documento === "Acuerdo Consejo de Gobierno"));
    setMemoriaEscuelaCompleta(documentos.some(doc => doc.Nombre_Documento === "Memoria del Título" || doc.Nombre_Documento === "Acta Memoria Escuela"));
    setMemoriaUniversidadCompleta(documentos.some(doc => doc.Nombre_Documento === "Acuerdo Consejo de Gobierno Memoria"));

    // Verificar si la solicitud de verificación ya ha sido realizada
    const responseHistorial = await fetch(`http://localhost:3001/api/titulos/${codigo}/historial`);
    if (!responseHistorial.ok) throw new Error("Error al obtener historial");

    const historial = await responseHistorial.json();
    setSolicitudVerificacionCompleta(historial.some(entry => entry.Subestado === "En Verificación"));

  } catch (error) {
    console.error("Error al verificar el progreso:", error);
  }
};
```

Ilustración 61. Código función verificarProgreso en DetallesTituloDiseno

En lo referente al *backend* el servidor expone un conjunto de rutas GET y POST que permiten gestionar tanto la visualización como la carga de la documentación requerida. Estas rutas son comunes y se reutilizan en las distintas secciones del sistema, como en las utilizadas previamente en las secciones de aprobación. Las rutas en cuestión son las siguientes:

- **GET /api/titulos/:código** -véase *Ilustración 39. Código ruta GET /api/titulos/:codigo*-

- **GET /api/titulos/:codigo/documentos** – véase *Ilustración 49. Código ruta GET/api/titulos/:codigo/documentos* –
- **GET /api/titulos/:codigo/historial** – véase *Ilustración 43. Código ruta GET /api/titulos/:codigo/historial* -
- **GET /api/documentos/:filename** – véase *Ilustración 89. Código ruta /api/download/:filename* -
- **POST /api/titulos/:codigo/documentos:** Esta ruta permite registrar nuevos documentos en el sistema, así como actualizar el estado y subestado del título según el tipo de aprobación correspondiente. El endpoint acepta múltiples archivos y campos del formulario, y su lógica de procesamiento se adapta automáticamente a la fase del proceso (escuela, universidad, memoria o verificado).

  Durante su ejecución, esta ruta:
  - Valida que se hayan proporcionado todos los documentos requeridos para el tipo de aprobación seleccionado.
  - Inserta los documentos en la tabla DocumentacionTitulo.
  - Actualiza el subestado (y en su caso también el estado general) del título en la tabla Titulo.
  - Registra el cambio en el historial de estados (HistorialProcesoTitulo), cerrando el subestado anterior y abriendo uno nuevo.

4.3 Diseño e implementación del subsistema de gestión de títulos vigentes

El subsistema de gestión de títulos vigentes tiene como objetivo principal ofrecer a los usuarios institucionales una visión estructurada y actualizada del estado de los títulos oficiales gestionados por el centro. Esta funcionalidad es esencial para el seguimiento y control de la oferta académica en todas sus fases: desde la verificación inicial hasta su posible extinción.

4.3.1 Visualización de títulos vigentes y extinguidos

Desde la pantalla de inicio del rol correspondiente (jefe de estudios, subdirector de calidad, unidad técnica de calidad o responsable de título), al pulsar el botón

"Gestionar Títulos Oficiales", se abre una interfaz que permite consultar de forma diferenciada dos categorías principales de títulos: vigentes y extinguidos – véase *Ilustración 62. Interfaz vista "Títulos Oficiales"*-. Esta distinción resulta clave para proporcionar una visión clara del estado actual del catálogo de titulaciones.

- En la primera tabla se presentan los **títulos vigentes**, es decir, aquellos cuyo estado corresponde a *Verificado*, *En Implantación* o *Implantado*. De cada título se muestra información relevante como: nombre, código RUCT, coordinador asignado, tipo de titulación (Grado, Máster o Doctorado), centro responsable y el estado actual.
- A continuación, se incluye una segunda tabla denominada "**Títulos Extinguidos**", donde se recogen aquellos títulos que han sido formalmente extinguidos o se encuentran en proceso de extinción. En este caso, la información mostrada incluye: nombre del título, tipo, centro responsable, fecha de extinción y estado.

Dado que estas tablas pueden albergar un volumen considerable de registros, se incorpora un campo de búsqueda por nombre de título para facilitar la localización rápida de un programa concreto. Asimismo, existen filtros desplegables para acotar la visualización según tipo de titulación, centro responsable o estado.

Ilustración 62. Interfaz vista "Títulos Oficiales"

4.3.1.1 Lógica del cliente (frontend)

El componente ConsultarTitulos constituye una interfaz desarrollada con React que permite visualizar y consultar la información relativa a los títulos oficiales ofertados por la universidad.

Al inicializarse el componente, se ejecuta un efecto mediante el hook useEffect – véase *Ilustración 63. Código useEffect en ConsultarTitulos*-, el cual realiza varias peticiones asíncronas al servidor. Estas peticiones recuperan los datos esenciales: la lista de títulos, los centros responsables, las asignaciones de coordinadores y la información personal de los coordinadores. Una vez recibidos y validados, los datos se almacenan en distintos estados del componente utilizando el *hook useState*, permitiendo su uso posterior en el renderizado y la aplicación de filtros – véase *Ilustración 64. Código estados en ConsultarTitulo*-.

```
useEffect(() => {
    const fetchData = async () => {
      try {
        // Hacer ambas peticiones en paralelo
        const [responseTitulos, responseCentros, resCoord, resPersonal] = await Promise.all([
          fetch('http://localhost:3001/api/titulos'),  // Obtener títulos
          fetch('http://localhost:3001/api/centros'),   // Obtener centros
          fetch('http://localhost:3001/api/coordinacion_titulos'),
          fetch('http://localhost:3001/api/personal')

        ]);

        // Verificar si ambas respuestas son correctas
        if (!responseTitulos.ok) throw new Error('Error al obtener títulos');
        if (!responseCentros.ok) throw new Error('Error al obtener centros');

        // Convertir a JSON
        const titulosData = await responseTitulos.json();
        const centrosData = await responseCentros.json();
        const coordData = await resCoord.json();
        const personalData = await resPersonal.json();

        // Guardar en el estado
        setTitulos(titulosData);
        setCentros(centrosData);
        setCoordinaciones(coordData);
        setPersonal(personalData);
      } catch (error) {
        console.error('Error al obtener los datos:', error);
      }
    };

    fetchData();
  }, []); // Se ejecuta solo al montar el componente
```

Ilustración 63. Código useEffect en ConsultarTitulos

```
const ConsultarTitulos = () => {
  // Estados para gestionar datos y filtros del componente.
  const [titulos, setTitulos] = useState([]); // Lista de títulos obtenida del servidor.
  Create Jira Issue
  const [filtroEstado, setFiltroEstado] = useState('Todos'); // Filtro para el estado del título.
  Create Jira Issue
  const [filtroTipo, setFiltroTipo] = useState('Todos'); // Filtro para el tipo de título.
  Create Jira Issue
  const [filtroCentro, setFiltroCentro] = useState('Todos'); // Filtro para el centro responsable.
  const [centros, setCentros] = useState([]);
  const [busqueda, setBusqueda] = useState(''); // Texto ingresado en el campo de búsqueda.
  const navigate = useNavigate(); // Hook para redirigir entre rutas.

  const [coordinaciones, setCoordinaciones] = useState([]);
  const [personal, setPersonal] = useState([]);
```

Ilustración 64. Código estados en ConsultarTitulo

La funcionalidad central del componente gira en torno a un sistema de filtrado – véase *Ilustración 65. Código filtrado en ConsultarTitulos*- compuesto por varios criterios: tipo de título (Grado, Máster o Doctorado), estado (Implantado, Verificado, En Extinción, etc.) y centro responsable. Además, se incluye un campo de búsqueda textual que permite localizar títulos por nombre. A través de estas herramientas, el usuario puede acotar la lista de títulos vigentes según sus necesidades.

```
// Filtra los títulos vigentes según los filtros seleccionados y la búsqueda.
const titulosVigentes = titulos.filter((titulo) => {
  if (titulo.Estado === 'Extinto' || titulo.Estado === 'En Diseño' || titulo.Estado === 'En Verificación' ) return false;
  if (busqueda && !titulo.Nombre.toLowerCase().includes(busqueda.toLowerCase())) {
    return false; // Filtrar por texto de búsqueda.
  }
  Create Jira Issue
  if (filtroEstado !== 'Todos' && titulo.Estado !== filtroEstado) {
    return false; // Filtrar por estado.
  }
  Create Jira Issue
  if (filtroTipo !== 'Todos' && titulo.Tipo !== filtroTipo) {
    return false; // Filtrar por tipo.
  }
  Create Jira Issue
  if (filtroCentro !== 'Todos' && titulo.Codigo_Centro_Responsable !== filtroCentro) {
    return false; // Filtrar por centro responsable.
  }
  Create Jira Issue
  return true; // Incluir el título si pasa todos los filtros.
});

// Filtra los títulos extintos según la búsqueda.
const titulosExtintos = titulos.filter((titulo) => {
  return (
    titulo.Estado === 'Extinto' &&
    (!busqueda || titulo.Nombre.toLowerCase().includes(busqueda.toLowerCase()))
  );
});
```

Ilustración 65. Código filtrado en ConsultarTitulos

Una característica adicional relevante es la visualización del nombre del coordinador activo de cada título – véase *Ilustración 66. Código función obtenerNombreCoordinador en ConsultarTitulos-*. Para ello, se consulta la tabla de coordinaciones, identificando la entrada correspondiente al título cuya fecha de finalización sea nula, lo que indica una asignación vigente. A partir del correo electrónico asociado, se localiza la entrada correspondiente en el conjunto de datos del personal y se muestra su nombre completo en la tabla de resultados.

```
//Funcion para obtener nombre del coordinador
const obtenerNombreCoordinador = (codigoTitulo) => {
const coordinacion = coordinaciones.find(
  (c) => c.Codigo_Titulo === codigoTitulo && c.Fecha_Fin === null
);
if (!coordinacion) return '';

const persona = personal.find((p) => p.Email === coordinacion.Email);
return persona ? `${persona.Nombre} ${persona.Apellidos}` : '';
};
```

Ilustración 66. Código función obtenerNombreCoordinador en ConsultarTitulos

El componente ofrece también interactividad a través del *hook useNavigate*, permitiendo redirigir al usuario a la vista detallada de un título concreto al hacer clic sobre la fila correspondiente en la tabla. Este comportamiento facilita la exploración en profundidad de la información asociada a cada titulación.

4.3.1.2 Lógica del servidor (backend)

El *backend*, desarrollado con Node.js y Express, expone un conjunto de rutas GET que proporcionan todos los datos necesarios para alimentar la interfaz de usuario:

- **GET /api/títulos:** Devuelve todos los títulos registrados en el sistema, incluyendo una unión con la tabla Centro para obtener el nombre del centro responsable de cada uno – véase *Ilustración 26. Código ruta /api/titulos-*.
- **GET /api/centros:** Proporciona el listado de todos los centros académicos registrados -véase *Ilustración 27. Código ruta /api/centros-*.
- **GET /api/coordinacion_titulos**: Recupera las coordinaciones activas, es decir, aquellas cuya fecha de fin es nula. Esta información se utiliza

para determinar qué persona coordina actualmente cada título – véase *Ilustración 67. Código ruta GET /api/coordinacion_titulos*-.

```
// Ruta para obtener todas las coordinaciones activas (Fecha_Fin IS NULL)
app.get('/api/coordinacion_titulos', (req, res) => {
  const query = `
    SELECT *
    FROM coordinacion_titulos
    WHERE Fecha_Fin IS NULL;
  `;

  connection.query(query, (err, results) => {
    if (err) {
      console.error('Error al obtener las coordinaciones:', err);
      return res.status(500).send('Error al obtener las coordinaciones');
    }
    res.json(results);
  });
});
```

Ilustración 67. Código ruta GET /api/coordinacion_titulos

- **GET /api/personal:** Retorna los datos básicos (nombre, apellidos y correo electrónico) del personal académico, necesarios para asociar correctamente el nombre de cada coordinador con su título -véase *Ilustración 68. Código ruta GET /api/personal*-.

```
// Ruta para obtener toda la información del personal
app.get('/api/personal', (req, res) => {
  const query = `
    SELECT Email, Nombre, Apellidos
    FROM personal;
  `;

  connection.query(query, (err, results) => {
    if (err) {
      console.error('Error al obtener el personal:', err);
      return res.status(500).send('Error al obtener el personal');
    }
    res.json(results);
  });
});
```

Ilustración 68. Código ruta GET /api/personal

Las consultas SQL empleadas están optimizadas para proporcionar los datos necesarios de forma directa, minimizando la necesidad de procesamiento adicional en el *frontend*. Gracias a esta organización, se garantiza una respuesta eficiente incluso ante un elevado volumen de títulos registrados.

4.3.2 Visualización del detalle de Títulos Vigentes

La visualización del detalle de títulos vigentes mantiene una estructura similar a la utilizada para los títulos en diseño, aunque adaptada a las particularidades propias de esta fase. Esta sección se activa al seleccionar un título concreto desde la tabla de *Títulos Vigentes* – véase *Ilustración 62. Interfaz vista "Títulos Oficiales"*-, permitiendo al usuario acceder a una vista detallada del mismo. En esta interfaz, es posible consultar la información general del título, con la opción de descargar la documentación oficial asociada. Asimismo, se facilita la gestión de los antecedentes académicos y se ofrecen datos relevantes sobre su proceso de implantación y, en su caso, de extinción, proporcionando así una visión integral y estructurada de su trayectoria dentro del sistema universitario.

4.3.2.1 Sección: Información General

La sección Información General dentro del apartado de Títulos Vigentes proporciona una vista consolidada de los datos más relevantes e identificativos de un título ya en vigor en el sistema universitario. Esta información corresponde a la primera pestaña visible al acceder al detalle de un título específico y permite al usuario consultar su configuración académica de forma clara y ordenada —véase *Ilustración 69. Interfaz "Información General" de título vigente-*

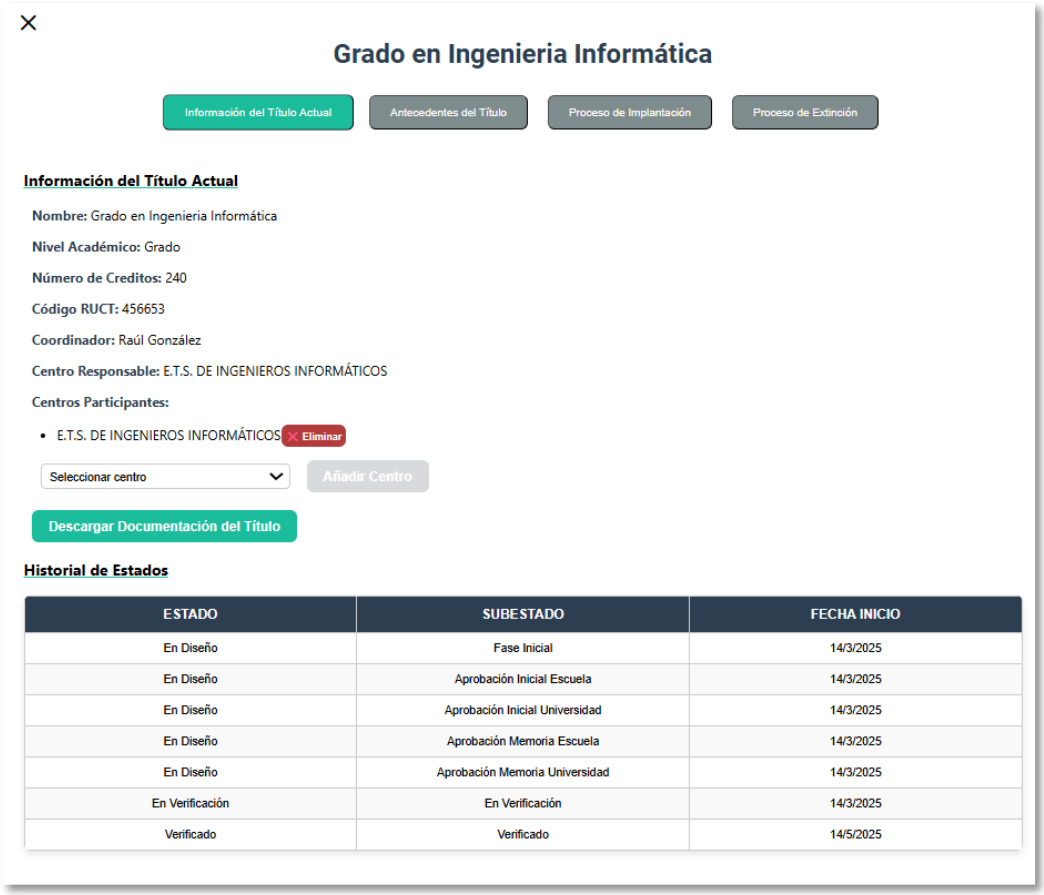


Ilustración 69. Interfaz "Información General" de título vigente

El acceso a esta sección se realiza al seleccionar un registro dentro de la tabla de "Títulos Vigentes" – véase *Ilustración 62. Interfaz vista "Títulos Oficiales"* -. Al hacerlo, se cargan y muestran los datos generales del título, entre los que destacan: su nombre oficial, nivel académico (grado, máster o doctorado), créditos ECTS y centro responsable. Además, se incluye información adicional, como el nombre del coordinador académico actual o el código RUCT, datos que no estaban disponibles en el proceso de diseño inicial del título. Esta información se presenta en formato estático, sin posibilidad de edición directa, para asegurar la integridad de los datos mostrados.

La gestión de la vista se realiza en un componente React principal que controla la visualización mediante un estado local *seccionActiva*, el cual se ajusta a *'informacion'* para mostrar esta pestaña al igual que ocurría en el caso de los títulos en diseño – véase *4.2.3.2* – .

De manera análoga a la sección de diseño, se integra un módulo interactivo para la gestión de los centros participantes asociados al título. Aquí, el usuario puede eliminar centros vinculados a través de botones específicos y añadir nuevos centros mediante un menú desplegable que lista todos aquellos aún no asociados.

También se mantiene de manera equivalente a la sección de diseño la tabla de historial de estados, que muestra la evolución del título desde su creación hasta su situación actual, incluyendo el estado, subestado y fecha de vigencia de cada entrada. Esta funcionalidad resulta fundamental para analizar la trazabilidad y evolución del expediente académico.

Como novedad respecto a la sección similar en “Títulos en Diseño”, la vista de Información General incorpora un botón titulado "Descargar Documentación del Título". Este botón abre una nueva página – véase *Ilustración 70. Interfaz de descarga de documentación asociada a un título* - dedicada a la consulta y descarga de todos los documentos generados o subidos durante el proceso de diseño e implantación del título, proporcionando transparencia documental y facilitando el acceso a la evidencia asociada al expediente.

Ilustración 70. Interfaz de descarga de documentación asociada a un título

Esta página de documentación, gestionada por el componente DocumentacionTitulo.js, se basa en la extracción y visualización dinámica de

los documentos vinculados al título cuyo código se obtiene a través del parámetro de URL (useParams). La obtención de documentos se realiza mediante una petición asincrónica a la API que devuelve un listado con los nombres y URLs de cada archivo – véase *Ilustración 71. Código obtención listado de documentos asociados a un título* -.

```
useEffect(() => {
  const fetchDocumentos = async () => {
    try {
      const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/documentos`);
      const data = await response.json();

      console.log("📄 Documentos recibidos:", data.map(doc => doc.Nombre_Documento));

      setDocumentos(data);
    } catch (error) {
      console.error('Error al obtener los documentos:', error);
    } finally {
      setCargando(false);
    }
  };

  fetchDocumentos();
}, [codigo]);
```

Ilustración 71. Código obtención listado de documentos asociados a un título

El componente reutiliza la lógica de estado y carga explicada previamente para los títulos (uso de useState y useEffect), pero se especializa en la presentación organizada de los documentos. En lugar de mostrarlos separados, integra todos los archivos en una única sección consolidada, dividida en categorías temáticas que agrupan los documentos según su tipo:

- **Propuesta de Título:** Incluye “Propuesta de Título”, “Acta” y “Acuerdo Consejo de Gobierno”.
- **Memoria Título Provisional:** Agrupa “Memoria del Título”, “Acta Memoria Escuela” y “Acuerdo Consejo de Gobierno Memoria”.
- **Otros documentos relevantes:** Presenta individualmente “Informe Provisional de Verificación”, “Informe Definitivo de Verificación” y “Resolución Consejo Universidades”.

Para cada documento disponible, el componente muestra un enlace directo para descargarlo, utilizando el atributo *download* y apuntando a la ruta: *http://localhost:3001/api/documentos/:URL_Documento*. Si el documento no

está disponible, se indica claramente con un texto que refleja su ausencia -véase *Ilustración 72. Código renderDocumento en DocumentacionTitulo* -.

```
const renderDocumento = (nombre) => {
  const doc = getDocByName(nombre);
  return (
    <li key={nombre}>
      {nombre}:{' '}
      {doc ? (
        <a href={`http://localhost:3001/api/documentos/${doc.URL_Documento}`} download>
          Descargar 📄
        </a>
      ) : (
        <span style={{ color: 'gray' }}>No disponible</span>
      )}
    </li>
  );
};
```

Ilustración 72. Código renderDocumento en DocumentacionTitulo

Adicionalmente, el componente incluye un botón (X) para cerrar la ventana y regresar a la vista previa del título, facilitando la navegación del usuario sin recargar la página o perder contexto.

Tanto el componente de información general como el módulo documental realizan diversas llamadas GET a la API para obtener la información necesaria:

- **GET /api/titulos/:código:** Carga los datos generales del titulo indicado en el parámetro "código" – véase *Ilustración 39. Código ruta GET /api/titulos/:codigo* -.
- **GET /api/titulos/:codigo/historial:** Obtiene el historial de estados por los que ha pasado el título – *véase Ilustración 43. Código ruta GET /api/titulos/:codigo/historial* -.
- **GET /api/titulos/:codigo/centros:** Muestra los centros asociados a un título – véase *Ilustración 40. Código ruta GET /api/titulos/:codigo/centros*.
- **GET /api/centros:** Obtiene todos los centros -véase *Ilustración 27. Código ruta /api/centros* -.
- **GET /api/titulos/:codigo/documentos:** Muestra un listado de todos los documentos vinculados a un título – véase Ilustración *49. Código ruta GET/api/titulos/:codigo/documentos* -.
- **GET /api/coordinacion_titulos:** Obtiene los datos de todos los coordinadores, se utiliza de manera cruzada para obtener nombre y

apellidos de la coordinación activa del título - véase *Ilustración 67. Código ruta GET /api/coordinacion_titulos* -.

- **GET /api/personal:** Obtiene la información de todo el personal, se utiliza de manera cruzada para obtener el nombre y apellidos de los coordinadores - véase *Ilustración 68. Código ruta GET /api/personal* -.

En cambio, la gestión de los centros (añadir y eliminar) se realiza con funciones que hacen peticiones POST y DELETE a las siguientes rutas, asegurando la no duplicidad y el correcto mantenimiento de las relaciones.

- **POST api/titulos/:codigo/centros:** Permite añadir un centro participante a un título en concreto – véase *Ilustración 41. Código ruta POST /api/titulos/:codigo/centros* -.
- **DELETE api/titulos/:codigo/centros/:codigoCentro:** Permite eliminar un centro participante de un título en concreto -véase *Ilustración 42. Código ruta DELETE /api/titulos/:codigo/centros/:codigoCentro*-.

Esta integración asegura una experiencia de usuario coherente y completa en la gestión, consulta y descarga de la documentación asociada a los títulos vigentes, aportando claridad, trazabilidad y acceso sencillo a la información académica y administrativa relevante.

4.3.2.2 Sección: Antecedentes del Título

La sección Antecedentes del Título permite al usuario consultar y gestionar los títulos que actúan como precedentes académicos de un título vigente concreto. Esta funcionalidad es especialmente relevante en el contexto de títulos que suponen una evolución, actualización o reemplazo de planes de estudios previos, y busca preservar la trazabilidad histórica de los programas ofrecidos por una universidad.

Desde el punto de vista del usuario, la interfaz ofrece una experiencia clara e intuitiva – véase *Ilustración 73. Interfaz sección "Antecedentes del Título"*-. En primer lugar, se presenta un desplegable que lista todos los títulos vigentes ofertados por el mismo centro, exceptuando el propio título actual y aquellos que ya se encuentran registrados como antecedentes. Este control permite seleccionar un título válido como posible antecedente. Una vez seleccionado, el

usuario puede asociarlo al título activo mediante el botón "Añadir Antecedente", lo que provoca su inclusión inmediata en la lista de "Títulos Antecedentes Asociados" mostrada en pantalla.

En esta lista se recogen todos los antecedentes previamente registrados, cada uno de ellos acompañado de un botón "Eliminar", que permite desvincular el título antecedente en caso necesario. Esta operación está protegida por un mensaje de confirmación que previene eliminaciones accidentales.

Ilustración 73. Interfaz sección "Antecedentes del Título"

La lógica de presentación y gestión de esta sección está implementada en React, utilizando el estado local para mantener sincronizada la información relevante. En concreto, se emplean las siguientes variables – véase *Ilustración 74. Código estados sección antecedentes*-:

- **titulosVigentesCentro:** almacena los títulos vigentes ofertados por el centro del usuario, obtenidos desde el almacenamiento local (localStorage) y consultados al iniciar la vista.
- **tituloSeleccionadoAntecedente:** recoge el valor del título seleccionado en el desplegable.
- **antecedentes:** contiene los antecedentes actualmente asociados al título activo.

```
//Estados seccion antecedentes
const [titulosVigentesCentro, setTitulosVigentesCentro] = useState([]);
const [tituloSeleccionadoAntecedente, setTituloSeleccionadoAntecedente] = useState("");
const [antecedentes, setAntecedentes] = useState([]);
const userCentro = localStorage.getItem('userCentro');
```

Ilustración 74. Código estados sección antecedentes

El procedimiento de adición de un nuevo antecedente se implementa en la función *handleAgregarAntecedente* – véase *Ilustración 75. Código función handleAgregarAntecedente en DetallesTitulo* -, que realiza una petición POST al *backend* y actualiza el estado local en caso de éxito. De igual modo, la función *handleEliminarAntecedente* – véase *Ilustración 76. Código función handleEliminarAntecedente en DetallesTitulo* - permite eliminar una asociación existente mediante una petición DELETE, actualizando el estado local para reflejar los cambios en la vista sin necesidad de recargar la página.

```
const handleAgregarAntecedente = async () => {
if (!tituloSeleccionadoAntecedente) return;

try {
  const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/antecedentes`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ codigoAntecedente: tituloSeleccionadoAntecedente }),
  });

  if (!response.ok) throw new Error("Error al añadir el antecedente");

  const antecedenteAgregado = titulosVigentesCentro.find(t => t.Codigo_Propuesto === tituloSeleccionadoAntecedente);
  setAntecedentes(prev => [...prev, antecedenteAgregado]);
  setTituloSeleccionadoAntecedente("");
} catch (error) {
  console.error("Error al añadir antecedente:", error);
  alert(error.message);
}
};
```

Ilustración 75. Código función handleAgregarAntecedente en DetallesTitulo

```
const handleEliminarAntecedente = async (codigoAntecedente) => {
  const confirmacion = window.confirm("¿Estás seguro de que quieres eliminar este antecedente?");
  if (!confirmacion) return;

  try {
    const response = await fetch(`http://localhost:3001/api/titulos/${codigo}/antecedentes/${codigoAntecedente}`, {
      method: "DELETE",
    });

    if (!response.ok) {
      throw new Error("Error al eliminar el antecedente");
    }

    setAntecedentes((prev) =>
      prev.filter((a) => a.Codigo_Propuesto !== codigoAntecedente)
    );

    alert("Antecedente eliminado correctamente");
  } catch (error) {
    console.error("Error al eliminar antecedente:", error);
    alert(error.message);
  }
};
```

Ilustración 76. Código función handleEliminarAntecedente en DetallesTitulo

En la vista, el desplegable aplica filtros para excluir tanto el propio título como aquellos que ya figuran como antecedentes, garantizando así la integridad semántica de los datos.

Por otro lado, el servidor expone las rutas necesarias para soportar las operaciones de consulta, inserción y eliminación de antecedentes. Todas ellas se implementan utilizando sentencias SQL directas sobre la base de datos:

- **GET /api/titulos/por-centro:** Esta ruta permite obtener todos los títulos vigentes impartidos por un centro específico, a partir de su nombre. La consulta une las tablas titulo, centro_imparte y centro para devolver los resultados -véase *Ilustración 18. Código ruta GET /api/titulos/por-centro* -.
- **GET /api/titulos/:codigo/antecedentes:** Realiza una consulta sobre la tabla antecedentes, unida con la tabla titulo, para recuperar los nombres y códigos de los antecedentes vinculados al título especificado -vease *Ilustración 77. Código ruta GET /api/titulos/:codigo/antecedentes* -.

```
//Ruta para obtener los antecedentes
app.get('/api/titulos/:codigo/antecedentes', (req, res) => {
  const codigo = req.params.codigo;

  const query = `
    SELECT t.Codigo_Propuesto, t.Nombre
    FROM antecedentes a
    JOIN titulo t ON a.Codigo_Titulo_Antecedente = t.Codigo_Propuesto
    WHERE a.Codigo_Titulo = ?
  `;

  connection.query(query, [codigo], (err, results) => {
    if (err) {
      console.error('Error al obtener los antecedentes:', err);
      return res.status(500).send('Error al obtener los antecedentes');
    }
    res.json(results);
  });
});
```

Ilustración 77. Código ruta GET /api/titulos/:codigo/antecedentes

- **POST /api/titulos/:codigo/antecedentes:** Inserta una nueva entrada en la tabla antecedentes, relacionando el código del título activo con el del título antecedente proporcionado en el cuerpo de la petición – véase *Ilustración 78. Código ruta POST /api/titulos/:codigo/antecedentes* -.

```
//Ruta para añadir un nuevo antecedente
app.post('/api/titulos/:codigo/antecedentes', (req, res) => {
  const codigo = req.params.codigo;
  const { codigoAntecedente } = req.body;

  const query = `
    INSERT INTO antecedentes (Codigo_Titulo, Codigo_Titulo_Antecedente)
    VALUES (?, ?)
  `;

  connection.query(query, [codigo, codigoAntecedente], (err) => {
    if (err) {
      console.error('Error al insertar antecedente:', err);
      return res.status(500).send('Error al insertar antecedente');
    }
    res.status(200).send('Antecedente añadido correctamente');
  });
});
```

Ilustración 78. Código ruta POST /api/titulos/:codigo/antecedentes

- **DELETE /api/titulos/:codigo/antecedentes/:codigoAntecedente:** Elimina la relación existente entre un título y su antecedente, previa comprobación de que dicha relación está registrada. En caso de error o inexistencia de la asociación, se devuelve el correspondiente mensaje de error – véase *Ilustración 79. Código ruta DELETE /api/titulos/:codigo/antecedentes/:codigoAntecedente*-.

```
// Ruta para eliminar un antecedente de un título
app.delete('/api/titulos/:codigo/antecedentes/:codigoAntecedente', (req, res) => {
  const { codigo, codigoAntecedente } = req.params;

  console.log(`Intentando eliminar el antecedente ${codigoAntecedente} del título ${codigo}`);

  const query = `DELETE FROM antecedentes WHERE Codigo_Titulo = ? AND Codigo_Titulo_Antecedente = ?`;

  connection.query(query, [codigo, codigoAntecedente], (err, results) => {
    if (err) {
      console.error("Error al eliminar el antecedente:", err);
      return res.status(500).json({ error: "Error al eliminar el antecedente" });
    }

    if (results.affectedRows === 0) {
      return res.status(404).json({ error: "El antecedente no está asociado a este título" });
    }

    res.json({ message: "Antecedente eliminado correctamente" });
  });
});
```

Ilustración 79. Código ruta DELETE /api/titulos/:codigo/antecedentes/:codigoAntecedente

Cada una de estas operaciones se encuentra protegida mediante manejo de errores, con el objetivo de ofrecer respuestas adecuadas tanto en caso de éxito como de fallo.

4.3.2.3 Sección: Proceso de implantación

Debido a las limitaciones temporales asociadas al desarrollo del proyecto, no ha sido posible implementar completamente la funcionalidad correspondiente a la sección del *Proceso de implantación*. No obstante, se ha diseñado un prototipo de baja fidelidad – véase *Ilustración 80. Prototipo baja fidelidad "Proceso de implantación"* -que refleja la lógica y el comportamiento esperado de esta sección en su futura implementación.

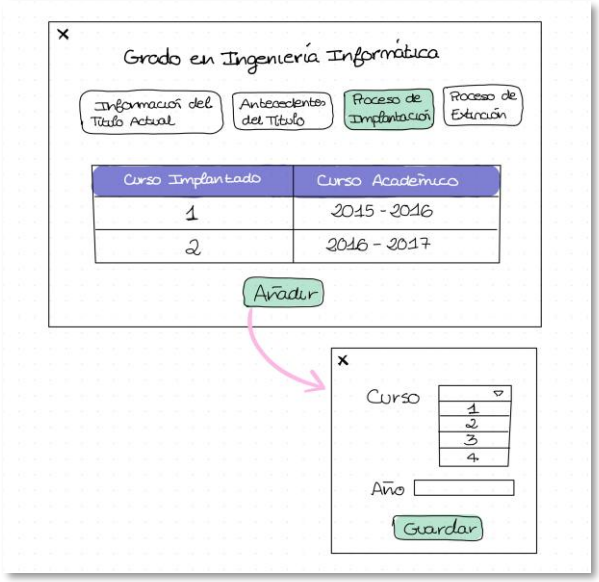


Ilustración 80. Prototipo baja fidelidad "Proceso de implantación"

El acceso a esta sección estará restringido a títulos cuyo estado sea "**Verificado**" o "**En implantación**", de modo que represente con fidelidad el flujo real del proceso de introducción progresiva del título en el sistema universitario.

Al acceder a la sección, el usuario visualizará inicialmente una tabla que recogerá la relación entre los cursos del plan de estudios y los cursos académicos en los que estos han sido implantados. Esta información se extraerá de la tabla *implantacion_titulo* de la base de datos, y estará vacía en caso de que el título aún no haya comenzado su proceso de implantación.

Justo debajo de dicha tabla se dispondrá un botón identificado como "Añadir", cuya funcionalidad será permitir el registro de nuevas implantaciones a medida que el título avanza en su despliegue académico. Al pulsar este botón, se abrirá una ventana emergente con un formulario para la inserción de nuevos registros.

Este formulario estará compuesto por los siguientes campos:

- **Curso a implantar**: será un campo tipo *select*, que mostrará únicamente aquellos cursos del título que aún no han sido implantados, con el fin de evitar redundancias. Esta selección dinámica se basará en una función que calcula el número total de cursos del título. Para ello, se dividirá el número de créditos totales del título (almacenado en la base de datos) entre 60, resultando en un valor entero que representa el número de cursos (asumiendo que cada curso comprende 60 ECTS).
- **Curso académico**: un campo de entrada tipo texto, en el que el usuario deberá introducir el curso académico en el que se implantará el curso seleccionado, siguiendo el formato 20XX-20XX.

Una vez completados ambos campos, el usuario podrá hacer clic en el botón "Guardar", lo que provocará la inserción de un nuevo registro en la tabla *implantacion_titulo* de la base de datos. Este registro representará una nueva etapa en la implantación del título.

Además, el sistema gestionará automáticamente la transición del estado del título en función del grado de implantación:

- Si se trata del **primer curso** implantado, el estado del título pasará de "**Verificado**" a "**En implantación**".
- Si se implanta el **último curso** pendiente, el estado cambiará de "**En implantación**" a "**Implantado**", indicando que el título se encuentra ya completamente operativo.

Este diseño anticipa un funcionamiento progresivo y controlado del proceso de implantación, garantizando la coherencia de los datos, la trazabilidad de las acciones y la evolución lógica del estado del título dentro del sistema.

4.3.2.4 Sección: Proceso de extinción

Al igual que en la sección anterior – véase *4.3.2.3* - debido a las restricciones temporales, no ha sido posible implementar de forma completa la funcionalidad correspondiente a la sección de Proceso de extinción. No obstante, se ha diseñado un prototipo de baja fidelidad – véase *Ilustración 81. Prototipo baja fidelidad "Proceso de extinción"*-que recoge la lógica de funcionamiento y las

transiciones esperadas dentro del ciclo de vida de un título en su fase de extinción.

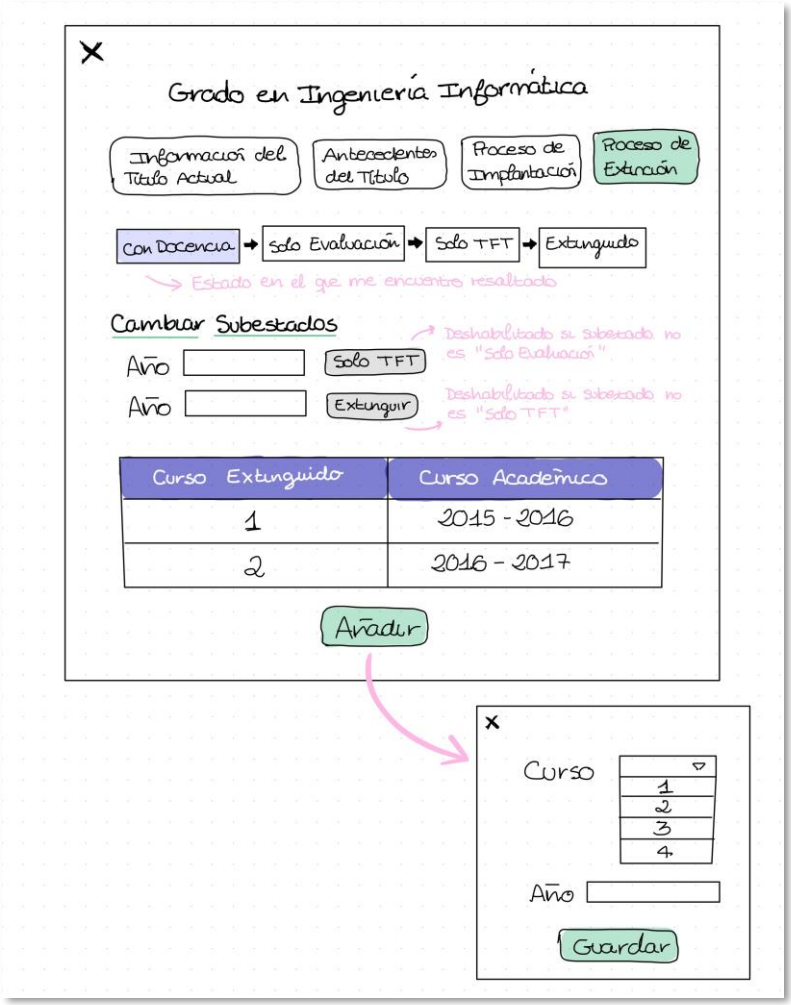


Ilustración 81. Prototipo baja fidelidad "Proceso de extinción"

Esta sección será accesible únicamente para aquellos títulos cuyo estado sea "Implantado" o "En extinción", reflejando así una coherencia con el flujo natural de los procesos académicos.

La vista principal de la sección mostrará, en primer lugar, una tabla en la que se relacionan los cursos del plan de estudios con los cursos académicos en los que se ha producido su extinción. Estos datos se obtendrán dinámicamente de la tabla *extincion_titulo* de la base de datos, y se irán completando conforme avanza el proceso de extinción del título.

Una característica diferencial de esta sección respecto a la de implantación es la incorporación de subestados asociados al estado “En extinción”, los cuales permiten reflejar con mayor precisión las fases progresivas de retirada de un título. Los subestados definidos son:

- **Con docencia**: Se ha iniciado la extinción del título, pero se siguen impartiendo clases.
- **Solo evaluación**: Ya no se imparte docencia, pero se siguen realizando exámenes o actividades de evaluación.
- **Solo Trabajo de Fin de Título (TFT)**: Ya no existe docencia ni evaluación, pero se permite la defensa del trabajo de fin de título.

Para facilitar la identificación del estado actual del proceso, se mostrarán todos los subestados posibles en la parte superior de la tabla, resaltando el subestado en el que se encuentra el título en ese momento. Esta visualización permitirá al usuario conocer de un vistazo el estado actual del proceso de extinción.

En la parte inferior de la tabla, se incluirá un botón "Añadir", que permitirá registrar nuevas extinciones de cursos. Al pulsar dicho botón se abrirá una ventana emergente que contiene un formulario, el cual se compondrá de los siguientes campos:

- **Curso a extinguir**: un campo tipo *select* que mostrará dinámicamente aquellos cursos del plan de estudios que aún no han sido extinguidos, para evitar redundancias y garantizar la consistencia de los datos.
- **Curso académico**: un campo de entrada de texto en el que se introducirá el curso académico en formato 20XX-20XX.

Una vez cumplimentado el formulario, al hacer clic en el botón "Guardar", se insertará un nuevo registro en la tabla *extincion_titulo* de la base de datos.

El sistema también gestionará de forma automática las transiciones de estado y subestado, en función del avance en el proceso:

- Al registrar el primer curso extinguido, el estado del título cambiará de “**Implantado**” a “**En extinción**” y el subestado se establecerá automáticamente en “**Con docencia**”.

- Una vez extinguido el último curso, el subestado pasará a "**Solo evaluación**".

Además, junto a la tabla del proceso de extinción se habilitará un apartado denominado **"Cambiar subestado"**, desde el cual será posible gestionar las siguientes transiciones:

- **A "Solo TFT"**: disponible únicamente cuando el subestado actual sea "Solo evaluación". Requiere indicar el curso académico en el que se produce la transición. Al pulsar el botón correspondiente, el subestado cambiará a "Solo TFT".
- **A "Extinguido"**: disponible únicamente cuando el subestado sea "Solo TFT". También se deberá especificar el curso académico. Al hacer clic en el botón, el estado del título cambiará a "Extinguido", marcando el fin oficial del ciclo de vida del título.

Cada cambio de estado o subestado quedará registrado automáticamente en la tabla *historialprocesotitulo*, almacenando el nuevo estado, la fecha de la transición y, cuando corresponda, el subestado asociado.

4.4 Integración del subsistema de gestión de la documentación asociada al título

En el marco del sistema de gestión del ciclo de vida de títulos académicos, se ha considerado esencial la incorporación de un subsistema específico para la gestión de la documentación asociada. Esta necesidad surge funcionalidad responde a la necesidad de seguir la trazabilidad y conservación de documentos justificativos en cada etapa del ciclo de vida de un título.

Este subsistema permite a los usuarios autorizados adjuntar, consultar y descargar archivos relevantes tales como memorias, informes de evaluación, certificados de acreditación u otros documentos de apoyo.

La implementación técnica del subsistema se ha abordado mediante una combinación de herramientas del entorno *Node.js* para el *backend*, en particular el middleware *Multer,* y tecnologías estándar del *frontend* para la gestión de

formularios y visualización. Todo ello integrado con el modelo relacional de datos para asegurar la persistencia y coherencia de la información.

4.4.1 Lógica del cliente (frontend)

Desde el punto de vista del cliente, la funcionalidad de gestión documental ha sido implementada mediante formularios dinámicos que permiten al usuario subir archivos desde su equipo local. Cada formulario está vinculado a un título específico, lo que garantiza que los documentos queden correctamente asociados en el sistema.

La interacción se produce mediante formularios HTML – véase *Ilustración 82. Formulario subida de archivos*- que incorporan un campo de tipo file para la selección del archivo. Una vez cumplimentado, el formulario realiza una petición POST al servidor utilizando el objeto *FormData* de *JavaScript*, lo cual permite transmitir el archivo de manera eficiente.

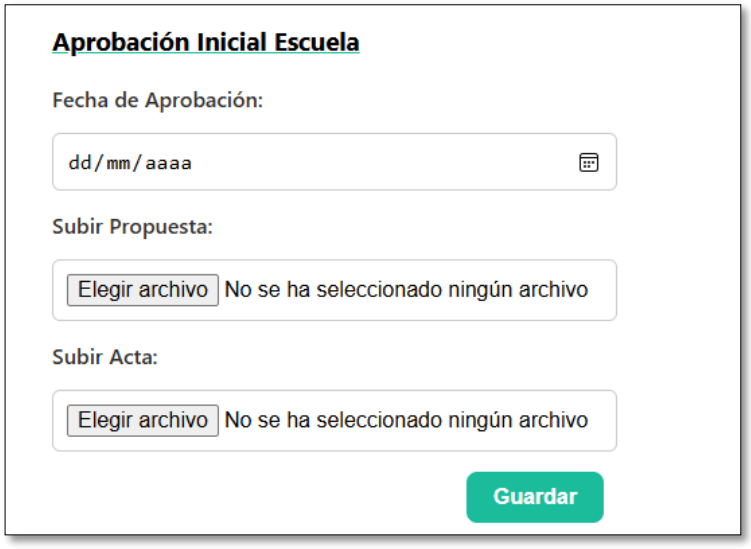


Ilustración 82. Formulario subida de archivos

El cliente incluye validaciones previas a la subida para evitar errores comunes como la comprobación de que se haya seleccionado un archivo. En caso de error, se proporciona retroalimentación inmediata mediante mensajes descriptivos visibles en la interfaz.

Asimismo, en las secciones necesarias se incorpora un listado dinámico de documentos ya cargados, con enlaces de descarga generados a partir de la URL

almacenada en la base de datos -véase *Ilustración 83. Listado descarga de archivos*-.

Ilustración 83. Listado descarga de archivos

4.4.2 Lógica del servidor (backend)

La gestión documental en el servidor se ha abordado mediante el uso del middleware *Multer*, una biblioteca especializada en el manejo de formularios con archivos en aplicaciones *Node.js*. Este middleware permite interceptar y almacenar archivos que los usuarios cargan a través del cliente, controlando tanto la ubicación como la nomenclatura de estos.

Para iniciar, se habilita el análisis de datos provenientes de formularios mediante *express.urlencoded* en el archivo *index.js*, y se definen los módulos necesarios para el manejo de archivos y rutas tal como me muestra en *Ilustración 84. Código configuración entorno Multer*.

```
const multer = require("multer");
const path = require("path");
const fs = require("fs");

//Multer almacenamiento archivos
app.use(express.urlencoded({ extended: true })); // Para datos en formularios
```

Ilustración 84. Código configuración entorno Multer

Por otro lado, se establece como directorio de almacenamiento local la carpeta *uploads* y en el caso de que esta no exista, se crea automáticamente – véase *Ilustración 85. Código directorio almacenamiento Multer*-.

```
const uploadDir = path.join(__dirname, "uploads"); //Carpeta donde se guardarán los archivos

//Si la carpeta "uploads" no existe, la crea automáticamente
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir);
}
```

Ilustración 85. Código directorio almacenamiento Multer

A continuación, se configura el motor de almacenamiento de *Multer*. Esta configuración gestionada con la función *diskStorage* permite especificar la carpeta de destino y una estrategia de nombrado que previene colisiones y facilita la trazabilidad de los archivos -vease *Ilustración 86. Código configuración motor de almacenamiento Multer*-. Además, en este fragmento se reemplazan los espacios en blanco por guiones bajos (_) y se antepone una marca temporal (*timestamp*) al nombre del archivo, asegurando así unicidad para evitar conflictos.

```
//Configuración de multer para almacenamiento de archivos
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "uploads/"); // Guarda los archivos en la carpeta "uploads"
  },
  filename: (req, file, cb) => {
    const sanitizedFilename = file.originalname.replace(/[\s]/g, "_"); // Reemplaza espacios por "_"
    cb(null, Date.now() + "-" + sanitizedFilename);
  },
});

const upload = multer({ storage });
```

Ilustración 86. Código configuración motor de almacenamiento Multer

Cuando se recibe una petición POST a la ruta */api/upload*, el middleware procesa el archivo, lo guarda en el directorio indicado y lo expone posteriormente a través de una URL relativa. Si la operación se completa con éxito, el servidor responde con un objeto JSON que incluye información sobre el archivo y su localización. En caso contrario, se devuelve un mensaje de error con un código HTTP adecuado -véase *Ilustración 87. Código ruta /api/upload*-.

```
// Ruta para subir archivos
app.post("/api/upload", upload.single("archivo"), (req, res) => {
  if (!req.file) {
    return res.status(400).json({ error: "No se subió ningún archivo" });
  }

  res.status(201).json({
    message: "Archivo subido correctamente",
    filename: req.file.filename,
    url: `/uploads/${req.file.filename}`,
  });
});
```

Ilustración 87. Código ruta /api/upload

En lo referente a la descarga de archivos, para facilitar el acceso a los archivos desde el cliente, se habilita el directorio */uploads* como recurso estático – véase *Ilustración 88. Código para descargar archivos estáticos*-.

```
// Servir archivos estáticos para descargar
app.use("/uploads", express.static(path.join(__dirname, "uploads")));
```

Ilustración 88. Código para descargar archivos estáticos

Adicionalmente, se habilita una ruta para la descarga directa de archivos mediante su nombre a través de una petición GET – véase *Ilustración 89. Código ruta /api/download/:filename* -. Esta ruta valida que el archivo solicitado exista antes de proceder a la entrega.

```
// Ruta para descargar un archivo
app.get("/api/download/:filename", (req, res) => {
  const filename  = req.params.filename.trim();
  console.log("Directorio actual:", __dirname);
  const filePath = path.join(__dirname, "uploads", filename);

  console.log("Intentando descargar:", filePath);

  if (!fs.existsSync(filePath)) {
    console.error(" Archivo no encontrado:", filePath);
    return res.status(404).json({ error: "Archivo no encontrado" });
  }

  res.download(filePath, (err) => {
    if (err) {
      console.error("Error al descargar el archivo:", err);
      res.status(500).send("Error al descargar el archivo");
    }
  });
});
```

Ilustración 89. Código ruta /api/download/:filename

Este mecanismo no solo proporciona acceso a los archivos, sino que también permite auditar solicitudes de descarga y controlar errores relacionados con archivos inexistentes.

El sistema complementa estas acciones con el registro de los metadatos del documento en la base de datos, en la tabla documentaciontitulo cuya estructura se muestra en *Ilustración 90. Estructura tabla "documentaciontitulo"*. Este registro se realiza mediante una inserción posterior a la subida, incluyendo el código de la propuesta, el estado del proceso, el nombre original del documento, su URL relativa y la fecha de carga.

Campo	Tipo	Clave	Nulo	Observaciones
ID_Documento	int	PRI	NO	Autoincremental
Codigo_Propuesto	varchar(10)	MUL	NO	Clave foránea
Codigo_Estado	varchar(50)		NO	Estado de la propuesta
Nombre_Documento	varchar(100)		NO	Nombre original del archivo
URL_Documento	varchar(500)		NO	Ruta relativa del archivo
Fecha_Subida	date		SÍ	Por defecto: `CURDATE()`

Ilustración 90. Estructura tabla "documentaciontitulo"

4.5 Reestructuración de la base de datos

Durante el desarrollo del proyecto, se identificó la necesidad de reestructurar la base de datos previamente existente, la cual presentaba diversas inconsistencias estructurales y conceptuales. El modelo inicial, heredado de un trabajo previo, no contemplaba de forma adecuada la integridad de ciertas relaciones, contenía redundancias y carecía de la flexibilidad necesaria para soportar nuevas funcionalidades.

A medida que el sistema fue evolucionando, surgieron nuevos requerimientos funcionales que no podían ser satisfechos de manera eficiente con la estructura original. Esta situación motivó la incorporación progresiva de nuevas tablas y la redefinición de relaciones entre entidades, con el fin de garantizar la coherencia de los datos, facilitar el mantenimiento y permitir una escalabilidad adecuada del sistema. La reestructuración llevada a cabo ha supuesto, por tanto, una mejora sustancial en términos de organización, consistencia e integridad del modelo de datos.

A continuación, se presenta el diagrama entidad-relación que representa el diseño final de la base de datos – véase *Ilustración 91. Diagrama Entidad-Relación*-:

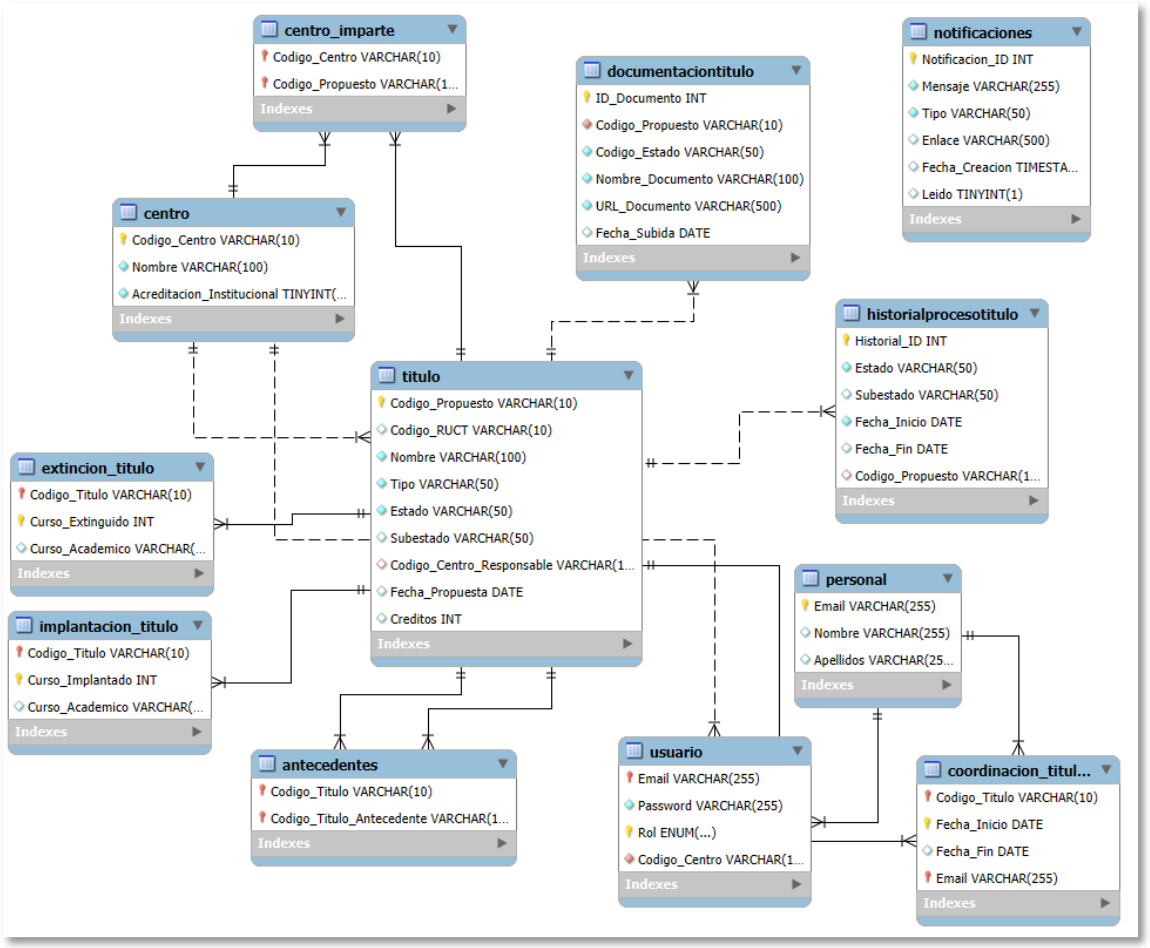


Ilustración 91. Diagrama Entidad-Relación

4.5.1 Descripción de las tablas

Cada una de las tablas que componen la base de datos desempeña un papel específico dentro del sistema y está diseñada para reflejar con fidelidad las entidades y relaciones involucradas en la gestión del ciclo de vida de titulaciones universitarias. A continuación, se presenta una descripción detallada del propósito, estructura y relaciones de cada tabla que integra el modelo.

4.5.1.1 Tabla “titulo”

La tabla *titulo* constituye el núcleo central del modelo de datos, almacenando la información fundamental sobre cada titulación gestionada en el sistema. Cada registro en esta tabla representa un título universitario identificado de manera única mediante el campo *Codigo_Propuesto*, que actúa como clave primaria.

Entre sus atributos principales se incluyen:

- **Codigo_Propuesto**: Clave primaria que actúa como identificador único del título.
- **Codigo_RUCT**: Código oficial asignado en el Registro de Universidades, Centros y Títulos (RUCT), que puede estar vacío en caso de títulos aún no verificados.
- **Nombre**: Denominación oficial del título.
- **Tipo**: Especifica el nivel académico del título, como grado, máster o doctorado.
- **Estado** y **Subestado**: Campos que reflejan la situación administrativa actual del título y posibles estados secundarios o específicos dentro del ciclo de vida.
- **Codigo_Centro_Responsable**: Clave foránea que vincula el título con el centro universitario responsable de su gestión, estableciendo así una relación directa con la tabla **centro**.
- **Fecha_Propuesta**: Fecha en la que comenzó el proceso de diseño del título.
- **Créditos**: Número total de créditos ECTS asociados a la titulación.

Esta estructura garantiza una representación detallada y coherente de los aspectos esenciales de cada título, facilitando la gestión, seguimiento y consulta dentro del sistema. Además, la relación con la tabla **centro** permite integrar la información organizativa de la universidad con la gestión académica.

4.5.1.2 Tabla "centro"

La tabla *centro* almacena la información relativa a los distintos centros universitarios responsables de la gestión de las titulaciones. Cada registro representa un centro, identificado de manera única mediante el campo *Codigo_Centro*, que funciona como clave primaria.

Los principales atributos de esta tabla son:

- **Codigo_Centro**: Identificador único del centro universitario dentro del sistema.
- **Nombre**: Denominación oficial del centro.

- **Acreditacion_Institucional**: Campo booleano que indica si el centro cuenta con acreditación institucional vigente (valor 1) o no (valor 0).

Esta tabla es fundamental para vincular los títulos con las unidades organizativas responsables, facilitando así la gestión descentralizada y el seguimiento administrativo de las titulaciones en función de su centro asignado.

4.5.1.3 Tabla "centro_imparte"

La tabla *centro_imparte* establece la relación entre los centros universitarios y los títulos que estos imparten. Su función principal es reflejar qué centros son responsables de la impartición de cada titulación, permitiendo una gestión multicentro cuando un título puede ser ofrecido por más de un centro.

Esta tabla contiene los siguientes campos:

- **Codigo_Centro**: Identificador del centro universitario que imparte el título, haciendo referencia a la tabla **centro**.
- **Codigo_Propuesto**: Código único del título universitario impartido, vinculado a la tabla **titulo**.

Ambos campos conforman la clave primaria compuesta, garantizando la unicidad de cada asociación entre centro y título. Además, se establecen restricciones de integridad referencial mediante claves foráneas que aseguran la coherencia y sincronización entre los registros de los centros y los títulos asociados, con acciones en cascada para actualizaciones y eliminaciones. Esta estructura es esencial para gestionar correctamente la asignación de titulaciones a los diferentes centros dentro de la Universidad Politécnica de Madrid.

4.5.1.4 Tabla "documentaciontitulo"

La tabla *documentaciontitulo* almacena los documentos asociados a cada título universitario, permitiendo la gestión y seguimiento de la documentación vinculada a las distintas etapas y estados del ciclo de vida de una titulación.

Sus principales campos son:

- **ID_Documento:** Identificador único y autoincremental de cada documento, que actúa como clave primaria.
- **Codigo_Propuesto:** Código del título al que está asociado el documento, vinculado mediante clave foránea a la tabla **titulo**.

- **Codigo_Estado:** Código que indica el estado o tipo de proceso al que corresponde el documento, facilitando su clasificación.
- **Nombre_Documento:** Nombre descriptivo del documento.
- **URL_Documento:** Ruta o enlace donde se encuentra almacenado el documento digitalizado.
- **Fecha_Subida:** Fecha en la que el documento fue añadido al sistema, con valor por defecto la fecha actual.

La relación entre documentos y títulos garantiza que la documentación pertinente a cada titulación quede organizada y accesible, apoyando la trazabilidad y gestión documental requerida para el correcto seguimiento administrativo y académico. La eliminación en cascada asegura que, al eliminar un título, se eliminen automáticamente todos los documentos asociados, manteniendo la integridad de la base de datos.

4.5.1.5 Tabla “historialprocesotitulo”

La tabla *historialprocesotitulo* tiene como finalidad registrar de forma cronológica los distintos estados y subestados por los que transita un título universitario a lo largo de su ciclo de vida, constituyendo así un historial detallado de su evolución administrativa.

Los campos principales son los siguientes:

- **Historial_ID:** Identificador único y autoincremental de cada registro de historial, que actúa como clave primaria.
- **Estado:** Representa el estado principal del proceso en el que se encuentra el título en un momento determinado (por ejemplo, "Diseño", "Verificación", "Implantado").
- **Subestado:** Especifica con mayor detalle la fase concreta dentro del estado principal; este campo es opcional.
- **Fecha_Inicio:** Fecha en la que se inicia el periodo correspondiente al estado registrado.
- **Fecha_Fin:** Fecha en la que finaliza dicho estado; puede ser nula si el estado está aún activo.

- **Codigo_Propuesto:** Código del título al que está asociado el historial, referenciado mediante clave foránea a la tabla **titulo**.

Además, se establece una restricción de unicidad sobre el conjunto de columnas (Estado, Subestado, Fecha_Inicio, Codigo_Propuesto), con el objetivo de evitar duplicidades que podrían comprometer la coherencia temporal del historial. Esta tabla es esencial para garantizar la trazabilidad completa de cada título y permite reconstruir con precisión su trayectoria a lo largo del tiempo.

4.5.1.6 Tabla "notificaciones"

La tabla *notificaciones* tiene como objetivo almacenar los mensajes informativos generados por el sistema para alertar o comunicar eventos relevantes a los usuarios, facilitando así el seguimiento de acciones importantes dentro del proceso de gestión de titulaciones. Esta tabla proviene del trabajo realizado anteriormente y no es utilizada en los subsistemas que abarca este TFG.

Sus principales campos son:

- **Notificacion_ID:** Identificador único y autoincremental que actúa como clave primaria de cada notificación.
- **Mensaje:** Texto descriptivo del contenido de la notificación, limitado a 255 caracteres.
- **Tipo:** Categoría o naturaleza de la notificación (por ejemplo, "informativa", "advertencia" o "error"), que permite su clasificación.
- **Enlace:** Campo opcional que contiene una URL o ruta que redirige al usuario al recurso o sección del sistema relacionada con la notificación.
- **Fecha_Creacion:** Marca temporal que indica la fecha y hora de creación de la notificación. Por defecto, se establece automáticamente en el momento de inserción.
- **Leido:** Campo booleano que indica si la notificación ha sido leída por el usuario (valor 1) o permanece sin leer (valor 0 por defecto).

Esta tabla permite implementar un sistema de alertas dentro de la aplicación, ofreciendo a los usuarios una experiencia más interactiva y un mayor control sobre los cambios y eventos relevantes que afectan al sistema.

4.5.1.7 Tabla “personal”

La tabla *personal* almacena la información básica del personal vinculado al sistema de gestión del ciclo de vida de titulaciones, permitiendo su identificación e interacción dentro de la plataforma.

Sus campos principales son los siguientes:

- **Email:** Dirección de correo electrónico que actúa como identificador único del registro y clave primaria de la tabla. Este campo sirve como referencia principal del usuario dentro del sistema.
- **Nombre:** Nombre propio del miembro del personal.
- **Apellidos:** Apellidos del miembro del personal.

El diseño de esta tabla proporciona una base estructurada para asociar posteriormente a cada usuario con roles, acciones o permisos específicos dentro del sistema, manteniendo un modelo flexible y fácilmente escalable.

4.5.1.8 Tabla “usuario”

La tabla *usuario* representa a los miembros del personal que tienen acceso al sistema y define su rol dentro de la plataforma de gestión del ciclo de vida de titulaciones universitarias. Su diseño permite controlar los permisos y funcionalidades disponibles para cada tipo de usuario, en función de su perfil y centro asociado.

Los campos que la componen son:

- **Email:** Dirección de correo electrónico del usuario, que actúa como identificador principal y clave foránea referenciada desde la tabla **personal**. Este campo, junto con el rol, constituye la clave primaria compuesta de la tabla.
- **Password**: Contraseña cifrada asociada a la cuenta del usuario. Permite su autenticación segura en el sistema.
- **Rol:** Rol funcional que desempeña el usuario dentro del sistema, definido como un valor enumerado entre: jefe-estudios, subdirector-calidad, utc (Unidad Técnica de Calidad) y responsable-titulo. Este campo determina las funcionalidades y vistas accesibles para cada perfil.

- **Codigo_Centro:** Código identificador del centro al que pertenece el usuario, actuando como clave foránea referenciada desde la tabla **centro.** Esta relación permite segmentar el acceso a la información según el ámbito institucional correspondiente.

La estructura de esta tabla garantiza una asignación precisa de permisos y facilita la gestión personalizada de la interacción con el sistema, asegurando trazabilidad y control de acceso según el rol institucional de cada usuario.

4.5.1.9 Tabla "coordinación_titulos"

La tabla *coordinacion_titulos* registra los periodos durante los cuales miembros del personal han ejercido funciones de coordinación sobre titulaciones específicas. Su objetivo es mantener un historial estructurado de las asignaciones de coordinadores a lo largo del tiempo, permitiendo tanto la trazabilidad como la gestión temporal de estas responsabilidades.

Los campos que la componen son:

- **Codigo_Titulo:** Código identificador del título universitario coordinado, actuando como clave foránea referenciada desde la tabla **titulo**.
- **Fecha_Inicio:** Fecha de inicio del periodo en el que el miembro del personal asume la coordinación del título. Forma parte de la clave primaria compuesta.
- **Fecha_Fin:** Fecha de finalización de la coordinación. Puede ser nula para indicar que la coordinación aún está activa en la fecha actual.
- **Email:** Dirección de correo electrónico del personal encargado de la coordinación, actuando como clave foránea referenciada desde la tabla **personal**. Forma parte de la clave primaria compuesta.

La clave primaria compuesta por Email, Codigo_Titulo y Fecha_Inicio asegura la unicidad de cada asignación temporal, permitiendo registrar múltiples coordinaciones sucesivas sobre un mismo título. Esta estructura es fundamental para reflejar con fidelidad la evolución en la asignación de responsabilidades a lo largo del ciclo de vida de una titulación.

4.5.1.10 Tabla “antecedentes”

La tabla *antecedentes* tiene como finalidad representar las relaciones de precedencia entre titulaciones universitarias, permitiendo identificar títulos que actúan como antecedentes de otros. Este mecanismo es esencial para trazar la evolución académica e institucional de los programas formativos.

Los campos que conforman esta tabla son:

- **Codigo_Titulo:** Código del título actual, es decir, el título que posee uno o más antecedentes. Actúa como clave foránea referenciada desde la tabla **titulo**.
- **Codigo_Titulo_Antecedente:** Código del título que se considera antecedente del título actual. También es una clave foránea referenciada desde la tabla **titulo**.

La clave primaria compuesta por ambos campos asegura que no se dupliquen relaciones de antecedencia entre un mismo par de títulos. Además, las restricciones de integridad referencial con eliminación en cascada garantizan la consistencia de los datos en caso de que un título sea eliminado del sistema.

4.5.1.11 Tabla “implantación_titulo”

La tabla *implantacion_titulo* tiene como objetivo registrar de forma detallada el calendario de implantación progresiva de un título universitario, es decir, especificar en qué curso académico se pone en marcha cada uno de los cursos que conforman el plan de estudios del título. Esta información resulta fundamental para la planificación académica y para el seguimiento del desarrollo efectivo del título en el tiempo.

Los campos que componen la tabla son los siguientes:

- **Codigo_Titulo:** Código identificador del título cuya implantación se desea registrar. Actúa como clave foránea referenciada desde la tabla **titulo**.
- **Curso_Implantado:** Número de curso (1.º, 2.º, etc.) que se implanta en un determinado curso académico. Forma parte de la clave primaria compuesta.

- **Curso_Academico:** Texto que indica el curso académico correspondiente (por ejemplo, "2024/2025"), permitiendo contextualizar cronológicamente la implantación.

La clave primaria compuesta por Codigo_Titulo y Curso_Implantado garantiza que no se registren duplicados en la implantación de cursos para un mismo título. Esta tabla es esencial para representar de forma estructurada la progresión temporal de la oferta formativa y facilita la trazabilidad del desarrollo académico de cada titulación dentro del sistema de gestión.

4.5.1.12 Tabla "extinción_titulo"

La tabla *extincion_titulo* tiene como finalidad registrar el proceso de extinción progresiva de un título universitario, indicando qué cursos concretos dejan de impartirse y en qué curso académico sucede dicha extinción. Esta información es fundamental para la gestión académica y administrativa, ya que permite planificar con precisión el cierre ordenado de titulaciones y ofrecer información clara tanto a los órganos de gestión como a los estudiantes afectados.

Los campos de la tabla son:

- **Codigo_Titulo:** Identificador del título al que pertenece el curso que se extingue. Se refiere a la clave primaria de la tabla **titulo** mediante una clave foránea.
- **Curso_Extinguido:** Número del curso que ha dejado de impartirse (por ejemplo, 1.º, 2.º, etc.). Forma parte de la clave primaria compuesta junto con Codigo_Titulo.
- **Curso_Academico:** Texto que representa el curso académico en el que se produce la extinción del curso correspondiente (por ejemplo, "2023/2024").

La clave primaria compuesta asegura que no se repitan registros para la extinción del mismo curso de un título. Esta tabla complementa la información contenida en *implantacion_titulo*, permitiendo gestionar de forma integral tanto el inicio como el fin de la vida académica de cada programa formativo.

4.6 Pruebas realizadas

Con el objetivo de verificar la correcta implementación de las funcionalidades desarrolladas, se han definido y ejecutado una serie de casos de prueba que simulan escenarios reales de uso del sistema. Cada prueba detalla las acciones que debe realizar el usuario, así como el comportamiento esperado por parte del sistema, incluyendo inserciones en la base de datos, mensajes informativos y transiciones de interfaz. A continuación, se describen los principales casos de prueba ejecutados.

4.6.1 Acceso al sistema (login)

Escenario: El usuario accede a la página de inicio de sesión.

Acciones del usuario:

- Introducir email, contraseña y seleccionar el rol asociado a dicho usuario.

Ilustración 92. Caso de prueba Acceso al sistema I

Comportamiento esperado del sistema:

- Verifica las credenciales contra la base de datos.
- Si son correctas, redirige al usuario a la pantalla de inicio correspondiente a su rol.
- Si las credenciales o el rol elegido son incorrectas, se muestra un mensaje de error indicando que el email, el rol o la contraseña son inválidos.

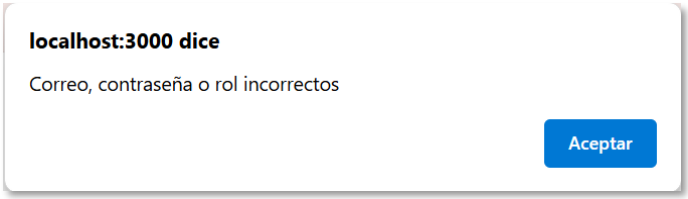


Ilustración 93. Caso de prueba Acceso al sistema II

4.6.2 Cambio de contraseña

Escenario: Un usuario autenticado desea actualizar su contraseña.

Acciones del usuario:

- Accede a la sección "Cambiar contraseña" pulsando el botón correspondiente en la barra lateral de su página de inicio -véase *Ilustración 8. Interfaz de inicio para el rol Jefe de Estudios*-.
- Introduce la contraseña actual y la nueva contraseña – véase *Ilustración 13.Interfaz de formulario para el cambio de contraseña*-.

Comportamiento esperado del sistema:

- Verifica la contraseña actual.
- Si es correcta, actualiza la nueva contraseña en la base de datos tras cifrarla.
- Se muestra un mensaje confirmando el cambio – vease *Ilustración 94. Caso de prueba cambio de contraseña (correcto)*.

Ilustración 94. Caso de prueba cambio de contraseña (correcto)

- Si la verificación falla, se notifica al usuario – véase *Ilustración 95. Caso de prueba cambio de contraseña (incorrecto)*-.

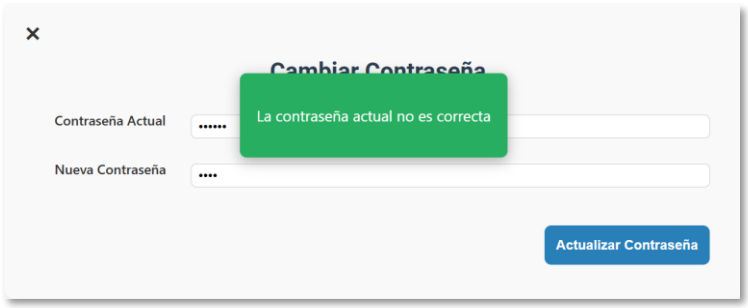


Ilustración 95. Caso de prueba cambio de contraseña (incorrecto)

4.6.3 Registro de responsable de título

Escenario: Un usuario con permisos (jefe de estudios) registra a un nuevo responsable de título.

Acciones del usuario:

- Accede al formulario de registro pulsando el botón "Administración" de la pantalla de inicio – véase *Ilustración 8. Interfaz de inicio para el rol Jefe de Estudios*- y una vez ahí pulsando en el botón "Registrar Responsable de Título" – véase *Ilustración 15. Interfaz vista Responsables de Titulo*-.
- Introduce los datos personales y selecciona el título correspondiente – véase *Ilustración 16. Interfaz formulario Registro de Responsable de Título*-.

Comportamiento esperado del sistema:

- Inserta el nuevo responsable en las tablas personal y usuario (en el caso de que no se encuentre ya en ellas) y en la tabla coordinacion_titulos.
- En el caso de que ya exista un responsable asociado al título en cuestión, se modifica el registro asociado al responsable actual en la tabla coordinación-titulos completando el campo Fecha_Fin con la fecha actual. También comprueba si es responsable actualmente de algún otro título, si no lo es se elimina en la tabla usuarios el registro asociado con el rol responsable-titulo para ese coordinador.
- Se genera un mensaje de confirmación.
- El nuevo usuario puede acceder al sistema con las credenciales asignadas.

4.6.4 Diseño de nuevo título

Escenario: Un jefe de estudios desea registrar una nueva propuesta de título.
Acciones del usuario:

- Accede al apartado "Diseñar títulos" desde la pantalla de inicio de su perfil - véase *Ilustración 8. Interfaz de inicio para el rol Jefe de Estudios*- y ahí pulsa el botón "Diseñar nuevo título" – véase *Ilustración 23. Interfaz visualización de títulos en proceso de diseño* -.
- Rellena los campos requeridos del formulario: nombre, tipo y créditos – véase *Ilustración 28. Formulario diseño nuevo título.*-.

Comportamiento esperado del sistema:

- Inserta un nuevo registro en la tabla titulo con estado inicial "Diseño" y subestado "Fase Inicial".
- Se genera automáticamente una entrada en el historial del proceso (historialprocesotitulo).
- Se genera un mensaje de confirmación -véase *Ilustración 96. Caso de prueba "Diseño de Nuevo Titulo" I*.

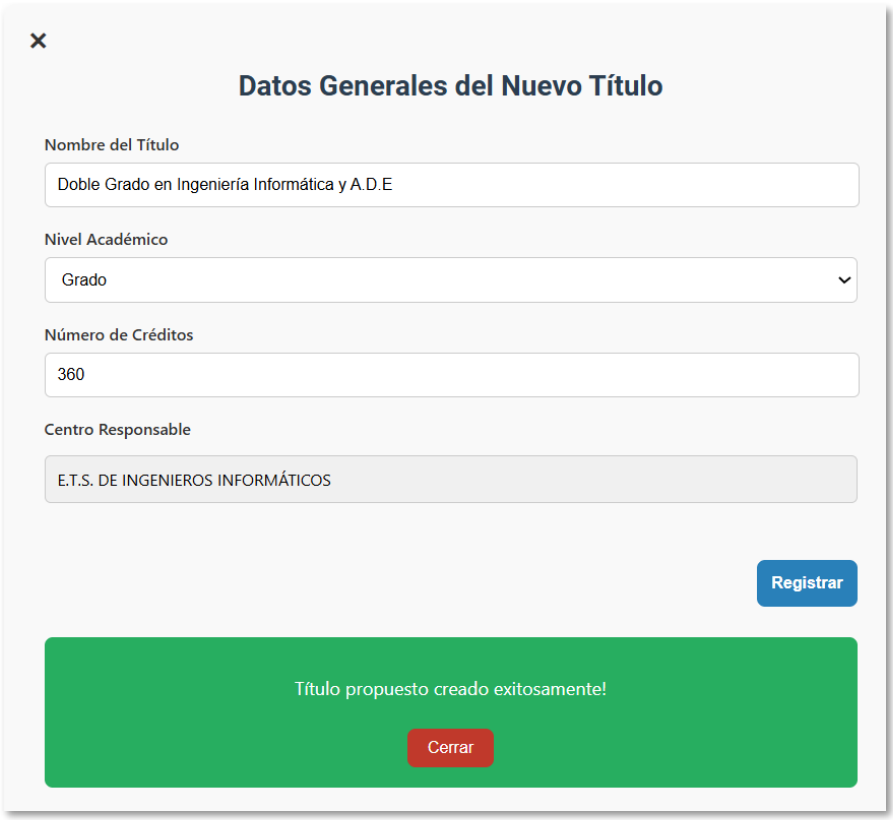


Ilustración 96. Caso de prueba "Diseño de Nuevo Titulo" I.

- Se muestra el nuevo título en la tabla "Títulos en Diseño" -véase *Ilustración 97. Caso de prueba "Diseño de Nuevo Titulo" II.*

Ilustración 97. Caso de prueba "Diseño de Nuevo Titulo" II.

4.6.5 Añadir centros participantes

Escenario: El jefe de estudios desea asociar uno o más centros que también impartirán dicho título.

Acciones del usuario:

- Accede a la información general del título – véase *Ilustración 33. Interfaz detalle de título en diseño*- desde la sección “Diseñar títulos” de la pantalla de inicio, pulsando sobre el nombre del título del cual se desean añadir los centros participantes .
- En la sección de "Centros Participantes", selecciona uno de los centros disponibles desde un desplegable.
- Confirma la asociación haciendo clic en el botón “Añadir centro” – véase *Ilustración 98. Caso de prueba "Añadir centros participantes" I*-.

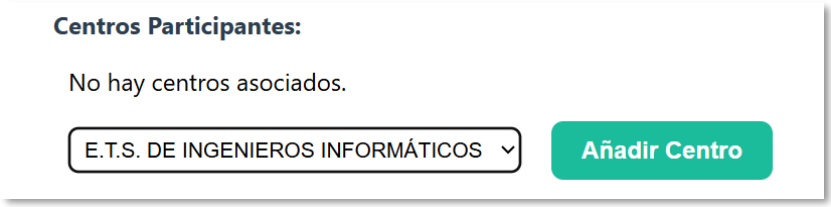


Ilustración 98. Caso de prueba "Añadir centros participantes" I

Comportamiento esperado del sistema:

- Se inserta un registro en la tabla centro_imparte, estableciendo la relación entre el código del título (Codigo_Propuesto) y el centro seleccionado (Codigo_Centro).
- La lista de centros asociados se actualiza dinámicamente.

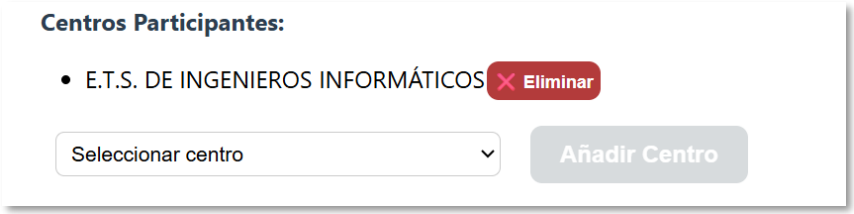


Ilustración 99. Caso de prueba "Añadir centros participantes" II

4.6.6 Eliminar centros participantes

Escenario: Un usuario con permisos desea eliminar la asociación de un centro que ya no participa en la impartición de un título.

Acciones del usuario:

- Accede a la información general del título – véase *Ilustración 33. Interfaz detalle de título en diseño*- desde la sección “Diseñar títulos” de la pantalla de inicio, pulsando sobre el nombre del título del cual se desean añadir los centros participantes .
- En la sección "Centros asociados", localiza el centro que desea eliminar.
- Hace clic en el botón “Eliminar” junto al nombre del centro – véase *Ilustración 99. Caso de prueba "Añadir centros participantes" II*-.

Comportamiento esperado del sistema:

- El sistema solicita confirmación al usuario para proceder con la eliminación.

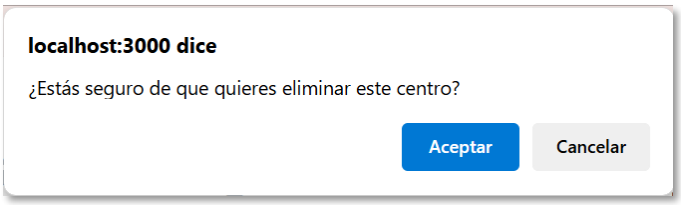


Ilustración 100. Caso de prueba "Eliminar centros participantes" I

- Una vez confirmada, se elimina el registro correspondiente de la tabla centro_imparte que vincula el centro con el título.
- Se actualiza automáticamente la lista de centros asociados mostrada al usuario.
- Se muestra un mensaje indicando que la operación se ha realizado correctamente – véase *Ilustración 101. Caso de prueba "Eliminar centros participantes" II*.

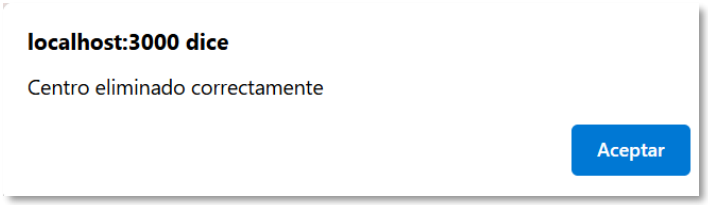


Ilustración 101. Caso de prueba "Eliminar centros participantes" II

4.6.7 Secciones de aprobación en títulos en diseño

Escenario: Un usuario accede a un título que se encuentra en estado de *Diseño* y desea cumplimentar y registrar las distintas secciones del apartado de "Aprobación", necesarias para la verificación del título.

Acciones del usuario:

- Accede a la sección de aprobación la cual quiera cumplimentar (Aprobación Inicial Escuela, Aprobación Inicial Universidad, Aprobación Memoria Escuela o Aprobación Memoria Universidad) desde la sección "Diseñar títulos" de la pantalla de inicio, pulsando sobre el nombre del título que se desee completar.
- Cumplimenta un formulario indicando la fecha de aprobación correspondiente, sube la documentación requerida y guarda los cambios – *véase Ilustración 44. Interfaz formularios subida documentos de títulos en diseño*- .
- El usuario puede volver en otro momento para completar otras secciones pendientes o para visualizar el contenido del formulario cumplimentado previamente – véase *Ilustración 45. Interfaz resultados formularios subida documentos de títulos en diseño*-.

Comportamiento esperado del sistema:

- El sistema permite visualizar únicamente las secciones habilitadas para el rol del usuario actual.
- Al guardar una sección, se valida que los campos obligatorios estén completos y los documentos adjuntos cumplan con el formato requerido.
- Si la validación es correcta, se guarda la información en la base de datos y se muestra un mensaje de éxito – véase *Ilustración 102. Caso de prueba "Secciones de aprobación en títulos en diseño" I*-.

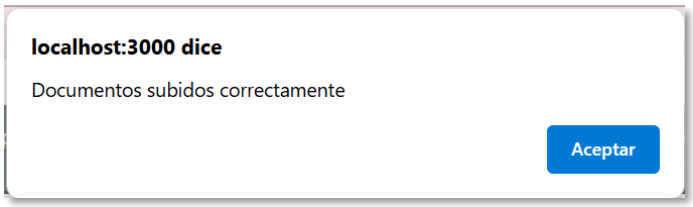


Ilustración 102. Caso de prueba "Secciones de aprobación en títulos en diseño" I

- El sistema registra la fecha de modificación en la que se realizó la acción y el nuevo subestado dentro de la tabla historialprocesotitulo .
- Si alguna sección queda incompleta o no válida, se indicará visualmente – véase *Ilustración 103. Caso de prueba "Secciones de aprobación en títulos en diseño" II-*.

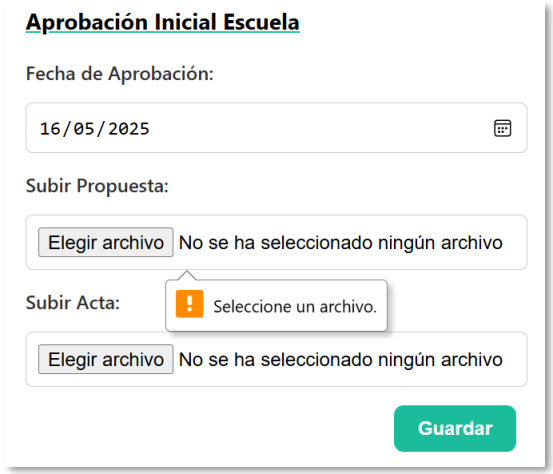


Ilustración 103. Caso de prueba "Secciones de aprobación en títulos en diseño" II

- Una vez todas las secciones estén completadas correctamente, el sistema podrá habilitar el paso a estado *EnVerificación* desde el siguiente apartado.

4.6.8 Solicitar verificación de título en diseño

Escenario: Un usuario desea solicitar la verificación de un título actualmente en estado *En Diseño*. Para ello, debe haber completado previamente todas las secciones requeridas del proceso de diseño, lo que habilita el acceso a esta funcionalidad.

Acciones del usuario:

- Navega al listado de títulos y selecciona uno que esté en estado *En Diseño*.

- Accede a la ficha del título y verifica que todas las secciones obligatorias (Aprobación Inicial Escuela, Aprobación Inicial Universidad, Aprobación Memoria Escuela y Aprobación Memoria Universidad) han sido completadas correctamente.
- Al cumplir estos requisitos, aparece habilitada una nueva sección o área con el botón "Solicitar verificación" -véase *Ilustración 51. Interfaz Esperando Verificación (Botón)*-.
- El usuario pulsa dicho botón para formalizar la solicitud.

Comportamiento esperado del sistema:

- El sistema comprueba automáticamente que todas las secciones necesarias del proceso de diseño estén completas.
- Al pulsar "Solicitar verificación" el estado del título cambia de *En Diseño* a *En Verificación* y se muestra un mensaje de confirmación – véase *Ilustración 104. Caso de prueba "Solicitar verificación de título en diseño"*-.

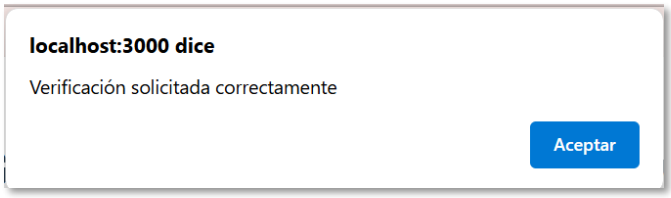


Ilustración 104. Caso de prueba "Solicitar verificación de título en diseño"

- Se inserta un nuevo registro en la tabla *historialprocesotitulo* con el nuevo estado “Verificado” y la fecha en la que se ha solicitado la verificación.
- Tras pulsar el botón, este desaparece de la interfaz y se muestra la fecha en la que se ha solicitado la verificación – véase *Ilustración 52. Interfaz Esperando Verificación (Fecha)*-.
- Se habilita automáticamente la sección “Verificado” para que el usuario pueda comenzar a cumplimentarla.

4.6.9 Verificación de título en diseño

Escenario: Un usuario autorizado desea cumplimentar la sección “Verificado” de un título que se encuentra en estado “En Verificación”.

Acciones del usuario:

- Navega al listado de títulos y selecciona uno que esté en estado *En Diseño*. – véase *Ilustración 23. Interfaz visualización de títulos en proceso de diseño*-.
- Entra en la sección "Verificado" habilitada para ese título – véase *Ilustración 59. Interfaz formulario sección "Verificado"*-.
- Cumplimenta los campos requeridos (fechas y código RUCT).
- Sube los documentos necesarios asociados al proceso de verificación.
- Pulsa el botón "Guardar" para almacenar los datos.

Comportamiento esperado del sistema:

- Verifica que el usuario tiene permisos suficientes para acceder y editar el título.
- Carga correctamente la información previa y habilita la edición.
- Al pulsar el botón "Guardar" valida que todos los campos requeridos han sido cumplimentados y muestra un mensaje de confirmación.
- Registra los documentos subidos en la tabla documentaciontitulo.
- Se inserta un nuevo registro en la tabla *historialprocesotitulo* con el nuevo estado y la fecha en la que se ha realizado el cambio de estado.
- Muestra un mensaje de confirmación: "Verificación guardada correctamente".
- Cambia el estado del título a "Verificado" y lo traslada a la tabla "Títulos Vigentes" – véase *Ilustración 62. Interfaz vista "Títulos Oficiales"*- a la cual se puede acceder pulsando en el botón "Gestionar Títulos Oficiales" desde la pantalla de inicio.
- Permite el acceso a la sección de implantación si se ha completado toda la verificación.

4.6.10 Descargar documentación de un título

Escenario: Un usuario autenticado desea descargar la documentación asociada a un título.

Acciones del usuario:

- Accede a la información general del título – véase *Ilustración 69. Interfaz "Información General" de título vigente*- desde la sección “Gestionar Títulos Oficiales” de la pantalla de inicio, pulsando sobre el nombre del título del cual se desea descargar la documentación.
- Accede a la sección de documentación del título pulsando sobre el botón “Descargar Documentación del Título”.
- Pulsa sobre el enlace "Descargar" asociado a un archivo específico – véase *Ilustración 70. Interfaz de descarga de documentación asociada a un título*-.

Comportamiento esperado del sistema:

- Verifica que el usuario tenga los permisos necesarios para acceder a la documentación del título seleccionado.
- Genera la descarga directa del archivo solicitado.
- Si el archivo no existe o hay un error en el servidor, muestra un mensaje de error.

4.6.11 Registrar antecedente de un título

Escenario: Un usuario autenticado con permisos adecuados desea registrar un título antecedente para un título distinto, con el fin de reflejar la continuidad académica o administrativa entre ambos.

Acciones del usuario:

- Accede a la sección de antecedentes de un título – véase *Ilustración 73. Interfaz sección "Antecedentes del Título"* - desde la sección “Gestionar Títulos Oficiales” de la pantalla de inicio, pulsando sobre el nombre del título del cual se desean añadir los antecedentes.
- Selecciona de un desplegable el título existente que actuará como antecedente.
- Pulsa el botón "Añadir antecedente".

Comportamiento esperado del sistema:

- Verifica que el título seleccionado como antecedente sea válido (es decir, distinto del actual y no registrado previamente como antecedente del mismo título).
- Inserta un nuevo registro en la tabla antecedentes de la base de datos, relacionando el código del nuevo título con el del título antecedente.
- Muestra un mensaje de confirmación al usuario indicando que el antecedente ha sido registrado correctamente.
- Actualiza la vista para mostrar la relación recién añadida en la tabla de antecedentes.

4.6.12 Eliminar antecedente de un título

Escenario: Un usuario autenticado desea eliminar la relación de antecedencia entre un título y uno de sus títulos predecesores registrados previamente.

Acciones del usuario:

- Accede a la sección de antecedentes de un título – véase *Ilustración 73. Interfaz sección "Antecedentes del Título"* - desde la sección "Gestionar Títulos Oficiales" de la pantalla de inicio, pulsando sobre el nombre del título del cual se desean añadir los antecedentes .
- Localiza en la lista de antecedentes el título antecedente que desea eliminar.
- Pulsa el botón de eliminación asociado a dicho registro.
- Confirma la acción en el cuadro de diálogo que aparece solicitando confirmación – véase *Ilustración 105. Caso de prueba "Eliminar antecedente de un título"*-.

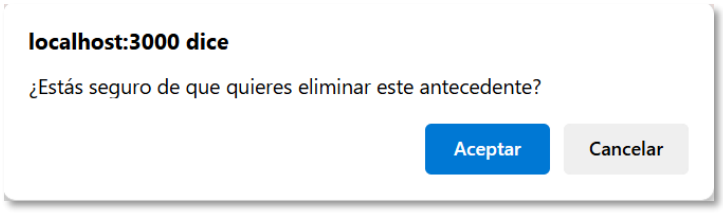


Ilustración 105. Caso de prueba "Eliminar antecedente de un título"

Comportamiento esperado del sistema:

- El sistema solicita confirmación al usuario antes de proceder con la eliminación para evitar acciones accidentales.
- Una vez confirmada la acción, elimina el registro correspondiente de la tabla antecedentes de la base de datos.
- Muestra un mensaje notificando al usuario que el antecedente ha sido eliminado correctamente.
- Actualiza la vista para reflejar los cambios, eliminando visualmente la relación borrada.

4.7 Instrucciones para instalación y uso

Para la correcta instalación y ejecución del sistema desarrollado, es necesario disponer de un entorno de desarrollo con los componentes adecuados tanto para la gestión de la base de datos como para el despliegue del *backend* y del *frontend*. A continuación, se detallan los pasos necesarios.

4.7.1 Requisitos previos

Antes de ejecutar la aplicación, es necesario tener instaladas las siguientes herramientas:

- **Node.js y npm:** Plataforma y gestor de paquetes necesarios para ejecutar el servidor y la aplicación web. Puede descargarse desde: https://nodejs.org
- **MySQL Server:** Motor de base de datos donde residirá la información. Disponible en: https://dev.mysql.com/downloads/mysql
- **MySQL Workbench:** Herramienta gráfica para gestionar la base de datos. Puede obtenerse en: https://dev.mysql.com/downloads/workbench

Además, el proyecto requiere instalar dos dependencias específicas de Node.js: *multer* (para la gestión de archivos) y *bcrypt* (para el cifrado de contraseñas). Estas se instalarán automáticamente mediante el comando *npm install* desde el directorio raíz del *backend*, en este caso será sistema-titulaciones.

4.7.2 Preparación de la base de datos

Antes de poder ejecutar la aplicación, es necesario preparar el entorno de base de datos importando el esquema y estableciendo la conexión con el servidor MySQL.

1. Instalar y configurar MySQL Server y MySQL Workbench.
2. Abrir MySQL Workbench y conectar con el servidor MySQL -véase *Ilustración 106. Conexión servidor MySQL Workbench*-.

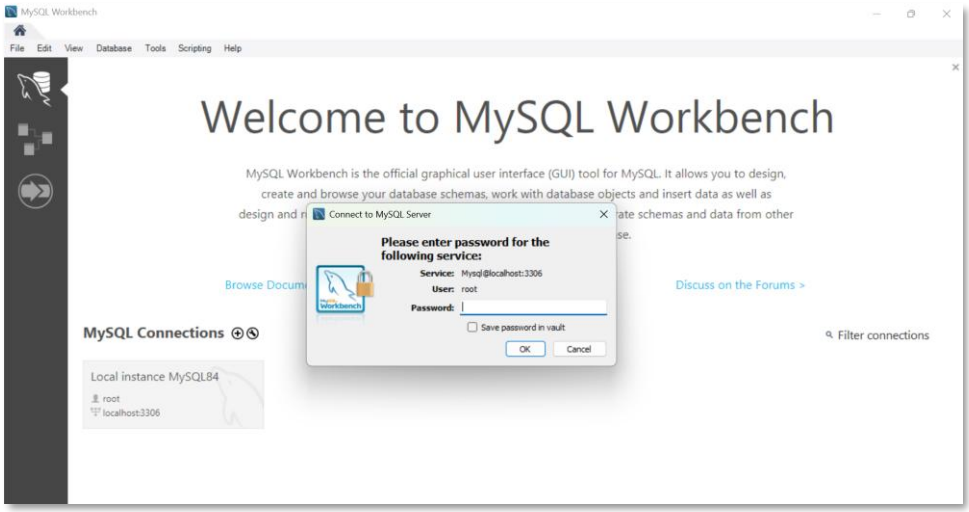


Ilustración 106. Conexión servidor MySQL Workbench

3. Importar el esquema de la base de datos utilizando el archivo proporcionado: `gestion_titulaciones.sql`. Este proceso creará las tablas necesarias y las relaciones correspondientes e incluirá los datos cargados previamente en dichas tablas.

Además, está pendiente la configuración del sistema para permitir la conexión a la base de datos mediante un usuario distinto al usuario por defecto *root*. Para ello, es necesario crear un nuevo usuario en MySQL con los permisos adecuados sobre la base de datos utilizada por la aplicación. Esto puede realizarse ejecutando los siguientes comandos en MySQL:

```
CREATE USER 'nombre_usuario'@'localhost' IDENTIFIED BY 'contraseña';
GRANT ALL PRIVILEGES ON nombre_base_datos.* TO 'nombre_usuario'@'localhost';
FLUSH PRIVILEGES;
```

Ilustración 107. Comandos creación usuario MySQL

Una vez creado el usuario, es imprescindible actualizar los parámetros de conexión a la base de datos en el archivo de configuración del backend (*index.js*), sustituyendo las credenciales actuales por las del nuevo usuario. De este modo, se mejora la seguridad del sistema al evitar el uso del usuario *root* para operaciones cotidianas y se ajusta el acceso a la base de datos según el principio de privilegios mínimos.

4.7.3 Ejecución del servidor backend

Una vez configurada correctamente la base de datos, el siguiente paso consiste en poner en marcha el servidor *backend*, que será el encargado de gestionar las operaciones relacionadas con la lógica de negocio, la validación de datos y la comunicación con la base de datos – véase *Ilustración 108. Ejecución del servidor*-.

Para ello, es necesario abrir una terminal del sistema operativo y situarse en el directorio raíz del proyecto, que contiene el código del *backend*, mediante el comando `cd sistema-titulaciones`.

Desde esta ubicación, se debe ejecutar el siguiente comando para iniciar el servidor: `npm start`.

Si todo está correctamente configurado, debería mostrarse un mensaje en consola indicando que la conexión con la base de datos MySQL se ha establecido correctamente, lo que confirma que el *backend* está operativo y preparado para recibir peticiones.

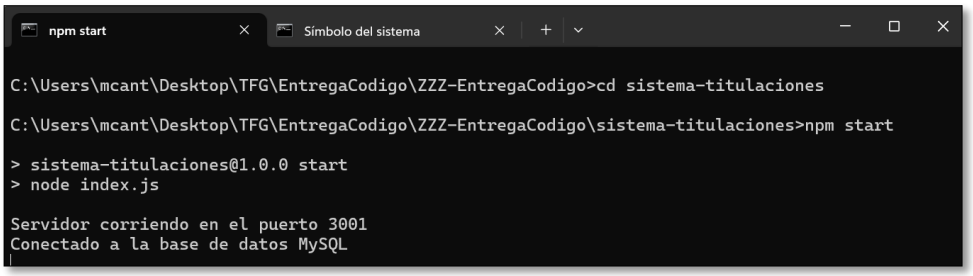


Ilustración 108. Ejecución del servidor

4.7.4 Ejecución del cliente frontend

Una vez el servidor *backend* está activo, se puede lanzar la interfaz de usuario desarrollada con React. Para ello, se recomienda abrir una nueva pestaña o

ventana del terminal, y desde ahí acceder al subdirectorio correspondiente al cliente web mediante el comando `cd sistema-titulaciones/frontend`.

A continuación, se debe ejecutar de nuevo el comando `npm start` para iniciar el entorno de desarrollo del *frontend*.

Este comando arrancará la aplicación React, que se abrirá automáticamente en el navegador web predeterminado, permitiendo al usuario interactuar con el sistema de forma visual. Durante la ejecución, el *frontend* establecerá comunicación con el *backend* para consultar y modificar los datos almacenados en la base de datos – véase *Ilustración 109. Ejecución del cliente*-.

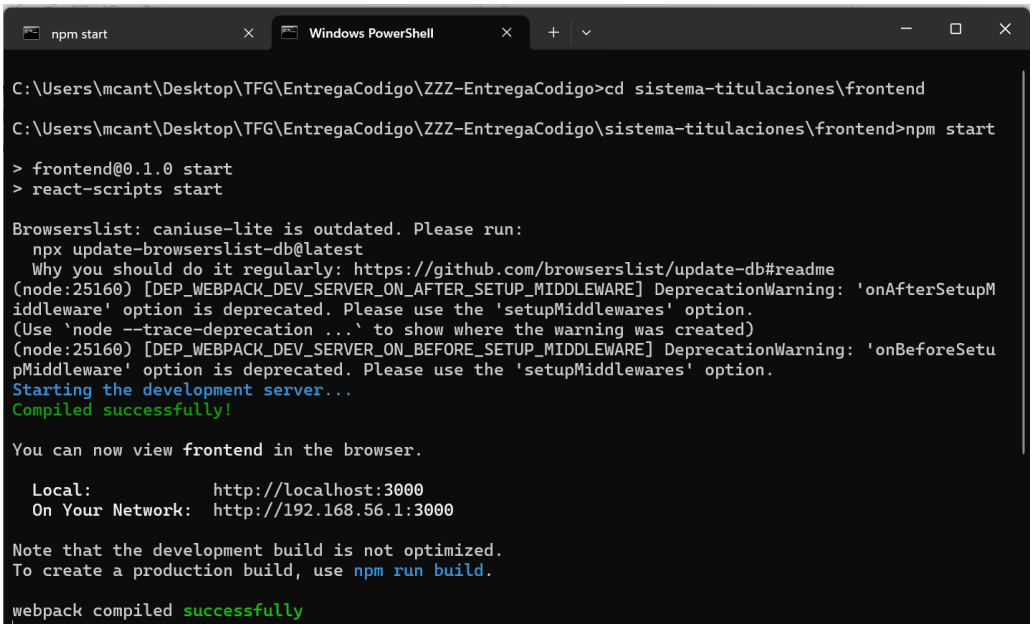


Ilustración 109. Ejecución del cliente

Esta sección proporciona una guía clara y ordenada que permite replicar fácilmente el entorno de ejecución del sistema, asegurando así su correcto funcionamiento en cualquier equipo compatible.

4.7.5 Acceso al sistema desde un ordenador externo

Para permitir el acceso al sistema desde un ordenador externo, es necesario que el servidor *backend* esté desplegado en una máquina con dirección IP pública o accesible dentro de una red local con la configuración adecuada. En primer lugar, se debe modificar el archivo de configuración del servidor *index.js* para que escuche en la IP de la máquina (en lugar de *localhost*) y en un puerto

accesible. Además, es imprescindible asegurarse de que el *firewall* del sistema permita conexiones entrantes a dicho puerto.

Desde el equipo remoto, se podrá acceder al *frontend* introduciendo en el navegador la dirección http://<IP-del-servidor>:<puerto>. En entornos de desarrollo o redes locales, es importante tener ambos dispositivos conectados a la misma red y, en caso necesario, configurar adecuadamente el CORS y las variables de entorno del cliente para que apunten a la IP del *backend*. Para entornos de producción, se recomienda emplear servicios de despliegue o servidores en la nube que ofrezcan mayor seguridad y disponibilidad.

5 Resultados y conclusiones

5.1 Resultados del trabajo

El presente Trabajo Fin de Grado ha culminado con el diseño y desarrollo de un sistema web para la gestión del ciclo de vida de títulos universitarios, orientado a facilitar y sistematizar los procesos administrativos y académicos asociados a la creación, seguimiento, implantación y extinción de titulaciones. A lo largo del proyecto se han obtenido resultados significativos que permiten valorar positivamente el alcance del trabajo realizado.

En primer lugar, se ha logrado implementar un sistema funcional compuesto por un *frontend* desarrollado en React y un *backend* en Node.js, apoyado por una base de datos MySQL. Este sistema permite la autenticación de usuarios, la gestión de distintos roles y perfiles, y la realización de tareas clave como el diseño de nuevos títulos, la asignación de responsables, la gestión documental y la consulta del estado actual de cada titulación.

Asimismo, se ha diseñado e implementado una base de datos relacional completa, con integridad referencial y un modelo de datos optimizado, que respalda las funcionalidades del sistema. Además, se han desarrollado interfaces de usuario intuitivas y accesibles que cubren con éxito los requisitos funcionales definidos en las fases iniciales del proyecto.

Durante el desarrollo se han documentado y ejecutado numerosos casos de prueba, los cuales han permitido verificar el correcto funcionamiento del

sistema en los distintos escenarios de uso, incluyendo la transición entre estados del título, la carga y descarga de documentación, la asignación de antecedentes, y la validación de acciones según los permisos del usuario.

Debido a limitaciones de tiempo, no ha sido posible implementar completamente algunas secciones del sistema, como el proceso detallado de implantación y extinción de títulos. No obstante, se han desarrollado prototipos de baja fidelidad de dichas funcionalidades, junto con una planificación detallada de su lógica de negocio y su futura integración en el sistema.

En resumen, los resultados obtenidos reflejan la viabilidad técnica del sistema propuesto, así como su potencial utilidad para mejorar la eficiencia, trazabilidad y calidad de los procesos relacionados con la gestión de titulaciones en el entorno universitario.

5.2 Líneas de trabajo futuro

A pesar de los avances conseguidos durante el desarrollo del Trabajo Fin de Grado, existen varias líneas de trabajo futuro que permitirían ampliar, mejorar y consolidar la solución propuesta, así como su integración definitiva en un entorno productivo.

En primer lugar, una prioridad clara es la implementación completa de las secciones correspondientes al proceso de implantación y proceso de extinción de los títulos. Aunque actualmente se dispone de prototipos de baja fidelidad que definen su estructura y lógica de funcionamiento, será necesario desarrollar e integrar estas funcionalidades en el sistema, permitiendo así cubrir todo el ciclo de vida de un título universitario de forma automatizada y coherente.

Otra línea de desarrollo relevante será la implementación de casos de uso relacionados con la gestión de la oferta académica, que han sido prototipados en trabajos anteriores [2].

Asimismo, se propone la ampliación del sistema para incluir la gestión de los planes de estudios asociados a los títulos. Esto implicará definir e integrar estructuras curriculares completas, con asignaturas, competencias, resultados

de aprendizaje, metodologías y sistemas de evaluación, alineándose con los estándares exigidos en los procesos de verificación y acreditación.

Una evolución lógica del sistema será también llevar a cabo el diseño, prototipado e implementación de nuevas funcionalidades orientadas a la gestión de convocatorias, especialmente aquellas relacionadas con la Renovación de la Acreditación (RA), el seguimiento de títulos y la solicitud del Sello de Calidad. Estas tareas, altamente demandadas por las agencias evaluadoras, representan hitos críticos dentro del ciclo de vida de un título, y su informatización permitiría optimizar significativamente los tiempos de gestión y asegurar una mayor calidad documental.

Paralelamente, se plantea la carga masiva de datos históricos en la base de datos, lo que permitirá poner a prueba el rendimiento del sistema con volúmenes reales, facilitar pruebas funcionales más completas y servir de base para un análisis longitudinal de la evolución de los títulos.

Finalmente, una fase clave será el despliegue del sistema en un entorno real de pruebas dentro de la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF). Esta acción permitirá evaluar el sistema en condiciones operativas, identificar barreras en la adopción y formar a los usuarios finales, como paso previo a una eventual puesta en producción.

En conjunto, estas líneas de trabajo futuro configuran un ambicioso, pero coherente plan de evolución del sistema, orientado a dotar a las universidades de una herramienta potente, flexible y adaptada a sus procesos de garantía de calidad, planificación académica y gestión del ciclo de vida de los títulos.

5.3 Conclusiones

La realización de este Trabajo de Fin de Grado ha supuesto una experiencia enriquecedora tanto a nivel académico como personal. A lo largo del desarrollo del sistema, he tenido la oportunidad de aplicar de forma práctica los conocimientos adquiridos durante el grado, enfrentándome a retos reales de diseño, modelado y desarrollo de software en un contexto institucional complejo como es la gestión universitaria.

Además, el proyecto me ha permitido familiarizarme y trabajar con tecnologías y herramientas que no había utilizado previamente, como React, Node.js o bibliotecas específicas para la gestión de formularios, subida de archivos y cifrado de contraseñas. Este proceso de aprendizaje ha sido clave para ampliar mis competencias técnicas y adaptarme a entornos de desarrollo modernos, orientados a la construcción de aplicaciones web funcionales y robustas.

Si bien no ha sido posible completar la implementación de todas las funcionalidades inicialmente previstas, el trabajo realizado constituye una base sólida y extensible sobre la que construir futuras mejoras. La estructuración de los procesos relacionados con el ciclo de vida de los títulos, así como el enfoque modular y escalable del sistema, permiten visualizar con claridad su potencial impacto en la digitalización y modernización de las universidades.

Personalmente, este proyecto me ha permitido profundizar en metodologías de desarrollo orientadas a la mejora de la calidad y la eficiencia, reforzando mi compromiso con el diseño de soluciones útiles y sostenibles que respondan a necesidades reales.

6 Análisis de Impacto

Los resultados obtenidos durante la realización de este Trabajo Fin de Grado tienen un impacto significativo en diversos contextos. La informatización de los procesos de gestión de titulaciones universitarias —que en muchos casos todavía se realizan de forma manual o mediante procedimientos poco estructurados— representa un avance notable en términos de eficiencia y organización institucional.

En primer lugar, desde un punto de vista **operativo y empresarial**, la digitalización de procesos como el diseño, verificación, implantación y extinción de títulos supone una mejora sustancial en la eficiencia administrativa. El sistema desarrollado permite automatizar tareas repetitivas, centralizar la información, y reducir la posibilidad de errores humanos, lo cual se traduce en un ahorro considerable de tiempo y recursos. Además, al contar con una interfaz intuitiva y accesible, se mejora la experiencia del usuario y se facilita el trabajo diario de los responsables académicos implicados en estos procedimientos.

Desde una perspectiva **social y cultural**, la adopción de este tipo de herramientas tecnológicas impulsa una transformación digital en el ámbito universitario, fomentando una cultura organizativa más moderna, ágil y orientada a la calidad. Aunque dicha transformación puede requerir un periodo de adaptación y formación para los usuarios, a medio y largo plazo se traduce en un entorno laboral más dinámico, eficaz y alineado con las nuevas demandas del entorno educativo.

En cuanto al **impacto personal**, la realización de este proyecto ha supuesto una experiencia formativa de gran valor, permitiendo consolidar conocimientos en tecnologías web, bases de datos relacionales, diseño de interfaces y metodologías de desarrollo. Asimismo, ha implicado el desarrollo de habilidades transversales como la planificación, documentación, trabajo autónomo y pensamiento crítico, todos ellos aspectos esenciales en el futuro profesional de un ingeniero informático.

Desde un punto de vista **económico**, la solución propuesta, basada en tecnologías de código abierto (Node.js, React, MySQL), representa una alternativa sostenible y de bajo coste para instituciones educativas que desean mejorar su gestión académica sin realizar grandes inversiones. La reutilización y escalabilidad del sistema permite además su posible adopción por otras universidades, reduciendo costes de desarrollo y promoviendo sinergias interinstitucionales.

En el plano **medioambiental**, la informatización de los procesos descritos contribuye a la reducción del uso de papel y materiales físicos, lo que se alinea con una gestión más sostenible y respetuosa con el entorno. Aunque el impacto ecológico directo puede parecer limitado, la transición hacia un modelo sin papel constituye una práctica responsable dentro del marco más amplio de sostenibilidad institucional.

En relación con los **Objetivos de Desarrollo Sostenible (ODS)** [8] de la Agenda 2030, el trabajo presentado contribuye principalmente a los siguientes:

- **ODS 4: Educación de calidad**, al mejorar la planificación, trazabilidad y evaluación de las titulaciones universitarias, lo que redunda en una mayor calidad del sistema educativo.
- **ODS 9: Industria, innovación e infraestructura**, mediante el desarrollo de una infraestructura tecnológica moderna que promueve la innovación en la gestión institucional.
- **ODS 12: Producción y consumo responsables**, al reducir la necesidad de soporte físico en los procedimientos administrativos, fomentando prácticas responsables y sostenibles.

7 Bibliografía

[1] Á. López Martínez, “Diseño de un Sistema de Gestión del Ciclo de Vida de Titulaciones Universitarias”, Trabajo de Fin de Grado, Univ. Politec. Madr., Madrid, 2024.

[2] A. Mendoza Carpintero, “Desarrollo de un sistema de gestión del ciclo de vida de titulaciones universitarias”, Trabajo de Fin de Grado, Univ. Politec. Madr., Madrid, 2025.

[3] React Documentation. React: A JavaScript Library for Building User Interfaces. Disponible en: https://reactjs.org/docs/getting-started.html

[4] Node.js. *Node.js v18.x Documentation*. Disponible en: https://nodejs.org/en/docs/

[5] MySQL. *MySQL Reference Manual*. Oracle Corporation. Disponible en: https://dev.mysql.com/doc/

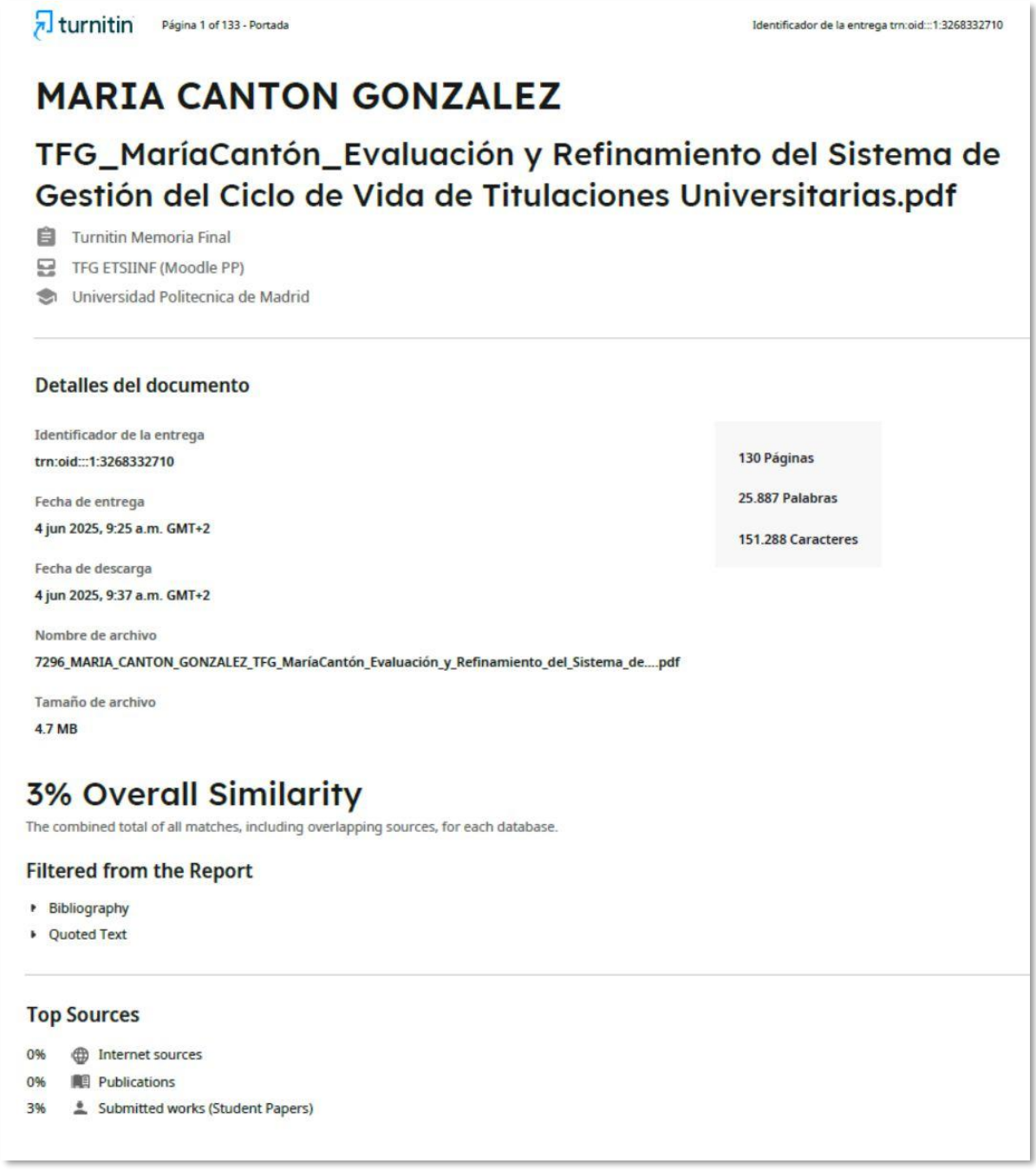
[6] “bcrypt”. npm. Disponible en: https://www.npmjs.com/package/bcrypt

[7] “Multer”. npm. Disponible en: https://www.npmjs.com/package/multer

[8] “Objetivos de Desarrollo Sostenible”. UN. Disponible: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

8 Anexos

8.1 Informe de Originalidad



turnitin Página 1 of 133 - Portada Identificador de la entrega trn:oid:::1:3268332710

MARIA CANTON GONZALEZ

TFG_MaríaCantón_Evaluación y Refinamiento del Sistema de Gestión del Ciclo de Vida de Titulaciones Universitarias.pdf

Turnitin Memoria Final
TFG ETSIINF (Moodle PP)
Universidad Politecnica de Madrid

Detalles del documento

Identificador de la entrega
trn:oid:::1:3268332710

Fecha de entrega
4 jun 2025, 9:25 a.m. GMT+2

Fecha de descarga
4 jun 2025, 9:37 a.m. GMT+2

Nombre de archivo
7296_MARIA_CANTON_GONZALEZ_TFG_MaríaCantón_Evaluación_y_Refinamiento_del_Sistema_de....pdf

Tamaño de archivo
4.7 MB

130 Páginas
25.887 Palabras
151.288 Caracteres

3% Overall Similarity
The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report
- Bibliography
- Quoted Text

Top Sources
0% Internet sources
0% Publications
3% Submitted works (Student Papers)

Ilustración 110. Informe Originalidad Turnitin

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 15:40:24 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)