ULPGC Universidad de Las Palmas de Gran Canaria | eite ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO DE FIN DE GRADO

Plataforma web para Implementación del procedimiento administrativo de Trabajos de Fin de Titulo de la Escuela de Ingeniería de Telecomunicaciones y Electrónica de la ULPGC

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Telemática

Autor: Freddy Alexander López Gutiérrez

Tutor: Dr. Luis Hernández Acosta

Fecha: **JULIO 2025**

Resumen

En un contexto de creciente transformación digital en la educación superior, la gestión actual de los Trabajos de Fin de Título (TFT) en la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la ULPGC se realiza mediante procesos fragmentados, manuales y con herramientas no integradas. Esto genera ineficiencia operativa, falta de transparencia y dificulta la toma de decisiones estratégicas.

Para resolver esta problemática, el presente Trabajo de Fin de Grado presenta la *Plataforma web para Implementación del procedimiento administrativo* de Trabajos de Fin de Título de la EITE de la ULPGC. Su objetivo general es proporcionar una solución integral y centralizada para la gestión automatizada del ciclo de vida completo de los TFT, desde la propuesta hasta la evaluación final.

La plataforma se basa en una arquitectura de microservicios distribuidos, utilizando Laravel 11 (PHP 8.2+) para el *backend* y su API RESTful, con **Laravel Sanctum** para autenticación JWT y **Laravel Reverb/Pusher-php-server** para comunicación en tiempo real del chat. El *frontend* está desarrollado con Angular 17 (Typescript 5.7.2), ofreciendo una interfaz de usuario moderna y responsiva. La base de datos es MySQL. El sistema soporta múltiples perfiles de usuario (Administrador Root, Director EITE, Estudiante, etc.) con roles y permisos específicos, e incluye un sistema de *testing* completo (PHPUnit y Jasmine/Karma) para garantizar la calidad y robustez del código.

Como resultado, la plataforma automatiza eficientemente la gestión de TFT, logrando una mejora significativa en la eficiencia operativa y una reducción de errores administrativos. Esto establece una base sólida para futuras expansiones y asegura una experiencia estandarizada, transparente y de máxima calidad para todos los estudiantes de la EITE.

Abstract

In a context of increasing digital transformation in higher education, the current management of Final Degree Projects (TFT) at the School of Telecommunication and Electronic Engineering (EITE) of the ULPGC is carried out through fragmented, manual processes and with non-integrated tools. This generates operational inefficiency, a lack of transparency, and hinders strategic decision-making.

To solve this problem, this Final Degree Project introduces the Web Platform for the Implementation of the administrative procedure for Final Degree Projects at the EITE of the ULPGC. Its general objective is to provide a comprehensive and centralized solution for the automated management of the complete life cycle of the TFTs, from the proposal to the final evaluation.

The platform is based on a distributed microservices architecture, using Laravel 11 (PHP 8.2+) for the backend and its RESTful API, with Laravel Sanctum for JWT authentication and Laravel Reverb/Pusher-php-server for real-time chat communication. The frontend is developed with Angular 17 (TypeScript 5.7.2), offering a modern and responsive user interface. The database is MySQL. The system supports multiple user profiles (Root Administrator, EITE Director, Student, etc.) with specific roles and permissions, and includes a complete testing system (PHPUnit and Jasmine/Karma) to ensure the quality and robustness of the code.

As a result, the platform efficiently automates the management of TFTs, achieving a significant improvement in operational efficiency and a reduction in administrative errors. This establishes a solid foundation for future expansions and ensures a standardized, transparent, and high-quality experience for all EITE students.

Agradecimientos

Quisiera agradecer primero a mi familia por apoyarme en todo momento durante todo el proceso de sacar esta carrera.

Por otro lado, deseo agradecer a los buenos compañeros con los que he compartido la carrera, es un orgullo para mi ver que han podido terminar con éxito el grado.

También a todo el personal de la ULPGC sea cual sea su estamento que me ha mostrado su apoyo ayudándome cuando más lo he necesitado.

Finalmente, y no por ello menos importante a mis amigos cercanos con los que siempre he compartido buenos momento y me han ayudado a hacer más llevadero este proceso.

Fue duro, pero se logró… GRACIAS

Tabla de contenido

Índice de Tablas

Índice de Ilustraciones

1. Introducción y Objetivos

En este primer capítulo abordaremos todo lo referente al estudio previo realizado antes de comenzar el desarrollo de la plataforma web. En esta sección además estableceremos los objetivos que han de ser completados en el Trabajo de Fin de Grado junto a la estructura que este mismo documento tendrá.

1.1. Contexto

La transformación digital actualmente va ganando terreno en todo ámbito a nivel mundial, no estando la educación superior, como la universitaria, exenta del impacto de dicho fenómeno [1]. Actualmente los avances tecnológicos en el sector de la educación están abriendo nuevas oportunidades para el aprendizaje online o en un modelo hibrido que explota las capacidades de los medios digitales de la mano de los tradicionales, aunque no solo se puede usar la transformación digital para la adquisición de conocimientos, sino que también para agilizar la administración y todos los procesos y gestiones que ello conlleva [1].

Debido a esto y la transformación del panorama educativo, cada vez va cobrando más importancia que los centros educativos superiores se adapten rápidamente hacia la transformación digital mejorando la eficiencia de los métodos tradicionales y aprovechado por otro lado los beneficios que ofrece la digitalización [2]. Cabe destacar que en Reino Unido ya se esta lidiando con la adaptación a estas nuevas metodologias pero teniendo muy presente el desafio de adaptarse a la transformación digital sin que ello afecte negativamente a la calidad de enseñanza [2].

Con esto va quedando claro que la transformación digital es un fenomeno global que no solo afecta a la educación sino a todas las areas afectadas por dicho cambio de paradigma abriendo la puerta a la mejora de eficiencia y rendimiento al permitir una gestión mejor de los sistemas simplificando procesos y las interacciones entre los estudiantes y los administrativos [3].

En el ambito de la Universidad española, se ha querido asumir el imperativo estrategico de la mejora digital. Un informe publicado por la fundación CYD en 2023 revela que el 75% de las universidades españolas disponen de una estrategia formal para la transformación digital abarcando a la totalidad de la comunidad universitaria [4]. Además un 81,25% establece objetivos concretos y medibles para guiar su implementación. Esto demuestra que la digitalización se esta consolidando como una politica prioritaria [4] provocado por el fenomeno de la digitalización que ha ido experimentando los servicios administrativos criticos.

Dicho informe ademas detalla que una mayoria de insituciones ha digitalizado en casi su totalidad los servicios como la secretaria, biblioteca y los canales de solicitud de información además de que casi la totalidad de las universidades ofrecen al alumnado

entornos virtuales de aprendizaje (LMS) y licencias de software necesarias para sus estudios siendo todo esto y aun estandar a nivel nacional [4].

Pero nos encontramos con los desafios mencionados por las universidades en su 73,56% que detallan como principal obstaculo el mantenimiento de una inversion economica adecuada y sostenida. Además otro obstaculo es el de impulsar la adaptación cultural y la participlacion activa del PDI y PAS, señalado por un 60,38% de las instituciones [4].

Hemos llegado a un punto de avance que nos hace pensar en si la cuestion no es si se debe digitalizar o no sino en como, con que grado de profundidad y con que proposito estrategico. No estando el foco ya puesto en la digitalización de servicios basicos como la matricula onlinea o el acceso a bibliotecas virtuales sino hacia servicios mas complejos y ambiciosos como la de procedimientos como es el caso de los Trabajos de Fin de Titulo.

Finalmente, en el ambito regional nos encontramos con que la Comunidad Autonoma de Canarias se encuentra en un proceso de modernización yendo de la mano de los fondos *Next Generation EU* donde la Ley 4/2021 establece un nuevo marco normativo destinado a agilizar los procesos administrativos y la gestión de fondos europeos, sentando las bases para una profunda transformación digital en el sector público Canario, incluidas sus universidades. Esta potenciación se encuentra alineada con los marcos estrategicos superiores como la Estrategia de Especialización Inteligente de Canarias (RIS3) [5].

Tanto la ULL como la ULPGC se encuentran en dicho proceso. Por un lado, la Universidad de La Laguna presento su plan de Modernización Administrativa 2024-2029 [6] mientras que la Universidad de Las Palmas de Gran Canaria tambien a traves de su Fundación Universitaria (FULP) participa en el programa Diginnova ofreciendo formación en materia de Transformación digital [7].

1.2. Motivación

La motivación principal y el punto de partida de este Trabajo de Fin de Grado es la constatación de una necesidad operativa y estratégica fundamental que la Escuela de Ingeniería de Telecomunicaciones y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria (ULPGC) tiene. Dicha necesidad es la de una plataforma web centralizada y dedicada a la gestión integral del procedimiento administrativo y académico asociado a los Trabajos de Fin de Título (TFT). Este hito académico, que representa la culminación de los estudios de Grado y Máster, se gestiona a través de un conjunto de procedimientos

fragmentados, en gran medida manuales y dependientes de herramientas no integradas. Un ejemplo de ello puede ser que la documentación como los formularios se descarguen desde un enlace puesto en el Campus Virtual que lleva a un *OneDrive*.

Esta carencia sitúa a la EITE en una posición de notable desfase con respecto a las tendencias consolidadas a nivel nacional y regional. Mientras el 77,78% de las universidades del Sistema Universitario Español ya han digitalizado en gran medida sus servicios de secretaría y gestión académica [4], y mientras la universidad homóloga en la región, la ULL, ha lanzado un plan explícito para construir una *universidad digital* y reducir los tiempos de tramitación [6], la gestión de un proceso tan crucial como el TFT en una Escuela de Ingeniería de vanguardia permanece anclada en metodologías que no aprovechan las eficiencias y capacidades de las tecnologías actuales. Esta brecha entre el estado del arte del sector y la práctica actual en la EITE constituye el problema central que este proyecto busca resolver.

La inexistencia de un sistema de gestión digitalizado e integral para los TFT no es una mera inconveniencia, sino que genera una serie de consecuencias negativas que afectan a todos los actores implicados en el proceso: el personal administrativo, los estudiantes, el profesorado y el equipo directivo de la Escuela.

En primer lugar, para el Personal de Administración y Servicios (PAS), la situación actual se traduce en una notable ineficiencia operativa. Tareas como la recepción y organización de propuestas de TFT, la asignación de tutores, el seguimiento de plazos críticos, la gestión de la entrega de memorias, la conformación de tribunales evaluadores y el registro final de calificaciones se realizan a través de canales dispersos como el correo electrónico, hojas de cálculo compartidas o incluso documentación física. Este enfoque manual no solo consume una cantidad desproporcionada de tiempo y recursos, sino que también es inherentemente propenso a errores humanos, inconsistencias en los datos y pérdida de información. Es contrario a los objetivos de agilizar la administración y aminorar las cargas administrativas que definen la modernización universitaria [4].

En segundo lugar, desde la perspectiva de los estudiantes y del Personal Docente e Investigador (PDI), la principal consecuencia es una falta de transparencia y trazabilidad. Los implicados carecen de un punto único y fiable de información donde puedan consultar en tiempo real el estado de un TFT, los pasos a seguir, la documentación pendiente o los plazos exactos. Esta opacidad genera incertidumbre, ansiedad y dificulta la planificación, afectando negativamente la experiencia del usuario en una etapa crucial de su vida académica. Un sistema digital, como se ha señalado en estudios sobre el tema, permitiría

un seguimiento continuo de la evolución del alumno, proporcionando claridad y seguridad a todas las partes [3].

En tercer lugar, para la gestión y la dirección académica, la ausencia de un sistema centralizado implica una carencia crítica de datos para la toma de decisiones estratégicas. La gestión manual impide la recopilación sistemática y el análisis de datos agregados valiosos, como el número de TFT defendidos por línea temática, el tiempo medio desde la asignación hasta la defensa, la carga de tutorización de cada profesor o las temáticas más demandadas. Esta información es vital para una gestión basada en la evidencia (*Data-Driven Management*), permitiendo optimizar la asignación de recursos, identificar cuellos de botella, equilibrar la carga de trabajo del PDI y, en última instancia, mejorar la calidad de la oferta académica y del propio proceso de TFT.

Finalmente, la gestión manual del proceso entraña un riesgo significativo de inconsistencia en la aplicación de la normativa. Sin un flujo de trabajo digital y estandarizado que guíe a todos los usuarios, aumenta la probabilidad de que la interpretación y aplicación del reglamento de TFT varíe de un caso a otro. Esto puede derivar en agravios comparativos entre estudiantes y generar problemas administrativos complejos de resolver a posteriori. Una plataforma web aseguraría que cada TFT siga un *workflow* predefinido, garantizando la uniformidad, la equidad y el cumplimiento estricto de la normativa vigente en todas las fases del procedimiento.

Por todo ello, la falta de una plataforma de gestión de TFT no debe ser vista como un simple problema de eficiencia. Es en realidad, un obstáculo para la calidad académica y la equidad. Un proceso manual y opaco es susceptible de generar un trato desigual entre los estudiantes y dificulta la implementación de mejoras basadas en la evidencia. Al impedir un análisis riguroso de los datos del proceso, se debilita la capacidad de la Escuela para garantizar una experiencia estandarizada, transparente y de máxima calidad para todos sus estudiantes en la culminación de su formación. Por lo tanto, este proyecto trasciende la conveniencia administrativa para convertirse en una herramienta estratégica indispensable para asegurar la excelencia y la equidad en la fase final de la formación de los futuros ingenieros de la EITE.

1.3. Antecedentes

Para abordar la creación de una plataforma para la EITE, es imprescindible analizar las soluciones existentes en otras instituciones del Sistema Universitario Español. Si nos paramos a investigar el panorama actual revela que numerosas universidades públicas han

desarrollado e implementado soluciones propias, generalmente en forma de aplicaciones web, para gestionar el ciclo de vida de los TFT. Estas plataformas, aunque heterogéneas en su alcance tecnológico y funcional, constituyen el estado del arte más relevante y ofrecen valiosas lecciones sobre las mejores prácticas y los desafíos comunes. La tendencia predominante es la creación de sistemas con acceso diferenciado por roles, que buscan digitalizar, al menos parcialmente, el complejo flujo administrativo de los TFT.

Nos encontramos con los siguientes casos:

- **Universidad de Cádiz (UCA):** Dispone de una Aplicación de Gestión de Trabajos Fin de Títulos. Su funcionalidad principal se centra en actuar como un portal de oferta y demanda, donde el profesorado puede publicar propuestas de TFT y el alumnado puede solicitarlas o proponer las suyas propias para la asignación de un tutor [8]. Podemos ver su pantalla de *login* en la Ilustración 1.

Ilustración 1: Plataforma de gestión de TFT de la Universidad de Cádiz

- **Universidad de Málaga (UMA):** Presenta un ecosistema con una plataforma de gestión del TFG para los procesos académicos internos y, de manera destacada, una plataforma específica denominada *Impulso TFE UMA* destinada a gestionar los trabajos realizados en colaboración con empresas e instituciones externas. Este enfoque dual sugiere una especialización que puede llevar a una fragmentación del proceso global si no está bien integrada [9]. Podemos ver su pantalla de *login* en la Ilustración 2.

Ilustración 2: Plataforma de gestión de TFT de la Universidad de Málaga

- **Universidad de Alcalá (UAH):** Su aplicación de gestión de Trabajos Fin de Grado modela de forma clara y secuencial la fase inicial del proceso. Define un flujo de trabajo que comienza con la solicitud de propuestas a los departamentos, seguida de su publicación, la selección por orden de interés por parte de los estudiantes y la asignación final por parte de la administración [10]. Podemos ver su pantalla de *login* en la Ilustración 3.

Ilustración 3: Plataforma de gestión de TFT de la Universidad de Alcalá

- **Universidad Rey Juan Carlos (URJC):** La plataforma de la URJC destaca por su arquitectura, que se basa en tres roles explícitos y diferenciados: Acceso Estudiantes, Acceso Profesores/as y Acceso Gestores/as. Esta estructura sugiere un sistema de mayor complejidad y alcance, diseñado para abarcar no solo la interacción alumno-tutor, sino también las tareas de supervisión, control y gestión administrativa por parte del personal de la universidad [11]. Podemos ver su pantalla de *login* en la Ilustración 4.

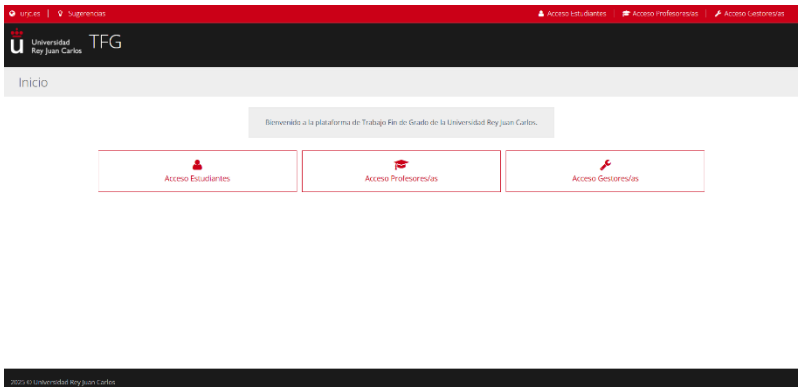

Ilustración 4: Plataforma de gestión de TFT de la Universidad Rey Juan Carlos

- **Universidad Pablo de Olavide (UPO):** Se encuentra actualmente desarrollando su propia *TFG-APP* como parte de su *Plan de Transformación Digital*. El objetivo declarado de este proyecto es lograr la *gestión integral* de los TFT. El uso del

término *integral* es significativo, ya que denota una ambición por cubrir todo el ciclo de vida del proceso dentro de una única herramienta [12]. Podemos ver una página mostrando su proceso en la Ilustración 5.

Ilustración 5: estado del desarrollo de la plataforma de TFT de la Universidad Pablo de Olavide

- **Solución comercial ExDIN:** perteneciente a la empresa DynFile que ofrece una solución de ciclo de vida completo que integra la presentación de trabajos, el registro y la gestión de tribunales de evaluación, la creación de un repositorio institucional y la publicación selectiva de los trabajos con herramientas para la protección de la propiedad intelectual. La existencia de estos productos comerciales no solo valida la necesidad de mercado para herramientas de este tipo, sino que también establece un estándar de calidad y completitud funcional con el que cualquier nuevo desarrollo debe compararse [13]. Podemos ver la pagina promocional de dicha herramienta en la Ilustración 6.

Ilustración 6: Portal web de la empresa DynFile ofertando ExDin

Para visualizar y comparar de manera efectiva las funcionalidades de estas soluciones, se presenta la Tabla 1 que permite identificar patrones, destacar las

características más comunes y señalar las áreas donde muchas soluciones presentan carencias, definiendo así el espacio de oportunidad para un nuevo desarrollo.

	Universidad o Empresa					
Característica	**UCA**	**UMA**	**UAH**	**URJC**	**UPO**	**ExDIN**
Gestión de propuestas	Si	Si	Si	Si	Si	Si
Asignación Tutor-Alumno	Si	Si	Si	Si	Si	Si
Gestión de Entregas	No especificado	Si	No especificado	Si	Si	Si
Flujo de Evaluación/Tribunal	No especificado	No especificado	No especificado	Si	Si	Si
Repositorio Institucional	No especificado	No especificado	No especificado	No especificado	Si	Si
Acceso por Roles (Admin)	No especificado	Si	No especificado	Si	Si	Si
Integración Externa (Empresas)	No especificado	Si	No especificado	No especificado	No especificado	No especificado

Tabla 1: Comparativa de funcionalidades plataformas TFT existentes

Como se puede observar, la mayoría de las plataformas existentes gestionan eficazmente la fase inicial de propuesta y asignación. Sin embargo, funcionalidades más avanzadas como la gestión digital del proceso de evaluación (conformación de tribunales, entrega de actas) y la integración directa con el repositorio institucional son menos comunes o no se publicitan de forma explícita. Casos como el de la URJC usando el rol de **Gestor**, y el de la UMA, con su módulo para empresas, representan los enfoques más especializados y completos entre las soluciones desarrolladas internamente. Siendo eso sí, el más completo hasta ahora la solución comercial ExDIN.

El análisis del estado del arte revela una brecha clave: la de la *integralidad*. Si bien existen múltiples soluciones, muchas de las plataformas desarrolladas por las universidades se enfocan principalmente en la fase inicial del proceso (propuesta y asignación), funcionando más como **Portales de oferta o Gestores de propuestas** que como **sistemas de gestión de ciclo de vida completo**. La verdadera gestión integral es aquella que abarca desde la idea inicial hasta la evaluación final por parte del tribunal y el

depósito automático en el repositorio institucional, todo ello dentro de un único flujo de trabajo digital, trazable y transparente sigue siendo un reto técnico y organizativo que a menudo no está completamente resuelto. La comparación con las funcionalidades de una solución comercial como *ExDin® TFGM* evidencia esta brecha funcional.

Por consiguiente, la complejidad del problema a resolver con este TFG no reside en la creación de una simple página web para la subida de documentos. El desafío radica en el diseño y desarrollo de un sistema de información robusto, escalable y seguro que aborde con éxito una serie de retos complejos:

- **Modelo de datos complejo**: el sistema debe ser capaz de modelar y gestionar las relaciones entre múltiples entidades: **Estudiante**, **Tutor**, **Co-tutor**, **Propuesta de TFT**, **TFT**, **Tribunal, Miembro de Tribunal, Documento (memoria, anexos), Hito de seguimiento, Calificación y Estado del proceso**.

- **Máquina de Estados (Workflow Engine):** es fundamental implementar una lógica que gestione el ciclo de vida de cada TFT, moviéndolo a través de diferentes estados como puede ser **Propuesto**, **Asignado**, **En desarrollo**, **Entregado para defensa**, **Evaluado** y **Archivado** en función de las acciones de los usuarios y del cumplimiento de los plazos establecidos en la normativa.

- **Control de Acceso Basado en Roles (RBAC):** Inspirado en el enfoque de la URJC, es imprescindible un sistema de permisos granular que otorgue capacidades específicas y diferenciadas a cada tipo de usuario: **Estudiante** (solo puede ver y actuar sobre su TFT), **PDI/Tutor** (puede supervisar a sus tutorandos), **Coordinador de TFT** (tiene una visión global de los TFT de una titulación) y **PAS/Administrador** (tiene control total sobre el sistema).

- **Integración con Sistemas Existentes (API):** Para maximizar la eficiencia y evitar la duplicidad de datos, la plataforma ideal debería poder interactuar con otros sistemas corporativos de la ULPGC, como el sistema de gestión académica (para validar la matrícula de los estudiantes y registrar las calificaciones finales) y el repositorio institucional (para el depósito automatizado de los trabajos aprobados).

- **Sistema de notificaciones y comunicación:** La plataforma debe automatizar el envío de notificaciones y recordatorios a todos los implicados como son los recordatorios de plazos de entrega, avisos de asignación de tribunal y notificaciones

de calificación para asegurar la fluidez del proceso y el cumplimiento de la normativa.

- **Usabilidad y Experiencia de Usuario (UX):** La interfaz debe ser intuitiva, clara y fácil de usar para todos los perfiles de usuario, incluidos aquellos con menores competencias digitales, con el fin de minimizar la curva de aprendizaje y fomentar una adopción rápida y generalizada.

En última instancia, el verdadero desafío de este TFG es de naturaleza sociotécnica. No se trata únicamente de un reto de programación, sino de la capacidad para modelar y digitalizar con éxito un proceso administrativo y académico que es inherentemente complejo, que está fuertemente arraigado en la cultura organizativa de la Escuela y que se encuentra regulado por normativas académicas estrictas. El éxito del proyecto se medirá no solo por la robustez y funcionalidad de su código, sino, y más importante aún, por su capacidad para ser adoptado por la comunidad de la EITE y para transformar realmente un proceso crítico, alineando a la Escuela con las mejores prácticas de la universidad digital del siglo XXI.

1.4. Objetivos

Tenemos como base el objetivo general de este Trabajo de Fin de Grado que es **desarrollar una plataforma que ofrezca una solución que permita la gestión centralizada de los TFT de la EITE y autenticación de los usuarios. Entonces la finalidad de este trabajo es proporcionar la implementación del procedimiento administrativo completo que un alumno, tutor o miembro de la comisión de TFT ha de realizar desde el inicio del curso hasta que se presenta y se defienden sus TFT.**

Los objetivos específicos que se pretenden alcanzar con el desarrollo de este proyecto con el fin de cumplir el objetivo general son los siguientes:

- **Objetivo 1: Analizar los requisitos funcionales y casos de uso para la plataforma.**
  Trataremos de analizar todos los posibles casos de uso con el fin de establecer unos requisitos acordes a estos que provoquen un diseño lo suficientemente cómodo para el usuario.

- **Objetivo 2: Diseñar la arquitectura del sistema que contenga el *frontend*, *backend*, base de datos y los tests.**

Una vez realizada la definición de los requisitos que debe tener nuestra plataforma a raíz del análisis inicial de los casos de uso procederemos a realizar el diseño de la arquitectura que ha de tener nuestra plataforma definiendo concretamente la arquitectura y estructura del *frontend*, *backend*, base de datos y tests. Teniendo en cuenta en todo momento al reglamento de TFT de la EITE presente.

- **Objetivo 3: Llevar a cabo la creación del *backend* usando PHP. Con el fin de desarrollar toda la lógica de la aplicación que será implementada totalmente desde el *backend* por motivos de seguridad. Además, se creará una API para que el *frontend* pueda solicitar y mostrar los datos. Para controlar el acceso de los usuarios, implementaremos el sistema de autenticación en el *backend* (la parte de la aplicación que funciona en el servidor) aprovechando las soluciones de seguridad que ya vienen incluidas en Laravel.**
  Una vez pasada la fase de diseño y teniendo clara la arquitectura y estructura que nuestra plataforma de gestión de TFT ha de tener se tratara de realizar toda la implementación en cuanto a *backend* se refiere para dejar el grueso de funcionalidades ya implementadas.

- **Objetivo 4: Desarrollar un *frontend* responsive con Angular para ofrecer una interfaz de usuario intuitiva y eficiente que permita a los distintos usuarios de la plataforma realizar sus gestiones.**
  Una vez implementado el *backend*, continuaremos con el desarrollo del *frontend*, que es la parte visual con la que los usuarios interactuarán en nuestra plataforma. Se buscará que la experiencia de usuario sea fluida e intuitiva en todo momento. Para ello, se seguirá el principio de diseño de minimizar el número de *clicks* necesarios para completar cualquier tarea, evitando así que el producto resulte tedioso de usar.

- **Objetivo 5: Verificar el correcto funcionamiento del código mediante la ejecución de los tests desarrollados para tal fin.**
  Con el fin de comprobar que nuestro código ya implementado tenga la calidad y robustez que definimos en la etapa de diseño y para además verificar que nuestras implementaciones siguen las directrices del reglamento de TFT de la EITE, nuestro código debe pasar los tests a los cuales será sometido y hemos definido en la etapa de diseño.

1.5. Contenidos

El proyecto se organiza en 7 capítulos principales y adicionalmente la bibliografía, el presupuesto y el anexo. Se describirá a continuación de forma breve cada uno de los capítulos y apartados de los que se compone esta memoria:

- **Capítulo 1: Introducción.** Con carácter introductorio en donde presentamos el estudio previo realizado antes de comenzar con el desarrollo de la plataforma. Además, establecemos los objetivos iniciales del proyecto y finalmente se explica la estructura de la memoria.

- **Capítulo 2: Tecnologías software.** Se realizará una descripción de las tecnologías software y librerías que se han usado para realizar este proyecto.

- **Capítulo 3**: Descripción general. Se abordará a rasgos generales el esquema del planteamiento realizado para la aplicación usando para ellos esquemáticos y diagramas de bloques.

- **Capítulo 4: Diseño.** Se presentará el diseño realizado previo a la elaboración e implementación del proyecto con los requisitos que la plataforma debe cumplir, el modelo de las pantallas que será de guía (mockups) y la estructura de la base de datos.

- **Capítulo 5. Implementación.** Aquí se abordará en detalle las consideraciones de la programación para poder realizar nuestro proyecto incluido las referidas a los tests.

- **Capítulo 6: Guía de usuario.** Se presentará una guía para el uso de la plataforma pasando por el flujo que sigue esta de principio a final a modo ilustrativo y explicativo además de describir para el usuario las características y funcionalidades que ofrece la plataforma.

- **Capítulo 7: Conclusiones.** En este capítulo principal final se abordarán todas las conclusiones alcanzadas en este Trabajo de Fin de Grado tras el desarrollo del software. Contiene los objetivos cumplidos y líneas futuras donde se proponen mejoras y características que se podrían añadir posteriormente.

- **Bibliografía.** Recopilatorio de referencias, webs y documentos que han sido consultados durante la elaboración de la memoria de este Trabajo de Fin de Grado.

- **Presupuesto.** Es una estimación del coste aproximado de la realización de este Trabajo de Fin de Grado.

- **ANEXO.** Se encontrará aquí los *endpoints* usados en este software que por razones de comodidad al lector se decidió poner en este lugar. Además, nos encontraremos aquí con un enlace al repositorio **Github** donde se encuentra alojado el código de este proyecto. Finalmente en el anexo se encontrarán los detalles de las implementaciones de los tests.

2. Tecnologías software

En este capítulo se describen las tecnologías y librerías utilizadas en el desarrollo de este proyecto.

2.1. PHP (versión 8.2 o superior)

Ilustración 7: PHP

La aplicación en el *backend* requiere de PHP en su versión 8.2 o superior. Siendo PHP un lenguaje de programación usado principalmente del lado del servidor y que ofrece la posibilidad de tener propiedades tipadas en clases, estructuras de control mejoradas y más funcionalidades que la hacen ideal para ser usada en el *backend* cuando se realiza una aplicación web [14]. Su logo lo podemos ver en la Ilustración 7.

Lo usaremos para desarrollar el *backend* API REST y definir toda su la lógica de negocio, gestionar las peticiones del cliente, procesar datos y comunicarnos con la base de datos. Todo ello usando Laravel.

2.2. Typescript 5.7.2

Ilustración 8: Typescript

Typescript es el lenguaje de programación usado para el *frontend*. Aportando tipado estático frente a Javascript. Tiene características como interfaces que da la posibilidad de definir los contratos, tipos genéricos, decoradores para Angular y tipos de unión [15]. Podemos ver su logo en la Ilustración 8.

Se ha elegido este lenguaje de programación debido a que para la librería angular para el *frontend* elegida para realizar este proyecto es la que está basada y teniendo soporte.

2.3. SQL

Ilustración 9: SQL

Se trata de un lenguaje de consulta estándar usado con el fin de interactuar con las bases de datos relaciones siendo muy importante para operaciones CRUD (crear, leer, actualizar y borrar) y definir esquemas de datos y ejecutar consultas complejas [16]. Podemos ver su logo en la Ilustración 9.

Lo hemos usado para interactuar con nuestra base de datos que es del tipo MySQL y definir la estructura de nuestras tablas y sus correspondientes relaciones.

2.4. MySQL

Ilustración 10: MySQL

Se trata de una herramienta de gestión de base de datos relacionales de código abierto que proporciona un entorno robusto y confiable para el almacenamiento, gestión y recuperación de datos. Ofrece un motor de base de datos optimizado que garantiza la integridad y consistencia de los datos incluyendo funcionalidades como índices optimizados, consultas complejas y procesamientos almacenados [17]. Podemos ver su logo en la Ilustración 10.

La usaremos como sistema de gestión de la base de datos de nuestra plataforma diseñando un esquema relacional que moda todas las entidades del sistema académico.

2.5. Laravel (versión 11.31)

Ilustración 11: Laravel

Se trata de un *framework* de desarrollo para PHP que sigue el patrón MVC (Modelo-Vista-Controlador) que proporciona una estructura organizada, funcionalidades y herramientas que ayudarán a construir el *backend* de manera ágil, legible y ordenada [18]. Podemos ver su logo en la Ilustración 11.

Como se dijo antes, usa la arquitectura MVC para separar la lógica de negocio, los datos y la presentación. Además, ayuda bastante a simplificar las interacciones con la base de datos debido a su ORM Eloquent y su sistema *Middleware* es usado para filtrar peticiones HTTP para por ejemplo autenticación además de una herramienta de línea de comandos como Artisan que puede automatizar tareas repetitivas [18].

Hemos usado este *framework* para construir todo nuestro *backend* siendo la API REST. Nos ha permitido la gestión de rutas en casos particulares, control de acceso a usuarios como autenticación y autorización además de definir la lógica de negocio de forma ordenada y estructurada.

2.6. Angular

Ilustración 12: Angular

Se trata de un *framework* de desarrollo para el *frontend* basado en Typescript y siendo mantenido por la empresa Google con el fin de construir *Single Page Applications* (SPA). Nos ofrece una arquitectura basada en componentes que es robusta, inyección de dependencias, sistema avanzado de enrutamiento y gestión de formularios reactivos entre

otras funciones muy útiles a la hora de desarrollar el *frontend* de una aplicación web [19]. Podemos ver su logo en la Ilustración 12.

Para desarrollar nuestra plataforma de TFT de la EITE lo usaremos para el desarrollo de toda la interfaz de usuario. Hemos aprovechado la arquitectura del *framework* y hemos usado componentes reutilizables gestionando la navegación entre vistas y manejado la interacción del usuario de forma dinámica.

2.7. Visual Studio Code

Ilustración 13: Visual Studio Code

Se trata de un editor de código ligero y sencillo de usar desarrollado por Microsoft que se puede usar en diversos sistemas operativos como Windows, macOS y Linux. Aparte de permitirnos editar código nos permite añadirle extensiones de diversas utilidades además de Git incorporado [20]. Podemos ver su logo en la ilustración 13.

Se ha usado en este proyecto para escribir, depurar y gestionar todo el código tanto el *backend* como el *frontend*.

2.8. Github

Ilustración 14: Github

Se trata de una plataforma de desarrollo colaborativo alojada en web usada con el fin de alojar proyectos usando el sistema de control de versiones denominado Git. Sirve como repositorio central remoto o simplemente como una nube en donde podremos subir

nuestro código para que este a buen recaudo y descargarlo cuando sea requerido. Es ideal para entornos donde la colaboración es fundamental donde se vaya a revisar el código normalmente y se gestionen proyectos [21]. Permite realizar el seguimiento de proyectos por varios usuarios.

2.9. Figma

Ilustración 15: Figma

Figma es una herramienta que nos permite trabajar en la nube que nos permite crear interfaces de usuario, prototipos interactivos y sistemas de diseño de manera eficiente y colaborativa si fuera el caso [22]. Esta herramienta nos proporciona un conjunto de herramientas de diseño que nos ayudara a crear interfaces de usuario complejas y detalladas. Podemos ver su logo en la Ilustración 15.

La hemos usado en nuestra plataforma para la etapa de diseño, concretamente para la del interfaz de usuario. Creamos los mockups usando Figma.

2.10. Postman

Ilustración 16: Postman

Postman es una plataforma de desarrollo de APIs que permite a los desarrolladores diseñar, construir, probar e iterar APIs de manera eficiente. Es esencialmente una herramienta que facilita el trabajo con APIs RESTful y otros tipos de servicios web [23]. En la Ilustración 16 podemos ver su logo.

Postman ofrece un conjunto completo de funcionalidades que lo convierten en una solución integral para el trabajo con APIs. Como cliente HTTP avanzado, permite enviar todo tipo de peticiones HTTP incluyendo **GET, POST, PUT, DELETE** y otros métodos, con la capacidad de personalizar *headers*, parámetros y cuerpos de petición [23].

En nuestra aplicación la usamos para comprobar que el *backend* atiende correctamente todas las solicitudes que le hacemos desde postman.

2.11. Librerías usadas

2.11.1. Laravel Sanctum

Se trata de un paquete oficial de Laravel con el fin de ser usado para la autenticación de APIs y SPA ofreciendo un sistema ligero y sencillo para emitir y validar *tokens* de API permitiendo que se protejan las rutas del *backend* [24].

Lo hemos usado en nuestro proyecto con el fin de proteger nuestra API REST en donde nuestro *frontend* obtiene un token al iniciar sesión usándolo para cada petición para poder demostrar que el usuario se encuentra autenticado. El uso de esta librería añade autenticación y seguridad.

2.11.2. Laravel reverb

Se trata de un servidor *WebSocket* oficial de Laravel con características de alta velocidad y escalable que permite una integración nativa en nuestro código Laravel [25].

Lo usamos en nuestra plataforma para ser el motor principal de la comunicación entre usuarios en tiempo real gestionando las conexiones *WebSocket* de los clientes para la funcionalidad del chat.

2.11.3. Pusher-php-server

Se trata de un SDK oficial de **pusher** para PHP. Ofrece una API sencilla para que el código escrito en PHP del servidor pueda enviar mensajes a través de los canales del *WebSocket* [26].

Se ha usado como la interconexión entre nuestro *backend* de Laravel pueda enviar elementos en tiempo real al servidor **Reverb** que los distribuye a los clientes asociados para nuestro chat.

2.11.4. PhpUnit

Se trata de una librería para realizar pruebas unitarias y de integración en el código escrito en PHP. Permite obtener una serie de herramientas útiles para escribir, ejecutar y analizar los resultados de las pruebas automatizadas de los tests [27].

Se ha usado esta librería en el proyecto para crear los tests que validarán el correcto funcionamiento de nuestras clases, métodos y *endpoints* de nuestra API asegurándonos que el código cumple con todos los requisitos establecidos.

2.11.5. Ngx-toastr

Es una librería de notificaciones emergentes como las *toasts* para Angular ofreciendo un sistema que nos permite personalizar las notificaciones evitando que sean intrusivas para dar retroalimentación al usuario [28].

La usamos para mostrar mensajes de éxito, error o informativos tras haber realizado el usuario una acción como el por ejemplo crear un nuevo usuario por parte del **administrador root.**

2.11.6. @angular/cli y @angular-devkit/build-angular

El primero es una CLI de Angular y la segunda las herramientas de construcción y desarrollo asociadas a esta. Nos da la posibilidad de usar varios comandos para poder generar los componentes, servicios y otras utilidades que Angular nos ofrece además de permitirnos compilar el código del *frontend*, probarlo y desplegar en modo de prueba nuestra aplicación [29].

En el desarrollo de nuestra plataforma de TFT de la EITE es una herramienta fundamental, siendo la principal para todo el ciclo de desarrollo del *frontend* ayudándonos a agilizar la creación y gestión del proyecto.

2.11.7. Karma y Jasmine-core

Se trata de un entorno de pruebas por defecto para Angular. Siendo Jasmine una librería donde se escriben pruebas y Karma el ejecutor, ósea, el test *runner* que las lanza a un navegador [30] [31]. Esto nos permite ejecutar pruebas unitarias del código realizado en el *frontend* de forma automática obteniendo los resultados de manera detallada y entendible.

En nuestra plataforma las hemos usado para escribir pruebas unitarias para nuestros componentes y servicios de Angular asegurándonos que cada pieza de la interfaz de usuario funciona de forma correcta de manera aislada.

3. Descripción general

En este capítulo abordaremos el planteamiento de nuestra plataforma de gestión de TFT de la EITE de una manera general usando para ello esquemas y diagramas con el fin de añadir claridad y que se exponga el funcionamiento de la mejor manera posible.

3.1. Funcionamiento

La plataforma constituye un sistema integral de gestión académica orientada a automatizar, centralizar y digitalizar los procesos implicados en los TFT de la EITE de la ULPGC.

Para la realización de esta plataforma nos hemos acogido al Reglamento de la EITE para la realización de Trabajos de Fin de Titulo aprobado en Consejo de Gobierno de la ULPGC siendo dicho reglamento el que rige el funcionamiento del sistema académico de gestión de TFT implementado en esta Trabajo de Fin de Grado.

Dicho sistema implementa, de la mano del reglamento de TFT de la EITE, una solución tecnológica que abarca desde la propuesta inicial de anteproyectos de TFT hasta su defensa y evaluación final.

Para desarrollar este sistema hemos diseñado 7 perfiles de usuario que son: **Administrador Root, director EITE, secretario EITE, Comisión TFT, Tutor Académico, Estudiante y Tribunal Evaluador.**

3.1.1. Inicio de sesión

El sistema de inicio de sesión para esta plataforma usa una arquitectura de autenticación basada en *tokens* JWT mediante **Laravel Sanctum** en el *backend* y **Angular Guards** en el *frontend* garantizando la seguridad, escalabilidad y ofreciendo una experiencia de usuario fluida. Este proceso involucra la colaboración de varios componentes de manera coordinada entre el *frontend* de **Angular** con el *backend* de **Laravel**.

El proceso de *login* comienza con un usuario accediendo a la plataforma donde en primer lugar el sistema verificará si hay guardado en el almacenamiento local del navegador un *token* valido. Si no hay *token*, el usuario será redirigido automáticamente a la pantalla de *login*. En caso de que, si hubiera un *token* valido, el usuario accederá automáticamente a la pantalla de **dashboard**.

Para iniciar sesión, el usuario deberá introducir dos credenciales: su DNI (sin la letra final) y su contraseña personal. El *frontend* valida que ambos campos estén completos con validación mínima de DNI con sus 8 caracteres además de deshabilitar el botón de inicio de sesión si hay una petición en curso y mostrando un indicador de carga.

Una vez enviada la petición al *backend* el servidor valida que se encuentren ambos campos y sintetiza los datos de entrada para evitar ataques de inyección. Busca en la base de datos al DNI introducido, si no lo encuentra devolverá el error de **Credenciales incorrectas**. Luego verificara la contraseña comparando la contraseña proporcionada por el usuario con el *hash* de esta guardado en la base de datos usando el algoritmo **BCrypt** para ello. En caso de que no sean iguales devolverá el error de **Credenciales incorrectas**. Tras esto verifica que el usuario se encuentra activo, de lo contrario, devolverá el error de **Usuario inactivo**. Si todas las verificaciones en el *backend* son exitosas, se devolverá un *token* JWT único. Dicho token se guardará en la base de datos vinculado al usuario y, a la vez, se actualizará la fecha y hora de su último acceso.

Tras esto el *frontend* guardara dicho token en el **localStorage** del navegador actualizando de manera global el estado de autenticación de la aplicación y redirigiendo automáticamente al **dashboard** según el rol del usuario. En la figura 17 podremos ver un diagrama de bloques que ilustra la ejecución del inicio de sesión de la plataforma.

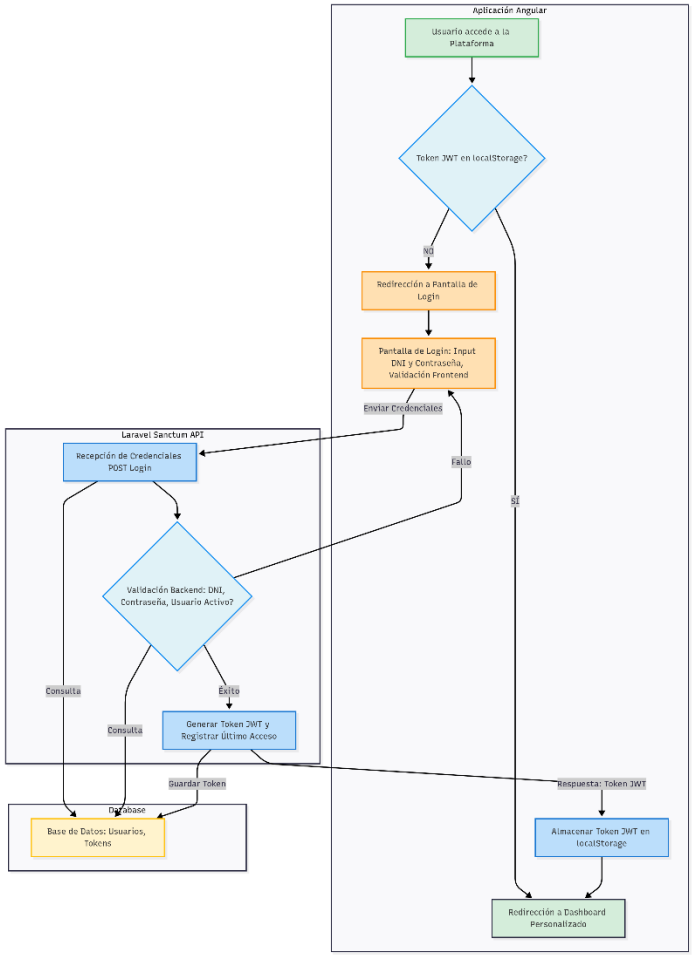

Ilustración 17: Diagrama de funcionamiento del inicio de sesión

3.1.2. Perfil del administrador root

El **administrador root** es el rol con máximo nivel de autoridad dentro de la plataforma teniendo acceso total al sistema siendo el responsable de la gestión completa de los usuarios, configuración del sistema y supervisión general de las operaciones. Dentro de la jerarquía de roles del sistema es quien tiene máximo control pudiendo acceder a todos los módulos y funcionalidades de manera ilimitada.

Puede acceder al **dashboard** y por otro lado gestionar usuarios teniendo control total aquí pudiendo crear, eliminar, modificar o desactivar a los usuarios. Además, el usuario tiene gestión total sobre los TFTs. También puede administrar los tribunales, lo que incluye crearlos desde cero y modificar su composición al añadir, eliminar o cambiar sus miembros.. Por otro lado, el **administrador root** puede realizar la gestión completa y sin restricciones del calendario realizando la gestión de convocatorias académicas además de poder configurar completamente los parámetros de la plataforma. Finalmente, tiene acceso a todos los documentos relacionados con los procedimientos académicos del TFT de la EITE. Ahora procederemos a mencionar con un poco más de detalle las funcionalidades que tiene este rol de usuario.

En cuanto a la gestión de usuarios este tipo de rol puede ver todos los usuarios del sistema con filtros para ayudar a identificarlos según la tipología de usuario que sea o por nombre, puede editarlos modificando datos de cualquier usuario, eliminarlos menos a sí mismo, activarlos o desactivarlos, cambiar contraseñas y los filtros pueden ser por rol, departamento, nombre…

En cuanto a los TFTs, el usuario puede visualizar todos los que se encuentran en el sistema, modificar cualquiera de ellos, gestionar sus estados y flujos de trabajo, acceder a toda la documentación y supervisar las evaluaciones y calificaciones.

En el caso de los tribunales el **administrador root** puede crear nuevos, asignar miembros a su respectivo cargo como es el presidente, secretario y vocal además de su correspondiente suplente de cada tipo y puede supervisar las calificaciones.

Y finalmente en cuanto a las convocatorias este usuario puede crearlas, configurar las fechas y plazos académicos y gestionarlos.

En la figura 18 podemos ver un esquema que ilustra las funciones del **administrador root.**

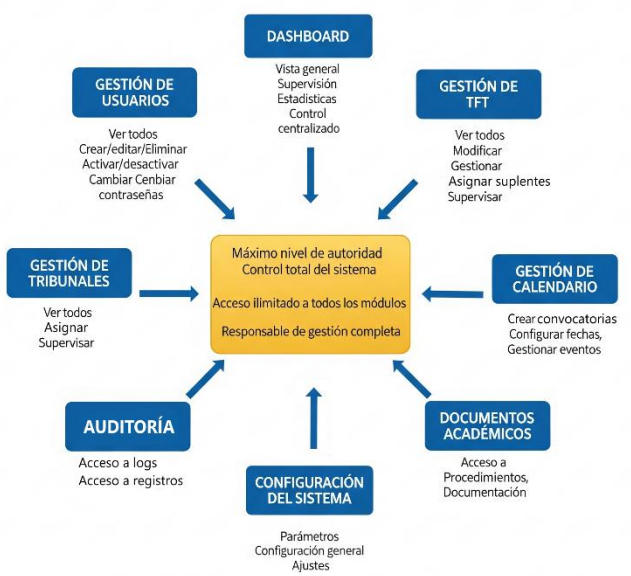

Ilustración 18: Esquema de administrador root

3.1.3. Perfil de director de la EITE

El **director de la EITE** es a nivel reglamentario de TFT de la EITE quien tiene mayor jerarquía según dicho reglamento actuando como presidente de la **comisión de TFT**. Entre las funciones reglamentarias de este usuario nos encontramos que puede presidir las **comisiones de TFT**, puede delegar funciones en subdirectores, participar en la aprobación de propuestas de TFT y en la designación de tribunales evaluadores además de gestionar administrativamente al centro.

En nuestra plataforma de gestión de TFT de la EITE eso se ve reflejado en que dicho usuario puede acceder a la gestión completa de los TFT donde por ejemplo se encuentra la función de rechazar o aceptar anteproyectos, gestionar a los tribunales evaluadores donde puede crearlos, añadir sus componentes, modificarlos o eliminarlo al igual que el **administrador root**. Además, puede participar en la gestión de eventos como crear eventos académicos y establecer fechas y plazos de entregas. Por otro lado, puede acceder a la lista de usuarios, pero a diferencia del **administrador root**, solo tiene permisos de lectura.

Otra de sus capacidades es el acceso a todos los formularios para su verificación. Esto le permite consultar tanto los documentos restringidos a los **tutores** y la **comisión**, como aquellos pertenecientes a los estudiantes.

El flujo de trabajo para este usuario comienza cuando un **estudiante** entrega su propuesta de anteproyecto. En ese momento, el **director de la EITE**, como miembro de la

comisión, recibe la documentación. A continuación, la examina junto al resto de miembros para decidir si cumple con los requisitos del reglamento y, en consecuencia, aprobarla o rechazarla. En caso de ser rechazada se deberá dar un motivo de rechazo que el **alumno** podrá ver y establecer un plazo de 10 días hábiles para la subsanación abriendo una nueva entrega para ello.

En caso de ser aprobado, el TFT pasa automáticamente al estado **EN DESARROLLO**. A partir de ese momento, el director se encarga de supervisar que se cumplan los requisitos que establece el reglamento de la institución. Tras la entrega de la documentación del TFT por parte del **alumno**, el director de la EITE en calidad de **Presidente de la comisión de TFT de la escuela** procederá junto a los demás miembros de la comisión a asignar un tribunal y sus correspondientes miembros al TFT. En la figura 19 podemos ver el diagrama de bloques que ilustra el funcionamiento hasta que se aprueba un anteproyecto donde se encuentra involucrado el **director de la EITE.**

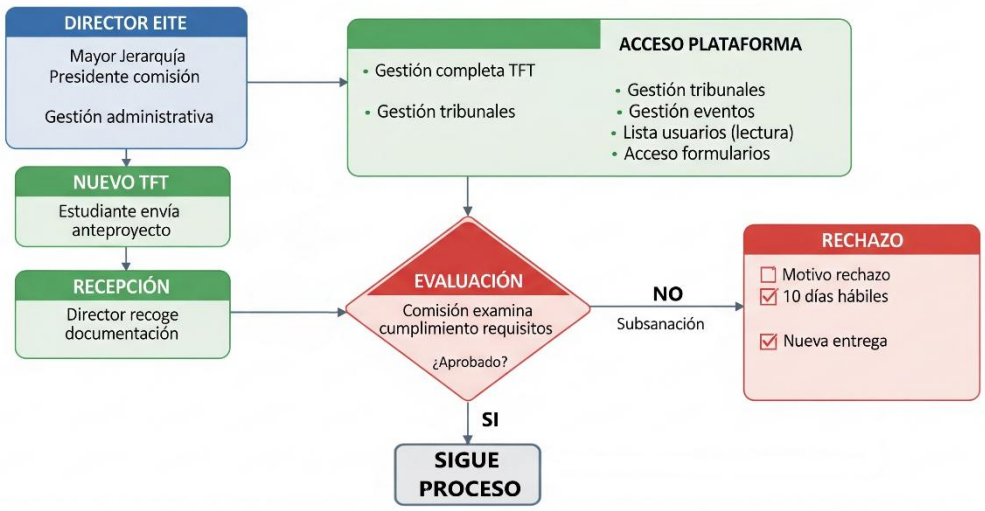

Ilustración 19: Diagrama de funcionamiento del director de la EITE

3.1.4. Perfil secretario de la EITE

Dentro del escalafón de nuestra plataforma, en el tercer peldaño nos encontramos con el **secretario/a de la EITE** que actúa como lazo entre la **Dirección Académica** y los **Procesos Administrativos** manteniendo la trazabilidad y control documental durante todo el proceso.

Entre las funciones del **secretario de la EITE** nos encontramos con que levanta las actas de las reuniones de la comisión de TFT, la gestión documental de los TFTs, la comunicación oficial entre **estudiantes** y **tribunales** que incluye la emisión de documentos

y comunicados oficiales. Finalmente, se encarga del custodio y mantenimiento del repositorio de TFT finalizados.

Puede acceder a la documentación completa al igual que el **director de la EITE** de todos los usuarios para verificación dejando para implementaciones futuras el añadir la gestión completa documental que implica control de versiones de archivos y subir los correspondientes archivos según su tipología y actor al que se dirige en el proceso administrativo de la gestión de TFT.

Al igual que el **director de la EITE** puede acceder a sus mismas pantallas, pero totalmente limitado a la lectura. En la Ilustración 20 podemos ver un esquema de las funciones del **secretario de la EITE**.

Ilustración 20: Esquema del secretario EITE

3.1.5. Perfil miembro comisión TFT

El rol principal de un **miembro de la comisión de TFT** es bastante parecido al del **director y secretario de la EITE** pero con mayores limitaciones al no tener la carga directiva ni administrativa de estos. Por lo que la funcionalidad principal de este perfil es la de visualizar y aprobar propuestas de TFT.

Las decisiones de la **comisión de TFT** se toman de forma colectiva durante sus reuniones. En ellas se tratan tanto la aprobación o el rechazo de propuestas como la deliberación sobre casos puntuales que surjan durante el ciclo de vida de un TFT, desde el anteproyecto hasta su defensa final y evaluación.

Dentro del su rol principal que es el de aprobar o rechazar las propuestas de TFT el flujo es el mismo que el del **director de la EITE** en este caso. Llega la documentación del anteproyecto del TFT pudiendo ser visualizada por todos los miembros de la comisión

de TFT incluido el **director de la EITE** y el **secretario** de la **Escuela**. Tras un análisis y revisión de las propuestas, se hará una reunión de toda la **comisión** para emitir un veredicto de aprobación o rechazo de propuestas dando un motivo de rechazo. Una vez hecho, será el **director de la EITE** quien a través de esta plataforma indicará si ha sido rechazada dando un motivo de rechazo o aprobación. En ese caso, se abrirá un nuevo plazo de subsanación de 10 días hábiles para que el **estudiante** entregue las correcciones. Una vez recibidas, los **miembros de la comisión** leerán los documentos y se reunirán de nuevo para valorar los cambios.

Entonces el perfil de **miembro de comisión de TFT** está más orientado a solo lectura de documentación ya que el **Director de la EITE** será quien cambie el estado de aprobado o rechazado en la plataforma, aunque el miembro de la comisión tiene la posibilidad de realizar la aprobación o rechazo en la plataforma. En la figura 21 podemos ver un esquema de las funciones del **miembro de la comisión de TFT**.

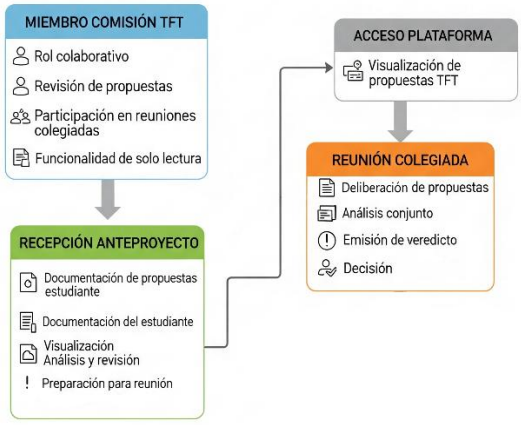

Ilustración 21: Esquema de miembro de la comisión de TFT

3.1.6 Perfil de Tutor académico

El **tutor académico** es fundamental dentro de la realización de un Trabajo de Fin de Titulo siendo su rol principal el de guía académico del **estudiante** durante la realización de su TFT. Siendo el tutor el enlace directo entre la institución académica y el **estudiante**. Además, dicho usuario será quien realizará el seguimiento del trabajo del estudiante durante la realización de su TFT. Por ello en nuestra plataforma, el tutor solo a modo de lectura podrá entrar al apartado del TFT correspondiente y ver el estado del trabajo viendo las fechas de entrega.

3.1.7. Perfil de Tribunal evaluador

Los **miembros del tribunal evaluador** tienen la función de verificar los requisitos académicos de defensa, levantar el acta de defensa y evaluar y calificar el TFT revisando la documentación del TFT entregada por el alumno mediante la plataforma de carga de documentación que ha sido habilitada en nuestra plataforma. El tribunal asiste a la defensa del TFT y, posteriormente, se encarga de registrar la calificación final en la plataforma. Como son 3 miembros, cada uno tras la defensa del TFT accede a la pantalla habilitada solo para ellos del TFT de evaluación en donde cada uno por separado pone la nota y el sistema una vez todos hayan puesto su nota calculara automáticamente la media de esas 3 notas y se quedara como calificación final mostrándoselo al estudiante y cambiando el estado del TFT a **EVALUADO**.

El tribunal estará conformado por tres miembros principales más sus respectivos suplentes.

- **Presidente del tribunal**: quien preside la sesión de evaluación, coordina la deliberación y garantiza el cumplimiento de los reglamentos.
- **Secretario**: es el encargado de levantar el acta oficial de la defensa, registrar las calificaciones y documentar las observaciones del tribunal.
- **Vocal**: participa en la evaluación y deliberación.
- **Suplentes de cada puesto**: en caso de ausencia justificada, los suplentes sustituirán a cada miembro ausentado permitiendo que se cumpla la composición reglamentara. Hay que tener en cuenta que existe el suplente de presidencia, suplente de secretario y el suplente del vocal.

En nuestra plataforma pueden acceder solamente a los TFT asignados a su correspondiente tribunal a modo de lectura para analizar y evaluar la documentación que el **alumno** ha enviado a su TFT. Tras eso puede dentro de la misma pantalla de TFT programar la defensa del TFT que una vez hecha el estudiante podrá verla y tras eso podrá cada miembro poner su calificación. También podrá acceder a la pantalla de **formularios** solo pudiendo visualizar la documentación que le compete al tribunal como es la rúbrica de evaluación. En la Ilustración 22 podemos ver el esquema de funcionamiento del **Tribunal Evaluador.**

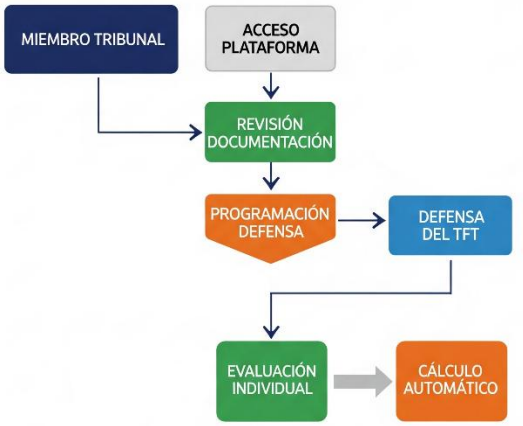

Ilustración 22: Esquema de funcionamiento de los miembros del tribunal evaluador

3.1.8. Perfil de Estudiante

El **estudiante** tiene la función de desarrollar el TFT aplicando los conocimientos adquiridos durante su formación académica, presentar su propuesta inicial tal y como establece el reglamento de TFT de la EITE, defender su trabajo ante el tribunal evaluador, estar atento a los plazos académicos establecidos desde la presentación de propuestas hacia la defensa del trabajo, ha de mantener una comunicación fluida con su tutor de manera regular y asegurarse que cumple con los requisitos académicos para poder presentarse a la defensa de su TFT.

El **estudiante** podrá acceder a una pantalla con el listado de todos los formularios que necesitará para su TFT, como la solicitud de defensa y la solicitud de anteproyecto, entre otros. pudiéndolos descargar desde la plataforma. El **estudiante** también puede acceder a su TFT asignado. Desde allí, podrá ver el flujo de trabajo completo, desde la fase de **anteproyecto** hasta la **evaluación final**. En la ilustración 23 podemos ver un esquema simple que presenta las funciones del **estudiante**.

Ilustración 23: Esquema de estudiante

4. Diseño

En este capítulo vamos a dedicarnos a analizar todas las decisiones que han sido tomadas durante la fase de diseño de la plataforma web. A continuación, se detallarán los requisitos funcionales organizados por tipo de usuario y, posteriormente, se describirán los elementos que compondrán cada una de las pantallas de la plataforma. También, se mostrará el diseño de la base de datos que la Plataforma de Gestión de TFT de la EITE tendrá.

4.1. Requisitos

Desde un primer momento se tenía claro durante el diseño de esta plataforma para la gestión de TFT que el objetivo mínimo de esta era proveer de un sistema unificado, seguro y fácilmente accesible para todo tipo de usuarios siendo esto nuestro requisito general. Pudiendo acceder sin problema un usuario de Windows, como uno de MAC o uno de Linux teniendo garantizada debido a como se ha programado esta plataforma que en todo momento estaremos en un entorno seguro.

Además del punto anterior, existen otros requisitos que la aplicación debe cumplir. Estos se analizarán en el siguiente orden: primero, los que son comunes a todos los perfiles; segundo, los específicos para cada tipo de usuario; y, para finalizar, los relativos a la seguridad y el control de acceso.

Es muy importante tener en cuenta y debe ser mencionado que los requisitos de usuario se pensaron no solo para la implementación y objetivos descritos en el proyecto actual, sino que se ha dejado la base para implementaciones futuras que debido a limitaciones de tiempo no han podido ser implementadas, un buen ejemplo de esto son los roles específicos, como el del secretario de la EITE o el del tutor académico, cada uno con sus propias funciones. Todas las líneas futuras se mencionarán en el capítulo 7.

4.1.1. Requisitos generales de autenticación

Para todos los usuarios de nuestra aplicación sea cual sea su rol, el proceso de autenticación será el mismo. Con estos requisitos queremos garantizar la integridad y consistencia de los datos.

4.1.1.1. Validaciones comunes para todos los usuarios

Campos obligatorios

- **Nombre completo** (*name*): String máximo de 255 caracteres.
- **Email** (*email*): de un formato valido y único en el sistema.
- **DNI** (*dni*): máximo 9 caracteres y único en el sistema
- **Contraseña** (*password*): mínimo con 6 caracteres y encriptada con Hash.
- **Rol** (*role*): que debe existir en la tabla de la base de datos roles.

Estados del usuario

- **Activo o inactivo** (*active*): boolean y por defecto se encuentra con el valor **true**.

- **Departamento** (*departamento_id*): de carácter opcional porque hay usuarios como los de rol **estudiante** que no tienen asignado uno haciendo referencia al departamento al que pertenezca el profesor.

4.1.1.2 Proceso de autenticación

Login

- Autenticación mediante DNI sin letra.
- Verificación de contraseña encriptada.
- Validación de usuario activo.
- Generación de *token* JWT con **Laravel Sanctum**.
- Carga de relaciones con rol y departamento.

Logout

- Eliminación del *token* de acceso actual.
- Limpieza de sesión

4.1.2 Requisitos específicos por rol

4.1.2.1. Administrador root (admin)

Aunque en el reglamento de TFT de la EITE no se mencione en ningún momento la figura del **administrador root**, se decidió añadir dicho rol con funciones completas para no delegar estas capacidades al rol de **director de la EITE.** Este diseño mejora la seguridad por una razón clave: un rol como el del **secretario** accederá a la plataforma con mucha más frecuencia que el **administrador root** para las tareas diarias. Al separar sus funciones, se evita el uso constante de la cuenta con máximos privilegios, reduciendo así los riesgos.

Tenemos las funciones completas de este rol:

- Acceso total a todas las funcionalidades del sistema.
- Gestión completa de usuarios con un CRUD sin restricciones.
- Configuración del sistema y sus parámetros globales.
- Logs de auditoria accediendo a todos los registros.

Requisitos en la plataforma

- Único rol con permisos de eliminación de usuarios.
- Puede cambiar roles de cualquier usuario.
- Gestión de departamentos y configuraciones.
- Control total y absoluto sobre el sistema.

4.1.2.2. Director de la EITE (director)

Funciones según el reglamento:

- Presidente de la comisión de TFT.
- Gestión de tribunales pudiéndolos crear, asignar y gestionar.
- Aprobación de propuestas.
- Gestión de calendario con los eventos académicos y convocatorias.

Requisitos en la plataforma

- No puede eliminar usuarios pudiendo solo el **administrador root** hacerlo.
- Puede ver a todos los usuarios, pero no editarlos.
- Acceso a todos los TFT.
- Gestión de convocatorias académicas.
- Asignación de tribunales evaluadores.

4.1.2.3. Secretario de la EITE (secretario)

Funciones según el reglamento:

- Hace de secretario de la comisión de TFT
- Supervisión de los formularios. Verificar que los formularios disponibles sean los correctos. Además de gestión documental de actas y documentos oficiales.
- Comunicación oficial mediante notificaciones a estudiantes y tribunales.
- Recopilación de TFT terminados

Requisitos en la plataforma:

- Acceso a gestión de tribunales.
- Verificación de documentación.

4.1.2.4. Miembro de la comisión de TFT (comisión_tft)

Funciones según el reglamento:

- Evaluación de propuestas aprobándolas o rechazándolas.
- Asignación de **tutores**.
- Gestión de tribunales teniendo la posibilidad de crearlos y asignarlo.
- Seguimiento de todos los TFT pudiéndolo monitorear y ver en qué estado se encuentra.

Requisitos en la plataforma:

- Solo puede gestionar TFT, pero no los usuarios.
- Acceso a evaluación de propuestas.
- Seguimiento de TFT en desarrollo.

4.1.2.5. Tutor académico (tutor)

Funciones según el reglamento:

- Gestión de solo sus TFT asignados.
- Seguimiento de sus estudiantes.
- Firma de anteproyectos y memorias.
- Asistencia a defensas de sus TFT.

Requisitos en la plataforma:

- Solo puede ver sus TFT asignados.
- No puede formar parte de tribunales de sus TFT.

4.1.2.6. Tribunal (tribunal)

Funciones según el reglamento:

- Evaluación de TFT asignados solo en los TFT donde participa.
- Asistencia a la defensa de su TFT asignado.
- Deliberación y calificación otorgando su correspondiente calificación.
- Levantamiento de actas con la documentación de evaluaciones.

Requisitos en la plataforma:

- Solo puede evaluar el TFT donde es miembro del tribunal.
- No puede ser tutor del TFT que evalúa.
- Acceso a documentación del TFT para su evaluación.

- Cada miembro del tribunal puede poner nota a su TFT asignado.

4.1.2.7. Estudiante (estudiante)

Funciones según el reglamento:

- Debe estar matriculado en la asignatura TFT.
- Gestión de su propio TFT.
- Presentar su propuesta de TFT.
- Subir la documentación en correcta forma, lugar y plazo.
- Seguimiento de plazos.

Validaciones especificas:

- Solo puede acceder y gestionar su propio TFT.
- Acceso muy limitado a funcionalidades del sistema.
- Presentar en primer lugar a través de la plataforma de carga de documentación toda la documentación sobre su propuesta de TFT y posteriormente subir la documentación de TFT.

4.1.3. Validaciones de seguridad y control de acceso

En este apartado de validaciones describiremos los requisitos que ha de cumplir el sistema de seguridad de la plataforma que garantiza que cada usuario acceda únicamente a los recursos y funcionalidades autorizadas según su rol implementando un modelo de seguridad en capas que protege tanto al *backend* como al *frontend*.

Este enfoque de tres capas, 2 en el *backend* que son el **Middleware** y **Policies** y una en el *frontend* que son los **Guards** garantizan que la *Plataforma de Gestión de TFT de la EITE* mantenga la integridad del proceso de gestión de los TFT según lo establecido en el reglamento de TFT de la EITE protegiendo los datos, así como las funcionalidades del sistema. Esto permite el cumplimiento del *Esquema Nacional de Seguridad* al cual están sujetos a su cumplimiento los organismos públicos, siendo la EITE uno de ellos.

4.1.3.1. Middleware de roles (CheckRole)

Una de las funcionalidades clave del *backend* es actuar como un filtro de seguridad en cada petición HTTP. Este filtro verifica que el usuario autenticado tenga los permisos necesarios para acceder al recurso solicitado. Para ello está el **Middleware de roles** o **CheckRole**.

Requisitos:

- Verificación de autenticación previa comprobando que el usuario este *logueado* antes de validar los roles.
- Validación de roles múltiples con comas permitiendo especificar varios roles válidos para un mismo *endpoint*.
- Respuestas HTTP apropiadas (401, 403) devolviendo los códigos de error estándar para acceso no autorizado.
- *Logging* de intentos de acceso registrando todos los intentos de acceso para realizar auditorías de seguridad. Para ello podemos usar el *log* de Laravel.

4.1.3.2. Policies de autorización

Son clases especializadas que implementan la lógica de negocio para determinar si un usuario puede realizar una acción específica sobre un recurso concreto. Todo ello basado en las reglas establecidas en el Reglamento de TFT de la EITE. Nosotros usamos 2 que son **TftPolicy** y **ConversationPolicy**.

Requisitos TftPolicy:

- Control granular de acceso a TFT definiendo quien puede ver, editar o eliminar cada TFT especifico.
- Verificación de membresía en tribunales comprobando si el usuario es miembro del tribunal asignado al TFT.
- Validación de roles específicos aplicando las reglas según el rol del usuario.
- Acceso diferencia por operación de *view/update* permitiendo diferentes niveles de acceso según la acción.

Requisitos de ConversationPolicy

- Control de acceso a conversaciones determinando quien puede ver cada conversación.
- Verificación de participación solo permitiendo el acceso a usuarios que participan en la conversación.
- Permisos de eliminación restringiendo la eliminación de conversaciones según el rol.

4.1.3.3. Validaciones de Frontend

El *frontend* debe incluir un sistema de validación propio que complemente las verificaciones del *backend*, garantizando así una experiencia de usuario más fluida y segura. Para ello tenemos los **guards** que describiremos y enunciaremos su respectivo requisito.

- **authGuard**: debe verificar la autenticación comprobando que el usuario haya iniciado sesión antes de cargas los componentes.
- **roleGuard**: debe validar los roles específicos verificando que el usuario tenga el rol requerido para acceder a una ruta.

Además, nos encontramos con los siguientes requisitos en el *frontend*:

- Redirección automática según permisos navegando de manera automática a paginas apropiadas cuando se detecta acceso no autorizado.
- Control de visibilidad de menús ocultando o mostrando elementos de la interfaz según los permisos del usuario.

4.2. Diseño de la base de datos

Una vez establecidos los requisitos de nuestra plataforma de manera específica se puede continuar con el diseño. Para ello nuestra segunda parada es la base de datos. Con este orden cumplimos la buena práctica de primero diseñar el modelo de datos antes del resto de componentes como puede ser el controlador y la presentación.

La base de datos de esta plataforma se ha diseñado de manera que se implemente un modelo relacional que refleje fielmente al reglamento de TFT de la EITE y estructurado en 15 tablas principales que gestionan los usuarios y roles del sistema académico, los **Trabajos de Fin de Título** con su flujo de trabajo completo, propuestas y evaluaciones, tribunales evaluadores, documentación, calendario académico y convocatorias y finalmente la comunicación interna.

La arquitectura establecida para el diseño de base de datos garantiza la normalización de datos, integridad referencial y seguridad de acceso mediante claves foráneas, *constraints* y validaciones específicas para cada rol. Hay que mencionar que al igual que los requisitos, se ha pensado en el diseño de la base de datos no solo para el producto actual sino para implementaciones futuras dejando una base sólida para ello.

Como nota es importante recalcar que esta base de datos puede ser escalable a otra facultad o al conjunto de la **ULPGC** en su totalidad con pocas modificaciones sencillas.

Tenemos los siguientes objetivos en la base de datos de la *Plataforma de Gestión de TFT de la EITE*:

- Gestionar el ciclo completo de TFT desde la propuesta inicial hasta su evaluación final.
- Implementar el reglamento de la EITE de TFT.
- Facilitar la coordinación entre **estudiantes**, **tutores**, **comisión TFT** y **tribunales**.
- Garantizar la trazabilidad para la toma de decisiones académicas.
- Mantener la integridad de los datos y el cumplimiento del **Esquema Nacional de Seguridad**.
- Establecer una base sólida que permita fácilmente nuevas implementaciones que se quieran hacer en el futuro.

La estructura de base de datos elegida para nuestra plataforma ofrece una solución completa y robusta para la gestión de TFTs, asegurando la trazabilidad, la integridad y el cumplimiento del reglamento de la EITE durante todo el proceso.

Para tener un amplio panorama de la base de datos observaremos la Ilustración 24 que se trata del esquema general de la base de datos para proceder con el análisis particular de cada tabla.

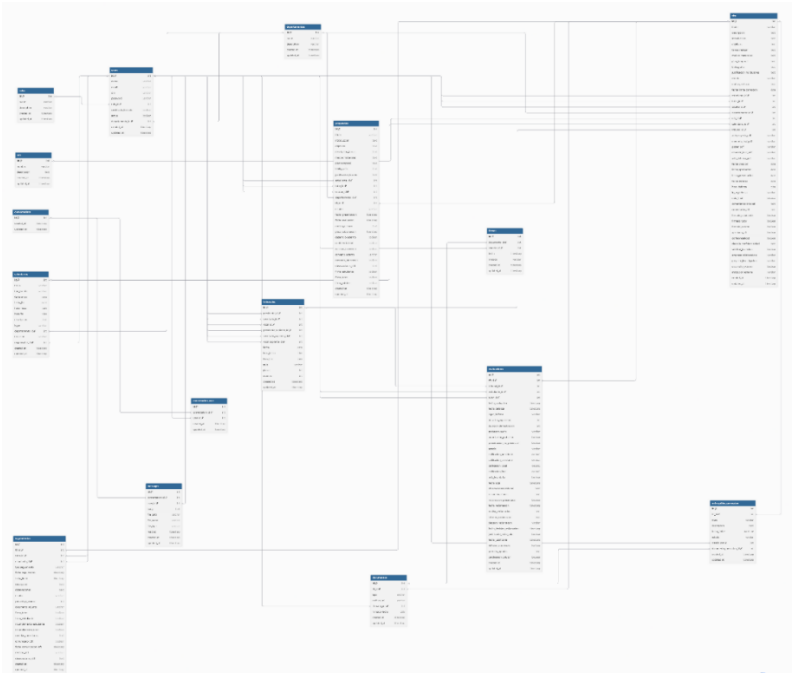

Ilustración 24: Esquema total de la base de datos

4.2.1. Tabla roles

Define los roles y permisos de los usuarios en el sistema según el reglamento de la EITE de TFT. Podemos observar en la Ilustración 25 sus campos en dicha tabla.

Ilustración 25: Tabla roles

Ahora detallaremos los campos:

- *id*: identificador único autoincrementado.
- *name*: nombre del rol que puede tomar los valores **admin, director, secretario, comisión_tft, tutor, cotutor** (planeado para futuras actualizaciones), **tribunal, estudiante, administrativo** (planeado para futuras actualizaciones), **bibliotecario** (planeado para futuras actualizaciones) y **subdirector** (planeado para futuras actualizaciones).
- *description*: aquí se encuentra la descripción detallada del rol y sus funciones.
- *created_at*: fecha de creación del registro.
- *updated_at*: fecha de última actualización.

4.2.2. Tabla departamentos

Contiene la organización departamental de la EITE para la gestión académica. Podemos observar esta tabla con sus campos en la Ilustración 26.

Ilustración 26: Tabla departamentos

Tiene los siguientes campos:

- *id*: identificador único autoincremental.
- *name*: nombre del departamento que puede tomar los valores de Señales y Comunicaciones, Ingeniería Electrónica y Automática, Telemática, Física y Matemáticas.
- *description*: es la descripción del departamento y sus áreas de conocimiento.
- *created_at*: fecha de creación del registro.
- *updated_at*: fecha de última actualización.

4.2.3. Tabla users

Tiene a los usuarios del sistema con sus credenciales y roles asignados. Podemos verla en la Ilustración 27.

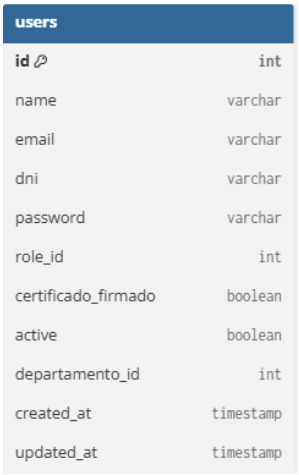

Ilustración 27: Tabla users

Tiene los siguientes campos:

- *id*: identificador único autoincremental.
- *name*: nombre completo del usuario.
- *email*: dirección de correo electrónica del usuario único.
- *dni*: es el Documento Nacional de Identidad único.
- *password*: es la contraseña encriptada con Hash.
- *role*: es el rol del usuario haciendo referencia a la tabla roles.name.
- *certificado_firmado*: boolean que indica si el usuario tiene certificado de firma (planeado para futuras actualizaciones).
- *active*: boolean que indica si el usuario esta activo en el sistema.
- *departamento_id*: hace referencia al departamento del usuario si lo tiene *departamentos.name*.

- *created_at*: fecha de creación del registro.
- *updated_at*: fecha de última actualización.

4.2.4. Tabla ods

Esta tabla contiene los Objetivos de Desarrollo Sostenible para clasificar los TFT en función de estos (planeado para futuras actualizaciones). De todas maneras, se puede observar dicha tabla con sus campos en la Ilustración 28.

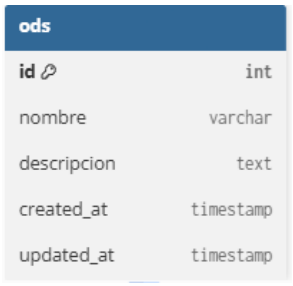

Ilustración 28: Tabla ods

Tiene los siguientes campos:

- *id*: identificador único autoincremental. (planeado para futuras actualizaciones).
- *nombre*: contiene el nombre completo del ODS. (planeado para futuras actualizaciones).
- *description*: tiene la descripción detallada del objetivo. (planeado para futuras actualizaciones).
- *created_at*: fecha de creación del registro. (planeado para futuras actualizaciones).
- *updated_at*: fecha de ultima de actualización. (planeado para futuras actualizaciones).

4.2.5. Tabla calendarios

Contiene la gestión de eventos académicos, convocatorias y plazos del sistema. Donde podemos observar en la Ilustración 29 dicha tabla con sus campos.

Ilustración 29: Tabla calendarios

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *titulo*: Título del evento o convocatoria.
- *tipo_evento*: Tipo de evento (**presentacion_propuestas, defensa_tft, reunion_ctft**, etc.)
- *fecha_inicio*: Fecha de inicio del evento
- *fecha_fin*: Fecha de fin del evento
- *hora_inicio*: Hora de inicio (opcional)
- *hora_fin*: Hora de fin (opcional)
- *descripcion*: Descripción detallada del evento
- *lugar*: Ubicación del evento
- *departamento_id*: Departamento responsable del evento
- *titulacion*: Tipo de titulación (**grado, máster, titulo_propio, todas**)
- *es_periodo_habilitado*: Boolean que indica si es período hábil para presentaciones. (planeado para futuras actualizaciones).
- *es_periodo_inhabil*: Boolean que indica si es período no hábil. (planeado para futuras actualizaciones).
- *observaciones*: Observaciones adicionales
- *activo*: Boolean que indica si el evento está activo. (planeado para futuras actualizaciones).
- *repetitivo*: Boolean que indica si el evento se repite. (planeado para futuras actualizaciones).

- *frecuencia_repeticion*: Frecuencia de repetición (**anual, semestral, trimestral, mensual)**. (planeado para futuras actualizaciones).
- *color_calendario*: Color para visualización en calendario
- prioridad: Prioridad del evento (alta, media, baja)
- *responsable_id*: Usuario responsable del evento. (planeado para futuras actualizaciones).
- *publicado*: Boolean que indica si el evento está publicado. (planeado para futuras actualizaciones).
- *notificacion_activa*: Boolean que indica si hay notificaciones activas. (planeado para futuras actualizaciones).
- *dias_anticipacion_notificacion*: Días de anticipación para notificaciones. (planeado para futuras actualizaciones).
- *created_at*: Fecha de creación del registro
- *updated_at:* Fecha de última actualización

4.2.6. Tabla tribunales.

Contiene a los tribunales evaluadores para la defensa de TFT. Podemos observar en detalle dicha tabla con sus correspondientes campos en la Ilustración 30.

Ilustración 30: Tabla tribunales

Tiene los siguientes campos:

- *id*: Identificador único autoincremental.
- *presidente_id*: Usuario que actúa como presidente del tribunal
- *secretario_id*: Usuario que actúa como secretario del tribunal
- *vocal_id*: Usuario que actúa como vocal del tribunal
- *presidente_suplente_id*: Usuario suplente del presidente
- *secretario_suplente_id*: Usuario suplente del secretario
- *vocal_suplente_id*: Usuario suplente del vocal
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.7. Tabla tfts

Contiene a los TFT con toda su información y estado. Podemos ver en la Ilustración 31 la tabla con sus campos.

Ilustración 31: Tabla tfts

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *titulo*: Título del TFT
- *descripcion*: Descripción general del trabajo. (planeado para futuras actualizaciones).
- *introduccion*: Introducción y antecedentes del TFT. (planeado para futuras actualizaciones).
- *objetivos*: Objetivos específicos a conseguir. (planeado para futuras actualizaciones).
- *tareas_realizar*: Descripción detallada de las tareas. (planeado para futuras actualizaciones).
- *medios_materiales*: Medios y materiales necesarios. (planeado para futuras actualizaciones).
- *plan_temporal*: Plan temporal de desarrollo. (planeado para futuras actualizaciones).
- *bibliografia*: Bibliografía en formato IEEE. (planeado para futuras actualizaciones).
- *justificacion_multitutores*: Justificación para múltiples tutores. (planeado para futuras actualizaciones).
- *estado*: Estado actual del TFT (anteproyecto, propuesta_presentada, propuesta_aceptada, etc.)
- *motivo_rechazo*: Motivo en caso de rechazo
- *fecha_limite_correccion*: Fecha límite para correcciones
- *estudiante_id*: Estudiante autor del TFT
- *tutor_id*: Tutor académico asignado
- *cotutor_id*: Co-tutor asignado (opcional). (planeado para futuras actualizaciones).
- *ods_id*: ODS relacionado con el TFT. (planeado para futuras actualizaciones).
- *calendario_id*: Convocatoria académica
- *tribunal_id*: Tribunal evaluador asignado
- *anteproyecto_pdf*: URL del archivo de anteproyecto
- *memoria_final_pdf*: URL del archivo de memoria final
- *poster_pdf*: URL del archivo de póster. (planeado para futuras actualizaciones).
- *resumen_ieee_pdf*: URL del resumen IEEE. (planeado para futuras actualizaciones).
- *acta_defensa_pdf*: URL del acta de defensa. (planeado para futuras actualizaciones).

- *fecha_creacion*: Fecha de creación del TFT
- *fecha_aprobacion*: Fecha de aprobación por CTFT
- *fecha_presentacion*: Fecha de presentación
- *fecha_defensa*: Fecha de defensa
- *hora_defensa*: Hora de la defensa
- *lugar_defensa*: Lugar de la defensa
- *nota_final*: Calificación final del TFT
- *comentarios_tribunal*: Comentarios del tribunal
- *comentarios_ctft*: Comentarios de la comisión TFT
- *firmado_estudiante*: Boolean que indica si el estudiante ha firmado. (planeado para futuras actualizaciones).
- *firmado_tutor*: Boolean que indica si el tutor ha firmado. (planeado para futuras actualizaciones).
- *firmado_cotutor*: Boolean que indica si el co-tutor ha firmado. (planeado para futuras actualizaciones).
- *aprobado_ctft*: Boolean que indica si está aprobado por CTFT
- *confidencialidad*: Boolean que indica si tiene cláusulas de confidencialidad. (planeado para futuras actualizaciones).
- *clausula_confidencialidad*: Texto de la cláusula de confidencialidad. (planeado para futuras actualizaciones).
- *solicitud_invencion*: Boolean que indica si está sujeto a solicitud de invención. (planeado para futuras actualizaciones).
- *empresa_colaboradora*: Empresa colaboradora. (planeado para futuras actualizaciones).
- *proyecto_investigacion*: Proyecto de investigación relacionado. (planeado para futuras actualizaciones).
- *desarrollo_externo*: Boolean que indica si se desarrolla externamente. (planeado para futuras actualizaciones).
- *institucion_externa*: Institución externa colaboradora. (planeado para futuras actualizaciones).
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.8. Tabla propuestas

Contiene las propuestas de TFT antes de su aprobación. Podemos ver en la Ilustración 32 dicha tabla con sus campos.

propuestas	
id	int
titulo	varchar
introduccion	text
objetivos	text
descripcion_tareas	text
medios_materiales	text
plan_temporal	text
bibliografia	text
justificacion_tutores	text
estudiante_id	int
tutor_id	int
cotutor_id	int
departamento_id	int
tft_id	int
estado	varchar
fecha_presentacion	timestamp
fecha_evaluacion	timestamp
motivo_rechazo	text
plazo_subsanacion	timestamp
desarrollo_externo	boolean
confidencialidad	boolean
solicitud_invencion	boolean
convenio_externo	varchar
memoria_intermedia	boolean
observaciones_ctft	text
firma_estudiante	boolean
firma_tutor	boolean
firma_cotutor	boolean
created_at	timestamp
updated_at	timestamp

Ilustración 32: Tabla propuestas

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *titulo*: Título de la propuesta
- *introduccion*: Introducción y antecedentes. (planeado para futuras actualizaciones).

- *objetivos*: Objetivos específicos. (planeado para futuras actualizaciones).
- *descripcion_tareas*: Descripción de tareas con hitos. (planeado para futuras actualizaciones).
- *medios_materiales*: Medios materiales necesarios. (planeado para futuras actualizaciones).
- *plan_temporal*: Plan temporal de desarrollo. (planeado para futuras actualizaciones).
- *bibliografia*: Bibliografía en formato IEEE. (planeado para futuras actualizaciones).
- *justificacion_tutores*: Justificación de múltiples tutores. (planeado para futuras actualizaciones).
- *estudiante_id*: Estudiante que presenta la propuesta
- *tutor_id*: Tutor académico asignado
- *cotutor_id*: Co-tutor asignado. (planeado para futuras actualizaciones).
- *departamento_id*: Departamento del TFT
- *tft_id*: TFT relacionado (cuando se aprueba)
- *estado*: Estado de la propuesta (borrador, enviada, en_evaluacion, aprobada, rechazada, subsanada, anulada)
- *fecha_presentacion*: Fecha de presentación
- *fecha_evaluacion*: Fecha de evaluación por CTFT
- *motivo_rechazo*: Motivo del rechazo
- *plazo_subsanacion*: Plazo para subsanación (10 días hábiles)
- *desarrollo_externo*: Boolean que indica desarrollo externo. (planeado para futuras actualizaciones).
- *confidencialidad*: Boolean que indica confidencialidad. (planeado para futuras actualizaciones).
- *solicitud_invencion*: Boolean que indica solicitud de invención. (planeado para futuras actualizaciones).
- *convenio_externo*: Convenio con entidad externa. (planeado para futuras actualizaciones).
- *memoria_intermedia*: Boolean que indica memoria intermedia. (planeado para futuras actualizaciones).
- *observaciones_ctft*: Observaciones de la comisión. (planeado para futuras actualizaciones).
- *firma_estudiante*: Boolean que indica firma del estudiante. (planeado para futuras actualizaciones).

- *firma_tutor*: Boolean que indica firma del tutor. (planeado para futuras actualizaciones).
- *firma_cotutor*: Boolean que indica firma del co-tutor. (planeado para futuras actualizaciones).
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.9. Tabla seguimientos

Contiene al seguimiento obligatorio de TFT que establece el reglamento de la EITE. Esta tabla enteramente se usará para implementaciones futuras. La podemos ver igualmente con sus campos en la Ilustración 33.

Ilustración 33: Tabla seguimientos

- *id:* Identificador único autoincremental
- *tft_id*: TFT al que pertenece el seguimiento

- *tutor_id*: Tutor responsable del seguimiento
- *estudiante_id*: Estudiante del TFT
- *tipo_seguimiento*: Tipo de seguimiento (**reunion, entrega, hito, incidencia, memoria_intermedia, contacto_mensual, verificacion_plazos, evaluacion_intermedia**)
- *fecha_seguimiento*: Fecha del seguimiento
- *fecha_limite*: Fecha límite para el seguimiento
- *descripcion*: Descripción del seguimiento
- *observaciones*: Observaciones adicionales
- *estado*: Estado del seguimiento (**pendiente, completado, retrasado, cancelado, en_proceso**)
- *porcentaje_avance*: Porcentaje de avance del TFT
- *documento_adjunto*: URL del documento adjunto
- *firma_tutor*: Boolean que indica firma del tutor
- *firma_estudiante*: Boolean que indica firma del estudiante
- *incumplimiento_estudiante*: Boolean que indica incumplimiento del estudiante
- *incumplimiento_tutor*: Boolean que indica incumplimiento del tutor
- *medidas_correctoras*: Medidas correctoras aplicadas
- *comunicado_ctft*: Boolean que indica si se comunicó a CTFT
- *fecha_comunicacion_ctft*: Fecha de comunicación a CTFT
- *decision_ctft*: Decisión de la CTFT (**continuar, advertencia, anulacion, cambio_tutor, ampliacion_plazo**)
- *observaciones_ctft*: Observaciones de la CTFT
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.10. Tabla evaluaciones

Esta tabla contiene a las evaluaciones de los TFT donde se encuentra las calificaciones y las fechas de defensa para realizar su presentación, cada una este asignado a un TFT. Podemos verla en la Ilustración 34 con sus campos.

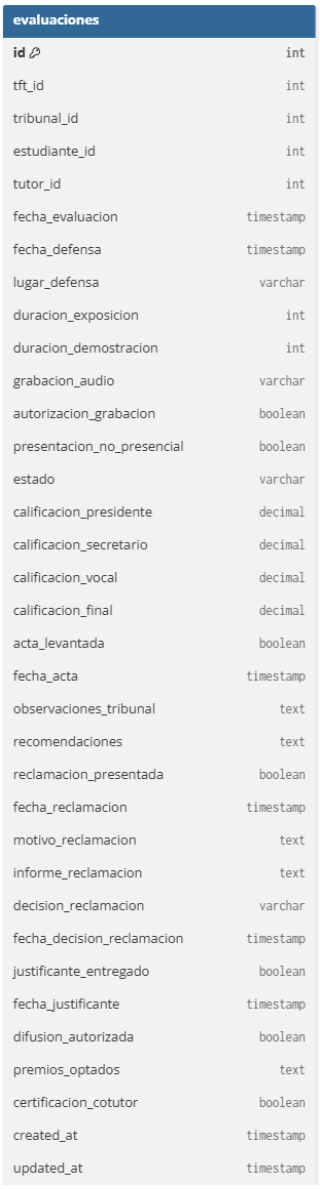

Ilustración 34: Tabla evaluaciones

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *tft_id*: TFT evaluado
- *tribunal_id*: Tribunal evaluador
- *estudiante_id*: Estudiante evaluado
- *tutor_id*: Tutor del TFT
- *fecha_evaluacion*: Fecha de la evaluación
- *fecha_defensa*: Fecha de la defensa
- *lugar_defensa*: Lugar de la defensa
- *duracion_exposicion*: Duración de la exposición (30-45 minutos). (planeado para futuras actualizaciones).

- *duracion_demostracion*: Duración de la demostración (máximo 10 minutos). (planeado para futuras actualizaciones).
- *grabacion_audio*: URL de la grabación de audio. (planeado para futuras actualizaciones).
- *autorizacion_grabacion*: Boolean que indica autorización de grabación. (planeado para futuras actualizaciones).
- *presentacion_no_presencial*: Boolean que indica presentación no presencial. (planeado para futuras actualizaciones).
- *estado*: Estado de la evaluación (programada, en_curso, completada, cancelada, aplazada). (planeado para futuras actualizaciones).
- *calificacion_presidente*: Calificación del presidente
- *calificacion_secretario*: Calificación del secretario
- *calificacion_vocal*: Calificación del vocal
- *calificacion_final*: Calificación final
- *acta_levantada*: Boolean que indica si se levantó acta. (planeado para futuras actualizaciones).
- *fecha_acta*: Fecha del acta. (planeado para futuras actualizaciones).
- *observaciones_tribunal*: Observaciones del tribunal. (planeado para futuras actualizaciones).
- *recomendaciones*: Recomendaciones del tribunal. (planeado para futuras actualizaciones).
- *reclamacion_presentada*: Boolean que indica si hay reclamación. (planeado para futuras actualizaciones).
- *fecha_reclamacion*: Fecha de la reclamación. (planeado para futuras actualizaciones).
- *motivo_reclamacion*: Motivo de la reclamación. (planeado para futuras actualizaciones).
- *informe_reclamacion*: Informe de la reclamación. (planeado para futuras actualizaciones).
- *decision_reclamacion*: Decisión sobre la reclamación. (planeado para futuras actualizaciones).
- *fecha_decision_reclamacion*: Fecha de la decisión. (planeado para futuras actualizaciones).
- *justificante_entregado*: Boolean que indica entrega de justificante. (planeado para futuras actualizaciones).

- *fecha_justificante*: Fecha de entrega del justificante. (planeado para futuras actualizaciones).
- *difusion_autorizada*: Boolean que indica difusión autorizada. (planeado para futuras actualizaciones).
- *premios_optados*: Premios a los que opta el TFT. (planeado para futuras actualizaciones).
- *certificacion_cotutor*: Certificación del co-tutor. (planeado para futuras actualizaciones).
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.11. Tabla documentos

Contiene los documentos adjuntos a los TFT con control de versiones. Podemos verla en la Ilustración 35 con sus campos.

Ilustración 35: Tabla documentos

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *tft_id*: TFT al que pertenece el documento
- *tipo*: Tipo de documento (**anteproyecto, memoria, poster**, etc.)
- *archivo_url*: URL del archivo almacenado
- *firmado_por*: Usuario que firmó el documento
- *firmado_fecha*: Fecha de firma del documento. (planeado para futuras actualizaciones).
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.12. Tabla firmas

Contiene el registro de firmas digitales de documentos. Esta tabla es para ser usada en una implementación futura. Igualmente se puede observar en la Ilustración 36.

Ilustración 36:Tabla firmas

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *documento_id*: Documento firmado
- *usuario_id*: Usuario que firma
- *fecha*: Fecha y hora de la firma
- *metodo*: Método de firma utilizado
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.13. Tabla conversations

Contiene las conversaciones entre usuarios del sistema. Podemos observarLA en la Ilustración 37 con sus campos.

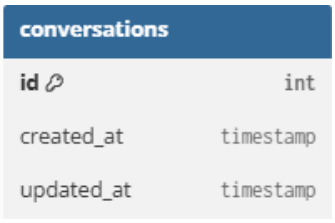

Ilustración 37: Tabla conversations

Posee los siguientes campos:

- *id*: Identificador único autoincremental
- *created_at*: Fecha de creación del registro

- *updated_at*: Fecha de última actualización

4.2.14. Tabla messages

Dentro de cada conversación nos encontramos los mensajes que son contenidos en esta tabla. Podemos observar la tabla en detalle con sus campos en la Ilustración 38.

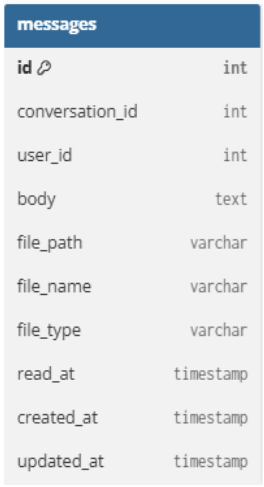

Ilustración 38: Tabla messages

Resultando en los siguientes campos:

- *id*: Identificador único autoincremental
- *conversation_id*: Conversación a la que pertenece
- *user_id*: Usuario que envía el mensaje
- *body*: Contenido del mensaje
- *file_path*: Ruta del archivo adjunto. (planeado para futuras actualizaciones).
- *file_name*: Nombre del archivo adjunto. (planeado para futuras actualizaciones).
- *file_type*: Tipo de archivo adjunto. (planeado para futuras actualizaciones).
- *read_at*: Fecha de lectura del mensaje. (planeado para futuras actualizaciones).
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.2.15. Tabla entregables_correcion

En caso de ser una propuesta rechazada, se ha de entregar una corrección de está siendo esta tabla el contenedor de dicha tabla. Podemos observar dicha tabla en la Ilustración 39 con sus campos.

Ilustración 39: Tabla entregables_correcion

Tiene los siguientes campos:

- *id*: Identificador único autoincremental
- *tft_id*: TFT al que pertenece el entregable
- *titulo*: Título del entregable
- *descripcion*: Descripción del entregable
- *fecha_limite*: Fecha límite para entrega
- *estado*: Estado del entregable (**pendiente, enviado, revisado**)
- *creado_por*: Usuario que creó el entregable. (planeado para futuras actualizaciones).
- *documento_correccion_id*: Documento de corrección relacionado
- *created_at*: Fecha de creación del registro
- *updated_at*: Fecha de última actualización

4.3. Patrón arquitectónico

La plataforma de gestión de TFT de la EITE implementa una arquitectura de microservicios distribuidos basada en el patrón **Modelo-Vista-Controlador (MVC)** en el *backend* y arquitectura basada en componentes en el *frontend*. Esta arquitectura se fundamenta en principios de *Clean Architecture* y *Domain-Driven Design (DDD),* garantizando la separación de capas y la independencia entre componentes.

El sistema se estructura en dos aplicaciones independientes que se comunican mediante **API REST**:

- *Backend*: **Laravel 11** con arquitectura **MVC** y **API RESTful.**
- *Frontend*: **Angular 17** con arquitectura basada en componentes y servicios

A continuación, vamos a analizar un poco más en detalle la arquitectura del *backend* y del *frontend*.

4.3.1. Arquitectura del backend

El *backend* implementa una arquitectura **MVC (Model-View-Controller)** con **API RESTful**, siguiendo los principios de **Clean Architecture** y **Repository Pattern**.

Nos hemos sustentado en esta elección por las siguientes razones**:**

- **Laravel 11**: *Framework* PHP maduro con excelente soporte para APIs REST, ORM robusto (Eloquent) y sistema de autenticación integrado (**Sanctum**)
- **MVC**: Separación clara de responsabilidades, facilitando el mantenimiento y *testing.*
- **API RESTful**: Interfaz estándar y escalable para comunicación con el *frontend*
- **Clean Architecture**: Independencia de *frameworks* y bases de datos, facilitando cambios futuros

En la Ilustración 40 podemos ver un esquema de las capas y clases usadas dentro del *backend*.

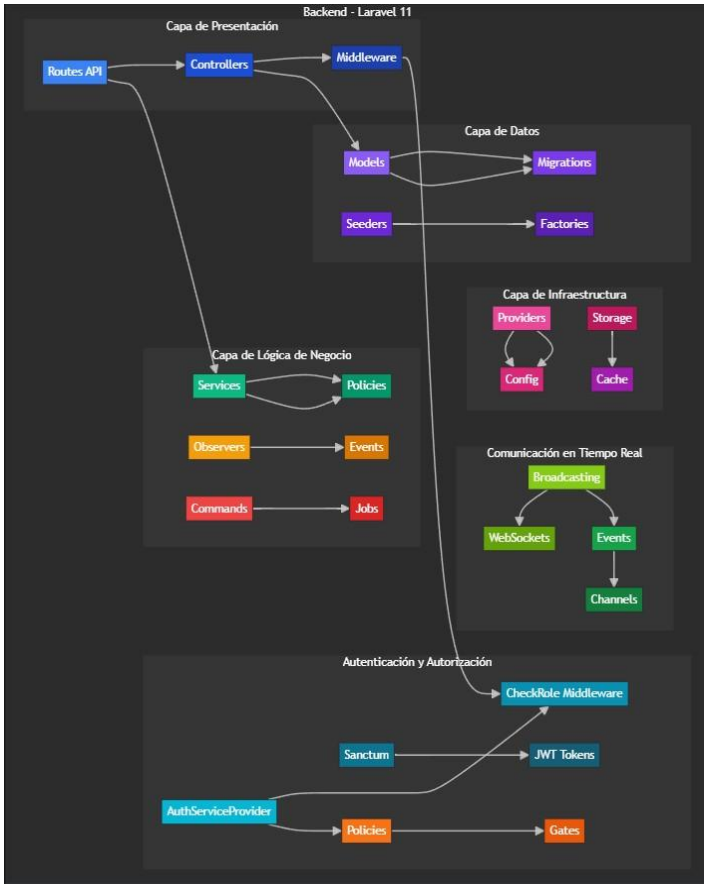

Ilustración 40: Esquema del patrón arquitectónico usado en el backend

4.3.1.1. Capa de presentación (Controllers)

Son los controladores que manejan las peticiones HTTP y coordinan la lógica de negocio. Está conformado por los siguientes componentes**:**

- **AuthController**: Gestión de autenticación y autorización
- **TFTController**: Gestión completa de trabajos de fin de titulación
- **UserController**: Administración de usuarios y roles
- **TribunalController**: Gestión de tribunales evaluadores
- **CalendarioController**: Gestión de eventos académicos
- **DocumentoController**: Manejo de documentos y archivos

Se ha elegido este tipo de arquitectura por las siguientes razones**:**

- **Separación por dominio**: Cada controlador maneja una entidad específica del reglamento EITE.
- **Responsabilidad única**: Cada controlador tiene una función específica y bien definida.
- **Testabilidad**: Fácil de testear de forma aislada mediante **PHPUnit.**

4.3.1.2. Capa de lógica de negocio (Models)

Son los **Modelos Eloquent** que representan las entidades del dominio y encapsulan la lógica de negocio. Se encuentra conformado por los siguientes componentes**:**

- **User**: Gestión de usuarios con roles y permisos
- **Tft**: Entidad principal que representa un trabajo de fin de titulación
- **Tribunal**: Composición y gestión de tribunales evaluadores
- **Calendario**: Eventos académicos y convocatorias
- **Seguimiento**: Control del seguimiento obligatorio de TFT
- **Evaluacion**: Proceso de evaluación y calificación

Se ha realizado de esta manera por las siguientes razones:

- **Eloquent ORM**: Mapeo objeto-relacional intuitivo y potente
- **Relaciones**: Definición clara de relaciones entre entidades
- **Scopes**: Filtros reutilizables para consultas complejas
- **Accessors/Mutators**: Transformación automática de datos

4.3.1.3. Capa de autorización (policies)

Son las clases que implementan la lógica de autorización basada en roles según el reglamento EITE. Está compuesto por**:**

- **TftPolicy**: control de acceso a trabajos de fin de titulación.
- **ConversationPolicy**: autorización para conversaciones.
- **EvaluacionPolicy**: permisos para evaluaciones.

Se ha desarrollado de esta manera por las siguientes razones:

- **Autorización granular**: Control específico por operación y entidad.
- **Cumplimiento normativo**: Implementación fiel del reglamento EITE.
- **Seguridad**: Prevención de acceso no autorizado a recursos.

4.3.1.4. Capa de Middleware

Son los filtros que procesan las peticiones HTTP antes de llegar a los controladores. Está formado por:

- **CheckRole**: Verificación de roles y permisos.
- **Sanctum**: Autenticación mediante *tokens* JWT.
- **CORS**: Configuración para comunicación *cross-origin.*

Se justifica esta elección por las siguientes razones:

- **Seguridad**: Verificación de autenticación y autorización en cada petición.
- **Reutilización**: Lógica común aplicable a múltiples rutas.
- **Flexibilidad**: Configuración granular de permisos por *endpoint.*

4.3.1.5. Capa de observadores (Observers)

Clases que reaccionan a eventos de los modelos, implementando lógica de negocio automática. Se encuentra conformado por:

- **TftObserver**: Automatización de procesos cuando cambia el estado de un TFT.

Se justifica esta elección por las siguientes razones:

- **Automatización**: Procesos automáticos sin intervención manual.

- **Consistencia**: Garantía de que ciertas acciones se ejecuten siempre.
- **Desacoplamiento**: Lógica de negocio separada de los modelos.

4.3.2. Arquitectura del frontend (Angular 17)

El *frontend* implementa una arquitectura basada en componentes con **separación de responsabilidades** y **gestión de estado reactiva**. Se ha decidido hacer de esta manera por las siguientes razones:

- **Angular 17**: *Framework* maduro con TypeScript, excelente para aplicaciones empresariales compleja.
- **Component-Based Architecture**: Reutilización de componentes y mantenibilidad.
- **TypeScript**: Tipado estático que reduce errores y mejora la productividad.
- **RxJS**: Programación reactiva para manejo eficiente de datos asíncronos.

En la Ilustración 41 podemos ver un esquema de las clases y componentes usadas en el patrón arquitectónico empleado en el *frontend*.

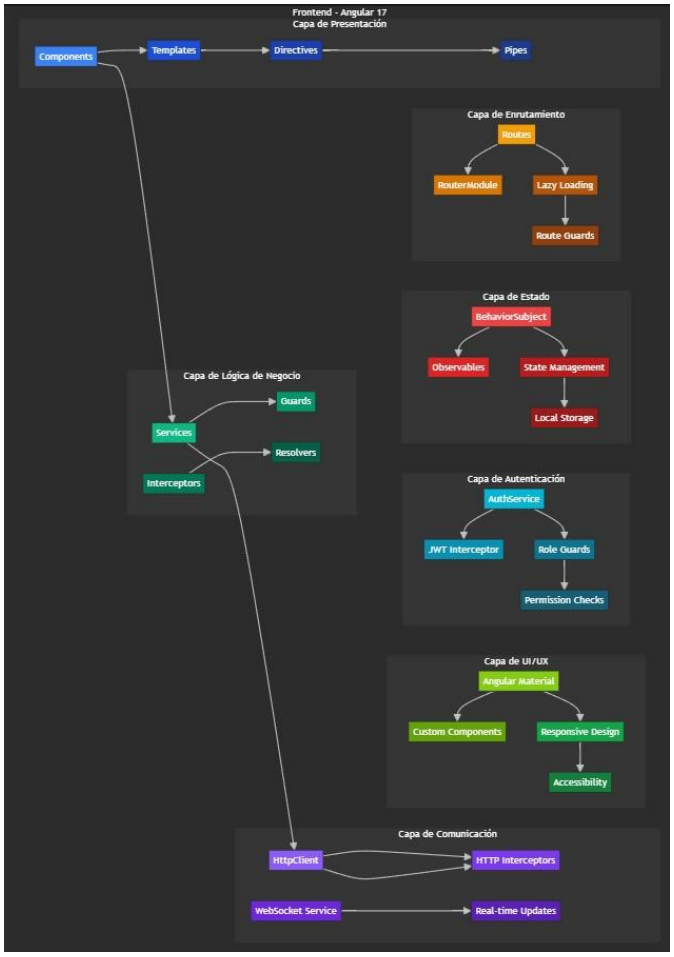

Ilustración 41: Esquema del patrón arquitectónico usado en el frontend

4.3.2.1 Capa de Core (Servicios y Utilidades)

Son los servicios centrales que proporcionan funcionalidades comunes a toda la aplicación. Está formado por:

- **AuthService**: Gestión de autenticación y *tokens.*
- **TftService**: Comunicación con la API de TFT.
- **UserService**: Gestión de usuarios.
- **CalendarioService**: Gestión de eventos académicos.
- **ChatService**: Sistema de mensajería.

Se ha realizado esta elección por las siguientes razones:

- **Centralización**: Lógica común reutilizable en toda la aplicación.
- **Inyección de dependencias**: Facilita el *testing* y la modularidad.
- **Gestión de estado**: Manejo centralizado del estado de la aplicación.

4.3.2.2. Capa de Guards e Interceptors

Son aquellos componentes que protegen las rutas y procesan las peticiones HTTP estando conformado por:

- **AuthGuard**: Protección de rutas según autenticación.
- **RoleGuard**: Verificación de roles específicos.
- **JWTInterceptor**: Inyección automática de tokens en peticiones.

Se ha elegido esta arquitectura por las siguientes razones:

- **Seguridad**: Protección automática de rutas sensibles.
- **UX**: Redirección automática según permisos del usuario.
- **Automatización**: Inyección automática de *headers* de autenticación.

4.3.2.3. Capa de Features (Módulos Funcionales)

Son los módulos organizados por funcionalidad que encapsulan componentes relacionados siendo los siguientes**:**

- **Auth**: Autenticación y gestión de sesión.
- **Dashboard**: Panel principal personalizado por rol.
- **TFT**: Gestión completa de trabajos de fin de titulación.

- **Tribunales**: Gestión de tribunales evaluadores.
- **Calendario**: Gestión de eventos académicos.
- **Usuarios**: Administración de usuarios.
- **Chat**: Sistema de comunicación interna.

Se ha realizado con esta configuración por las siguientes razones:

- **Organización**: Estructura clara y lógica por dominio.
- **Lazy Loading**: Carga bajo demanda para optimizar rendimiento.
- **Mantenibilidad**: Fácil localización y modificación de funcionalidades.

4.3.2.4. Capa de Shared (Componentes Compartidos)

Son los componentes reutilizables que se utilizan en múltiples módulos. Siendo los siguientes:

- **AutocompleteProfesor**: Componente de autocompletado para selección de profesores.
- **Formularios**: Componentes de formularios reutilizables.
- **Layout**: Componentes de estructura y navegación.

Se justifica esta elección por las siguientes razones:

- **Reutilización**: Evita duplicación de código.
- **Consistencia**: Interfaz uniforme en toda la aplicación.
- **Mantenimiento**: Cambios centralizados en componentes compartidos.

4.3.3. Arquitectura de comunicación (API REST)

Es la interfaz de comunicación estandarizada entre *frontend* y *backend* que permite una comunicación entre ambos componentes teniendo las siguientes características**:**

- **RESTful**: Uso de métodos HTTP estándar (**GET, POST, PUT, DELETE**).
- **JSON**: Formato de datos ligero y universal.
- **JWT**: Tokens de autenticación seguros y escalables.
- **CORS**: Configuración para comunicación *cross-origin.*

Nos basamos en esta elección por las siguientes razones:

- **Estándar**: Protocolo ampliamente adoptado y documentado

- **Escalabilidad**: Fácil integración con otros sistemas
- **Seguridad**: *Tokens* JWT con expiración y renovación automática
- **Flexibilidad**: Independencia entre *frontend* y *backend*

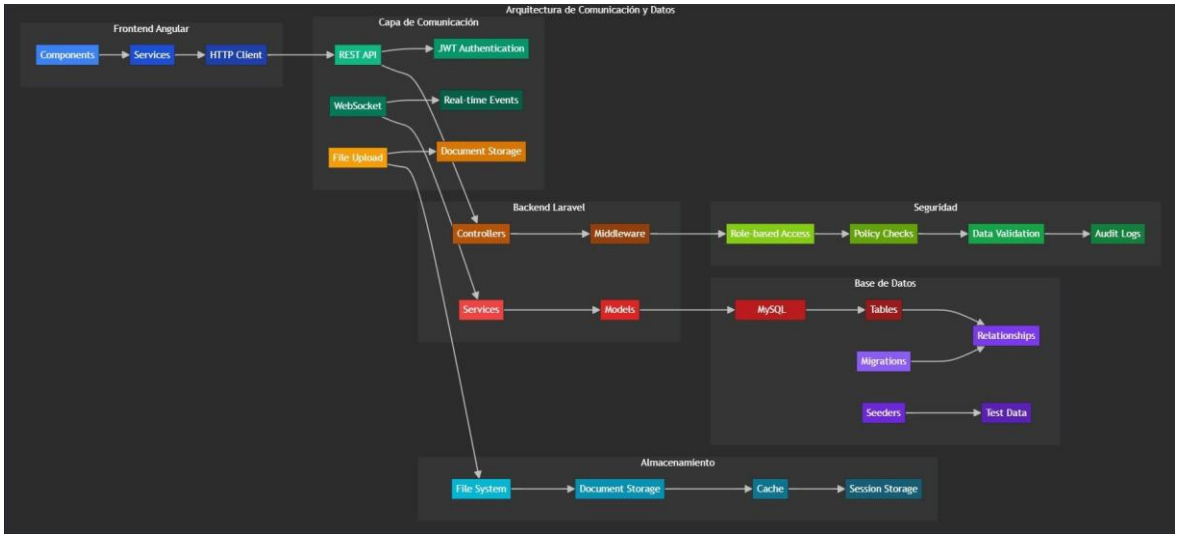

Ilustración 42: Esquema del patrón arquitectónico usado en la comunicación entre el frontend y backend como la de la a base de datos

En la Ilustración 42 podemos ver un esquema de las clases usadas tanto en la arquitectura de comunicación como en la de base de datos que veremos a continuación en el siguiente apartado.

Para mayor claridad, es importante retomar brevemente el tema de la base de datos, específicamente para recalcar su estructura y las razones de su elección, ya mencionadas en el apartado de arquitectura.

Se trata de una base de datos relacional con normalización y relaciones bien definidas que tiene las siguientes características:

- **MySQL**: Sistema de gestión de bases de datos robusto.
- **Normalización**: Estructura optimizada para evitar redundancias.
- **Relaciones**: Claves foráneas para integridad referencial.
- **Índices**: Optimización de consultas frecuentes.

Lo justificamos por las siguientes razones:

- **Integridad**: Garantía de consistencia de datos.
- **Rendimiento**: Consultas optimizadas mediante índices.
- **Escalabilidad**: Capacidad de manejar grandes volúmenes de datos.
- **Mantenibilidad**: Estructura clara y documentada.

4.4. Mockups

Una vez diseñados los requisitos, la base de datos y la arquitectura de la plataforma podemos proceder con el siguiente paso que es el diseño de la parte visual. Antes de ponernos a desarrollar código primero hemos de realizar los *mockups* que nos ayudarán en esta labor para luego poderlos materializar en nuestra plataforma mediante Angular.

Los *mockups* presentados en este apartado constituyen una parte fundamental del proceso de diseño de la interfaz de usuario para la *plataforma de gestión de Trabajos de Fin de Titulación (TFT) de la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria (ULPGC).*

Este trabajo se enmarca dentro del contexto de un Trabajo de Fin de Grado que busca modernizar y digitalizar los procesos administrativos académicos, implementando las mejores prácticas en el diseño de interfaces de usuario y experiencia de usuario (UX/UI). La necesidad de desarrollar estos *mockups* surge de la identificación de problemas específicos en el sistema actual de gestión de TFT, caracterizado por procesos manuales, falta de trazabilidad, dificultades en la comunicación entre actores y ausencia de una plataforma centralizada que facilite el cumplimiento del Reglamento para la Realización y Evaluación de Trabajos de Fin de Título.

La herramienta usada para la realización de los *mockups* fue **Figma**.

4.4.1 Metodología de Diseño Implementada

4.4.1.1. Enfoque Centrado en el Usuario (User-Centered Design)

La metodología de diseño adoptada sigue los principios del *User-Centered Design (UCD),* que coloca al usuario en el centro del proceso de diseño. Esta aproximación se materializa en:

- **Investigación de Usuarios**: Análisis exhaustivo de los diferentes roles involucrados en el proceso de TFT (**estudiantes, tutores, comisión TFT, director, secretario**) y sus necesidades específicas.
- **Personas y Escenarios**: Definición de perfiles de usuario detallados y casos de uso que representan las interacciones típicas con el sistema.
- **Iteración Continua**: Proceso de diseño iterativo basado en validación de prototipos.

4.4.1.2. Principios de Diseño Aplicados

Consistencia Visual y Funcional:

- Uso uniforme de colores institucionales de la ULPGC siendo el azul **#1e40af** y naranja **#ea580c**.
- Patrones de interacción consistentes en todas las pantallas.
- Jerarquía visual clara y predecible.

Accesibilidad Universal:

- Cumplimiento de estándares *WCAG 2.1 (Web Content Accessibility Guidelines).*
- Contraste de colores adecuado para usuarios con discapacidad visual.
- Navegación por teclado para usuarios con limitaciones motoras.

Responsividad y Adaptabilidad:

- Diseño *mobile-first* que se adapta a diferentes dispositivos.
- *Breakpoints* optimizados para *tablets*, *laptops* y pantallas de escritorio.
- Contenido escalable sin pérdida de funcionalidad.

Usabilidad y Eficiencia:

- Reducción del número de clics necesarios para completar tareas
- *Feedback* visual inmediato para todas las acciones
- Prevención de errores mediante validaciones proactivas
- Recuperación fácil de errores con mensajes claros

4.4.2. Marco Teórico y Fundamentación

4.4.2.1. Teorías de Diseño de Interfaces Aplicadas

Principios de Gestalt:

- **Proximidad**: Elementos relacionados se agrupan visualmente.
- **Similitud**: Elementos con función similar comparten características visuales.
- **Continuidad**: Flujo visual natural que guía la atención del usuario.
- **Cierre**: Uso de formas completas para facilitar la comprensión.

Leyes de UX de Nielsen:

- Visibilidad del estado del sistema: Información clara sobre el progreso.
- Correspondencia entre sistema y mundo real: Terminología familiar al usuario.
- Control y libertad del usuario: Opciones de cancelación y deshacer.
- Consistencia y estándares: Patrones de diseño reconocibles.
- Prevención de errores: Validaciones y confirmaciones.
- Reconocimiento antes que recuerdo: Información visible y accesible.
- Flexibilidad y eficiencia: Aceleradores para usuarios expertos.
- Estética y diseño minimalista: Interfaz limpia y funcional.
- Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de errores.
- Documentación de ayuda: Información contextual cuando sea necesaria.

4.4.2.2. Estándares de Diseño Web Moderno

Material Design (Google):

- Elevación y sombras para crear jerarquía visual.
- Animaciones fluidas y transiciones naturales.
- Paleta de colores coherente y accesible.
- Tipografía clara y legible.

Human Interface Guidelines (Apple):

- Claridad: Información legible y comprensible.
- Deferencia: El contenido es el protagonista.
- Profundidad: Jerarquía visual mediante capas.

4.4.3. Proceso de Desarrollo de Mockups

4.4.3.1. Fases del Proceso de Diseño

Fase 1: Investigación y Análisis

- Investigación sobre los roles y su implicación en el procedimiento de TFT (**director, secretario, tutores y estudiantes**)
- Análisis de procesos actuales y puntos de flaqueza.
- *Benchmarking* de plataformas similares en otras universidades
- Definición de requisitos funcionales y no funcionales

Fase 2: Conceptualización

- Creación de *wireframes* de baja fidelidad
- Definición de arquitectura de información
- Establecimiento de flujos de usuario (*user flows*)
- Validación mediante pruebas de usabilidad

Fase 3: Diseño Visual

- Desarrollo de *mockups* de alta fidelidad
- Definición de sistema de diseño (*design system*)
- Creación de componentes reutilizables
- Implementación de guías de estilo

Fase 4: Validación y Refinamiento

- Pruebas de usabilidad simulando los procesos que forman parte de un TFT
- Análisis de métricas de usabilidad
- Iteración basada en *feedback* propio.
- Optimización de rendimiento y accesibilidad

4.4.4. Pantalla de login

La pantalla de *login* constituye el punto de entrada principal al sistema de gestión de TFT de la EITE, diseñada con un enfoque minimalista que prioriza la usabilidad y la seguridad. El *mockup* presenta una disposición centrada con el logo institucional de la EITE en la parte superior, seguido de un formulario de autenticación que incluye campos para DNI y contraseña. La interfaz incorpora elementos visuales que proporcionan *feedback* inmediato al usuario, como validación en tiempo real de los campos y mensajes de error contextuales que guían al usuario en caso de problemas de acceso.

Las funcionalidades implementadas en esta pantalla abarcan un sistema de autenticación robusto que utiliza el DNI como identificador único, eliminando la letra para simplificar el proceso de entrada. El sistema implementa validación en tiempo real que verifica tanto el formato de los campos como la existencia del usuario en la base de datos, proporcionando retroalimentación inmediata al usuario. Una vez autenticado exitosamente, el sistema genera un *token* JWT que garantiza la persistencia de la sesión y redirige automáticamente al usuario a su **dashboard** personalizado según su rol específico en el sistema. El acceso a esta pantalla está disponible para todos los usuarios registrados en el sistema, independientemente de su rol, estableciendo un punto de entrada universal que posteriormente discrimina las funcionalidades según los permisos asignados. El sistema

mantiene un registro detallado de todos los intentos de acceso, tanto exitosos como fallidos, proporcionando una capa adicional de seguridad y auditoría que permite el seguimiento de actividades sospechosas y la generación de reportes de seguridad. Podemos ver en detalle este *mockup* en la Ilustración 43.

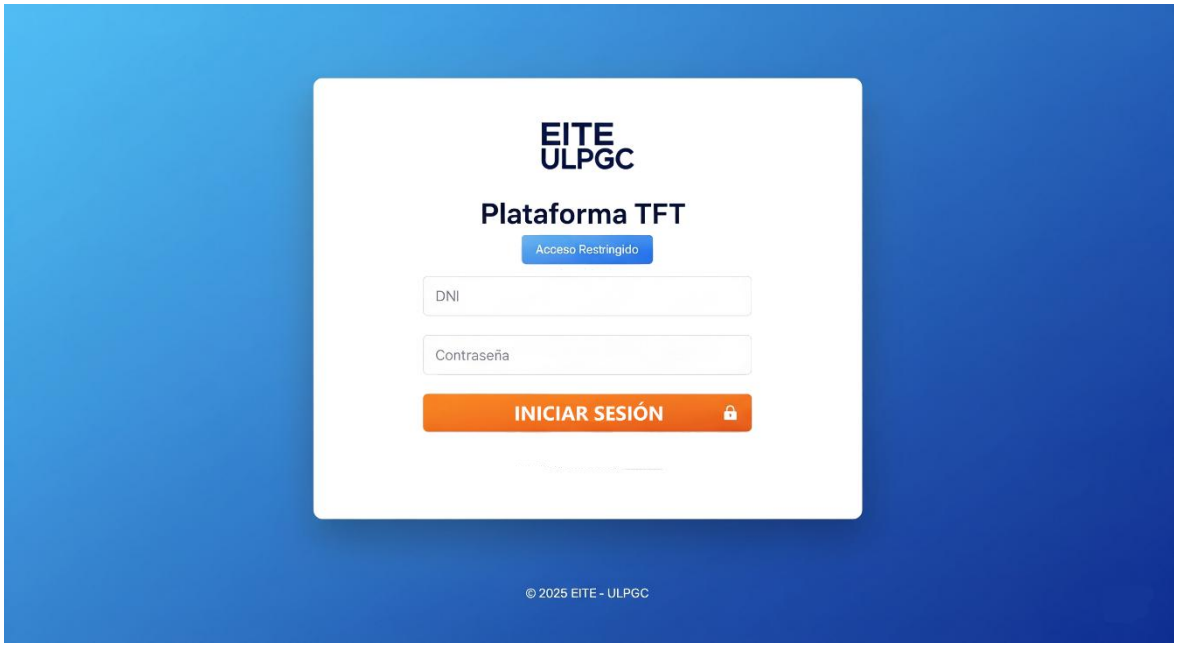

Ilustración 43: Diseño página login

4.4.5. Dashboard

Esta pantalla se diseñó pensando en futuras implementaciones que no pudieron realizarse por limitaciones de tiempo, por lo que quedan propuestas como líneas de trabajo futuro. Aquí se mencionará lo que se tenía pensado en un principio y que se deja como línea futura para luego ir a la parte real que ha sido implementada.

El **dashboard** principal representa el centro neurálgico de la plataforma, diseñado como un panel de control personalizado que se adapta dinámicamente a las necesidades específicas de cada rol de usuario. La interfaz presenta una barra de navegación superior que proporciona acceso rápido a todas las funcionalidades del sistema, junto con un área de perfil de usuario que muestra información contextual y opciones de configuración personal.

El área principal del **dashboard** está organizada en tarjetas de estadísticas que presentan métricas relevantes según el rol del usuario, complementadas con gráficos de actividad que visualizan datos de TFT de manera intuitiva y comprensible.

Para el **director de la EITE**, el **dashboard** se configura como un centro de mando que proporciona una visión panorámica del estado general de los TFT en la escuela. Las tarjetas de estadísticas muestran el total de TFT por estado y convocatoria, destacando aquellos que requieren atención inmediata como propuestas pendientes de evaluación o tribunales que necesitan ser asignados. El sistema incluye acciones rápidas que permiten al director aprobar propuestas, crear tribunales o programar defensas directamente desde el **dashboard**, optimizando el flujo de trabajo administrativo.

El **secretario de la EITE** encuentra en su **dashboard** un enfoque en la gestión documental y la comunicación oficial. Las métricas se centran en la documentación pendiente de procesamiento, como actas de reuniones por levantar o notificaciones por enviar a estudiantes y tribunales. El sistema proporciona acceso directo a la base de datos de TFT finalizados, permitiendo al secretario gestionar la documentación oficial y mantener actualizados los registros académicos.

Los **miembros de la Comisión TFT** disponen de un **dashboard** especializado en la evaluación académica y el seguimiento de TFT. Las estadísticas se enfocan en propuestas pendientes de revisión, TFT en desarrollo que requieren monitoreo, y la gestión de tutores académicos. El sistema facilita la toma de decisiones colegiadas proporcionando información detallada sobre cada TFT y herramientas para la asignación equilibrada de recursos académicos.

Los **Tutores Académicos** acceden a un **dashboard** personalizado que muestra exclusivamente los TFT que tienen asignados, junto con información sobre seguimientos mensuales pendientes y documentación que requiere revisión. La interfaz incluye herramientas de comunicación directa con los estudiantes y un sistema de alertas que notifica cuando se acercan fechas límite importantes o cuando los estudiantes suben nuevos entregables.

Los **Estudiantes** encuentran en su **dashboard** una visión clara del progreso de su TFT, con indicadores visuales que muestran el estado actual del trabajo y los próximos plazos importantes. El sistema proporciona acceso directo a la documentación que deben entregar y herramientas para la comunicación con su tutor académico, facilitando el seguimiento del desarrollo de su trabajo de fin de titulación.

Por ahora, y debido a la falta de tiempo, el **dashboard** solo actúa como un punto de entrada común para todos los usuarios. Su única funcionalidad es un botón, visible para todos, que dirige al reglamento de TFT de la EITE para facilitar su consulta y localización. La barra de navegación superior se mantiene. Podemos ver esto en la Ilustración 44.

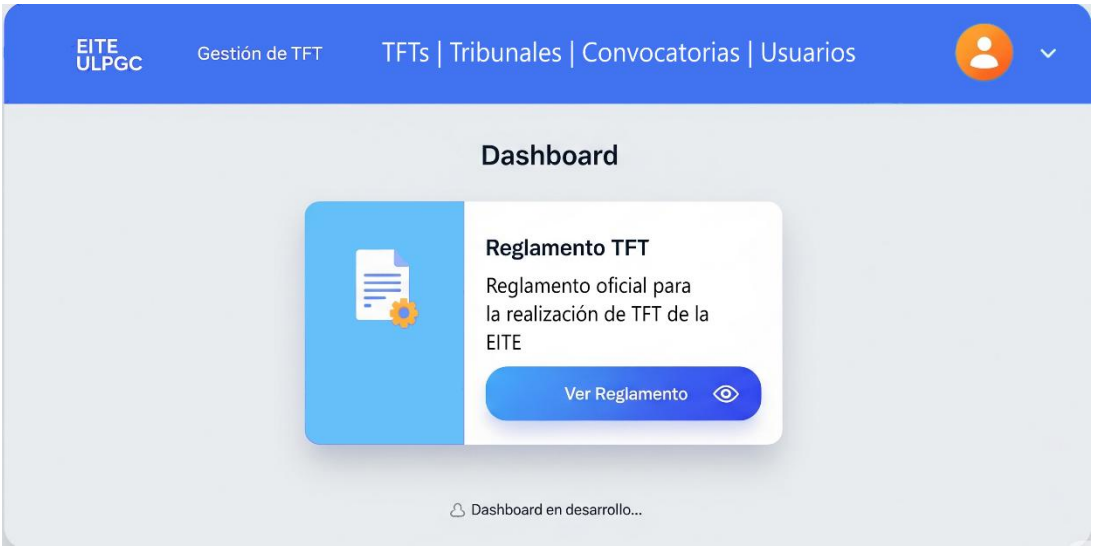

Ilustración 44: Diseño de página dashboard

4.4.6. Gestión de TFT

La pantalla de gestión de TFT constituye el módulo central del sistema, diseñada para proporcionar una interfaz completa y eficiente que permita la administración integral de todos los TFT. El *mockup* presenta una estructura organizada que incluye filtros avanzados en la parte superior, permitiendo la búsqueda de TFTs por múltiples criterios, como su **estado**, **convocatoria**, **departamento**, o bien por el **tutor** o **estudiante asociado**.

La sección principal está ocupada por una tabla dinámica que muestra los TFT con información relevante y acciones contextuales según el rol del usuario.

Para el **director de la EITE** y los miembros de la **Comisión TFT**, esta pantalla se convierte en un centro de control académico que permite la evaluación y aprobación de propuestas de TFT. El sistema implementa un *workflow* automatizado que guía al usuario a través del proceso de evaluación, solicitando motivos detallados en caso de rechazo y proporcionando herramientas para la asignación de tribunales evaluadores. La interfaz incluye funcionalidades avanzadas para la programación de defensas, permitiendo la selección de fechas y horarios que se integran automáticamente con el calendario académico del sistema.

Los **Tutores Académicos** encuentran en esta pantalla las herramientas necesarias para el seguimiento del desarrollo de los trabajos asignados. La interfaz incluye un sistema de seguimiento que permite al tutor registrar el progreso del estudiante.

Los **Estudiantes** acceden a una versión simplificada de esta pantalla que muestra exclusivamente la información relevante para su TFT. La interfaz proporciona una visión clara del estado actual del trabajo, los próximos plazos y la documentación que deben entregar. El sistema incluye herramientas para la subida de entregables con validación automática de formatos y tamaños de archivo.

Podemos ver el *mockup* de esta pantalla en la Ilustración 45.

Ilustración 45: Diseño de página de gestión de TFT

4.4.7. Gestión de tribunales

La pantalla de gestión de tribunales representa un módulo especializado diseñado para facilitar la creación, asignación y administración de los tribunales evaluadores que constituyen el órgano final de evaluación de los TFT. El *mockup* presenta una estructura organizada que incluye una lista general de tribunales con filtros avanzados, un formulario de creación que guía al usuario a través del proceso de asignación de miembros, y un panel detallado que muestra la información completa de cada tribunal.

Para el **director de la EITE** y los **miembros de la Comisión TFT**, esta pantalla proporciona las herramientas necesarias para crear tribunales que cumplan estrictamente con los requisitos establecidos del reglamento. El sistema implementa validaciones automáticas que verifican que los tutores no puedan ser miembros del tribunal, garantizando la imparcialidad en la evaluación. La interfaz incluye funcionalidades para la gestión de suplentes, permitiendo la asignación de miembros alternativos que puedan sustituir a los titulares en caso de ausencia.

La interfaz proporciona herramientas de programación visual que permiten coordinar las fechas de defensa con la disponibilidad de todos los miembros del tribunal, integrando automáticamente esta información con el calendario académico del sistema.

Los miembros del tribunal acceden a una versión personalizada de esta pantalla, que muestra exclusivamente los TFT que tienen asignados para evaluar. La interfaz incluye formularios estructurados para la calificación de los trabajos, con criterios específicos que se alinean con los estándares académicos de la EITE. Podemos ver este *mockup* en la Ilustración 46.

Ilustración 46: Diseño de pantalla de tribunales

4.4.8. Gestión de usuarios

La pantalla de gestión de usuarios constituye un módulo administrativo fundamental que permite la administración integral de todos los usuarios del sistema, desde su creación inicial hasta la gestión de sus permisos y estados. El *mockup* presenta una estructura tabular que muestra la información básica de todos los usuarios, complementada con filtros avanzados que permiten la búsqueda por múltiples criterios como rol, departamento, estado o nombre. La interfaz proporciona un formulario que le permite al **administrador root** crear y editar los perfiles de usuario de forma guiada.

Para el **administrador root**, esta pantalla proporciona acceso completo a todas las funcionalidades de gestión de usuarios, incluyendo la creación, lectura, actualización y eliminación de usuarios. El sistema implementa controles de seguridad que requieren confirmación para operaciones críticas como la eliminación de usuarios o el cambio de roles administrativos. La interfaz incluye herramientas para la gestión de roles que permiten

asignar y modificar permisos de manera granular, asegurando que cada usuario tenga acceso únicamente a las funcionalidades necesarias para su rol específico.

El **director de la EITE** accede a una versión de solo lectura de esta pantalla, permitiendo la consulta de información detallada sobre los usuarios del sistema sin la capacidad de realizar modificaciones. La interfaz proporciona estadísticas sobre la distribución de usuarios por roles y departamentos, facilitando la toma de decisiones administrativas y la planificación de recursos académicos. Podemos ver el *mockup* de esta pantalla en la Ilustración 47.

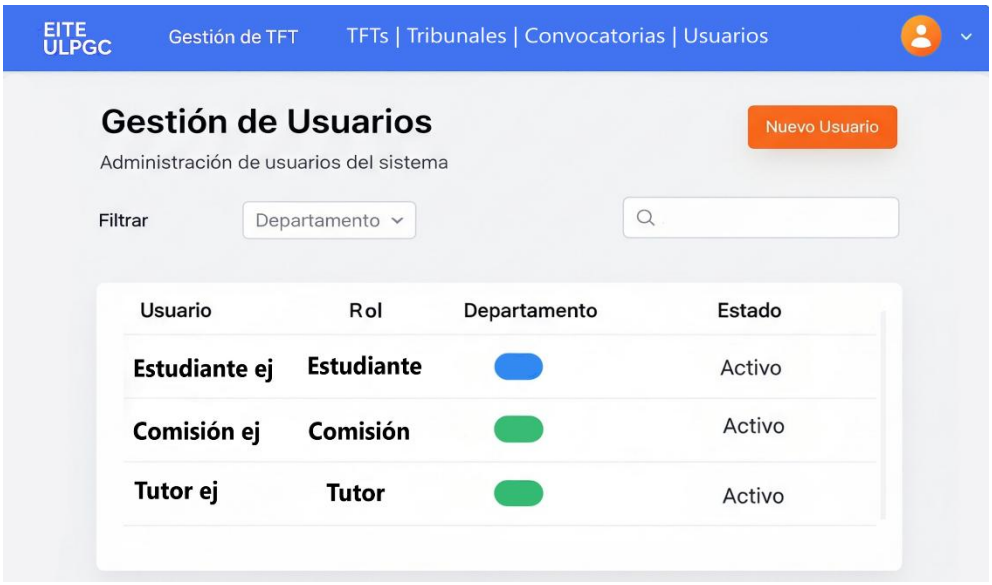

Ilustración 47: Diseño de pantalla de gestión de usuarios

4.4.9. Calendario

La pantalla de gestión de calendario representa un módulo integrado que proporciona una visión completa del calendario académico de la EITE, facilitando la planificación y coordinación de todos los eventos relacionados con los TFT. El *mockup* presenta una interfaz de calendario moderna que permite diferentes vistas temporales, desde una perspectiva semanal hasta una visión anual, adaptándose a las necesidades específicas de cada usuario. Actualmente es más una pantalla de consulta que de planificación que se irá puliendo en implementaciones futuras.

La interfaz incluye herramientas de filtrado que permiten mostrar únicamente los eventos relevantes según el rol del usuario.

Para el **director de la EITE** y los **miembros de la Comisión TFT**, esta pantalla se convierte en una herramienta de planificación estratégica que permite la creación y gestión

de eventos académicos como defensas de TFT, reuniones de la comisión, y períodos de convocatorias.

Podemos ver el *mockup* de esta pantalla en la Ilustración 48.

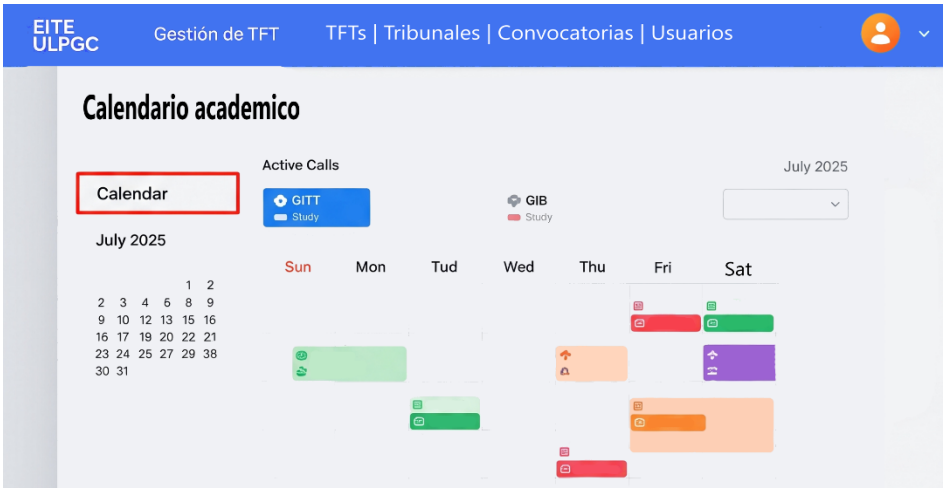

Ilustración 48: Diseño de calendario

4.4.10. Sistema de chat

La pantalla del sistema de chat constituye un módulo de comunicación en tiempo real que facilita la interacción entre todos los participantes del proceso de TFT, desde la comunicación entre estudiante y tutor hasta la coordinación entre miembros del tribunal. El *mockup* presenta una interfaz moderna dividida en dos secciones principales: una lista de conversaciones activas en el panel izquierdo y una ventana de chat principal en el área central. La interfaz incluye un panel de participantes que muestra información sobre los usuarios activos en cada conversación.

El sistema implementa tecnología *WebSockets* que garantiza la comunicación bidireccional en tiempo real, permitiendo que los mensajes se transmitan instantáneamente entre los participantes sin necesidad de recargar la página. El sistema mantiene un historial completo de todas las conversaciones, facilitando la consulta de información anterior y proporcionando trazabilidad en las comunicaciones académicas.

El sistema implementa controles de seguridad que verifican los permisos de cada usuario antes de permitir su participación en conversaciones específicas, asegurando que solo los usuarios autorizados puedan acceder a las comunicaciones relevantes. Podemos ver el *mockup* de esta pantalla en la Ilustración 49.

Ilustración 49: Diseño de pantalla chat

4.4.11. Gestión de documentación

La pantalla de gestión de documentación representa un módulo especializado diseñado para facilitar la administración integral de todos los documentos relacionados con los TFT, desde la propuesta inicial hasta la documentación final de evaluación.

El *mockup* presenta un explorador de archivos moderno que organiza los documentos por TFT. El sistema implementa un visualizador integrado que permite la consulta directa de documentos en formatos comunes como PDF, eliminando la necesidad de descargar archivos para su revisión. Podemos observar el *mockup* de la pantalla de gestión de documentación en la Ilustración 50.

Ilustración 50: Diseño de pantalla de formularios

5. Implementación

Una vez realizado el proceso de diseño, ya podemos proceder con el desarrollo propiamente dicho de la plataforma. Simplemente es llevar el diseño a la realidad. A continuación, describiremos las funcionalidades implementadas en el sistema. Además, detallaremos los tests de *backend* y *frontend* que se utilizaron para validar que el desarrollo cumple con los requisitos del diseño.

La *plataforma de gestión de Trabajos de Fin de Titulación (TFT) de la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE)* implementa una arquitectura moderna basada en el **patrón cliente-servidor** con separación clara de responsabilidades. El sistema está compuesto por un *backend* desarrollado en **Laravel 11** que proporciona una **API RESTful** robusta, y un *frontend* construido con **Angular 17** que ofrece una interfaz de usuario moderna y responsiva.

La arquitectura implementa el patrón MVC (*Model-View-Controller*) en el *backend*, complementado con el patrón de servicios y políticas de autorización que garantizan la seguridad y escalabilidad del sistema. En el *frontend*, se utiliza una **arquitectura basada en componentes** con **inyección de dependencias** y **gestión de estado reactiva** que facilita el mantenimiento y la extensibilidad del código.

5.1. Implementación del backend (Laravel 11)

5.1.1. Controlador de Autenticación (AuthController)

El controlador de autenticación gestiona todo el proceso de identificación y autorización de usuarios en el sistema, implementando las mejores prácticas de seguridad y validación de datos.

Método: login(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con DNI y contraseña
- **Parámetros de salida:** *JsonResponse* con token JWT y datos del usuario
- **Funcionalidad:** Autentica al usuario mediante DNI y contraseña, verifica el estado activo, genera token JWT y retorna información del usuario con sus relaciones cargadas.

Método: register(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con datos completos del usuario
- **Parámetros de salida:** *JsonResponse* con usuario creado y token
- **Funcionalidad:** Crea un nuevo usuario en el sistema con validación completa de datos, encriptación de contraseña y asignación de roles.

Método: logout(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* autenticado
- **Parámetros de salida:** *JsonResponse* de confirmación
- **Funcionalidad:** Elimina el token de acceso actual del usuario y limpia la sesión.

Método: profile(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* autenticado
- **Parámetros de salida:** *JsonResponse* con datos del usuario
- **Funcionalidad:** Retorna el perfil completo del usuario autenticado con todas sus relaciones cargadas.

Método: updateProfile(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con datos a actualizar
- **Parámetros de salida:** *JsonResponse* con usuario actualizado
- **Funcionalidad:** Actualiza la información personal del usuario con validación de datos únicos.

Método: changePassword(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con contraseña actual y nueva
- **Parámetros de salida:** *JsonResponse* de confirmación
- **Funcionalidad:** Cambia la contraseña del usuario verificando la contraseña actual y encriptando la nueva.

5.1.2. Controlador de Gestión de TFT (TFTController)

El controlador principal de TFT implementa toda la lógica de negocio relacionada con el ciclo de vida de los TFT, desde la creación hasta la evaluación final.

Método: index(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con filtros opcionales
- **Parámetros de salida:** *JsonResponse* con lista paginada de TFT

- **Funcionalidad:** Lista los TFT según el rol del usuario, aplicando filtros por estado, convocatoria y relaciones cargadas.

Método: show(Tft $tft): JsonResponse

- **Parámetros de entrada:** Modelo **Tft**
- **Parámetros de salida:** *JsonResponse* con TFT completo
- **Funcionalidad:** Muestra un TFT específico con todas sus relaciones y verifica permisos de acceso mediante políticas.

Método: store(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con datos del TFT
- **Parámetros de salida:** *JsonResponse* con TFT creado
- **Funcionalidad:** Crea un nuevo TFT con validación de datos, asignación de estado inicial y relaciones con usuarios.

Método: update(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con datos a actualizar y modelo **Tft**
- **Parámetros de salida:** *JsonResponse* con TFT actualizado
- **Funcionalidad:** Actualiza un TFT existente verificando permisos y estado permitido para edición.

Método: enviarPropuesta(Tft $tft): JsonResponse

- **Parámetros de entrada:** Modelo **Tft**
- **Parámetros de salida:** *JsonResponse* de confirmación
- **Funcionalidad:** Envía la propuesta de TFT a la comisión para evaluación, cambiando el estado a propuesta_presentada.

Método: aprobarPropuesta(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* y modelo **Tft**
- **Parámetros de salida:** *JsonResponse* de confirmación

- **Funcionalidad:** Aprueba una propuesta de TFT por parte de la comisión, cambiando el estado a propuesta_aceptada.

Método: rechazarPropuesta(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con motivo y modelo **Tft**
- **Parámetros de salida:** *JsonResponse* de confirmación
- **Funcionalidad:** Rechaza una propuesta de TFT con motivo detallado y establece plazo de corrección.

Método: subirDocumento(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con archivo y modelo **Tft**
- **Parámetros de salida:** *JsonResponse* con documento creado
- **Funcionalidad:** Sube un documento asociado al TFT con validación de formato y almacenamiento seguro.

Método: firmarDocumento(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con tipo de firma y modelo *Tft*
- **Parámetros de salida:** *JsonResponse* de confirmación
- **Funcionalidad:** Registra la firma digital de un documento por parte de **estudiantes**, **tutores** o **co-tutores**.

Método: asignarTribunal(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con miembros del tribunal y modelo **Tft**
- **Parámetros de salida:** JsonResponse con tribunal asignado
- **Funcionalidad:** Asigna un **tribunal evaluador** al TFT verificando restricciones y disponibilidad de miembros.

Método: programarDefensa(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con fecha, hora y lugar de defensa
- **Parámetros de salida:** *JsonResponse* de confirmación

- **Funcionalidad:** Programa la defensa del TFT verificando disponibilidad y estableciendo el estado correspondiente.

Método: evaluarTFT(Request $request, Tft $tft): JsonResponse

- **Parámetros de entrada:** *Request* con calificación y comentarios
- **Parámetros de salida:** *JsonResponse* con evaluación registrada
- **Funcionalidad:** Registra la evaluación final del TFT por parte del tribunal con calificación y comentarios detallados.

5.1.3. Controlador de gestión de usuarios (UserController)

El controlador de usuarios proporciona funcionalidades completas para la administración de usuarios del sistema, incluyendo creación, edición y gestión de roles.

Método: index(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con filtros opcionales
- **Parámetros de salida:** *JsonResponse* con lista de usuarios
- **Funcionalidad:** Lista usuarios con filtros por rol, departamento y búsqueda textual, incluyendo relaciones cargadas.

Método: store(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con datos del usuario
- **Parámetros de salida:** *JsonResponse* con usuario creado
- **Funcionalidad:** Crea un nuevo usuario con validación completa de datos únicos y encriptación de contraseña.

Método: show(User $user): JsonResponse

- **Parámetros de entrada:** Modelo **User**
- **Parámetros de salida:** *JsonResponse* con usuario completo
- **Funcionalidad:** Muestra un usuario específico con todas sus relaciones cargadas.

Método: update(Request $request, User $user): JsonResponse

- **Parámetros de entrada:** *Request* con datos a actualizar y modelo **User**
- **Parámetros de salida:** *JsonResponse* con usuario actualizado
- **Funcionalidad:** Actualiza un usuario existente con validación de datos únicos y gestión opcional de contraseña.

Método: destroy(User $user): JsonResponse

- **Parámetros de entrada:** Modelo **User**
- **Parámetros de salida:** *JsonResponse* de confirmación
- **Funcionalidad:** Elimina un usuario del sistema con borrado en cascada de sus TFT asociados.

Método: tutoresDisponibles(): JsonResponse

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *JsonResponse* con lista de tutores
- **Funcionalidad:** Retorna todos los tutores disponibles con certificado firmado y departamento cargado.

Método: miembrosComision(): JsonResponse

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *JsonResponse* con lista de miembros
- **Funcionalidad:** Retorna todos los miembros de la comisión TFT con departamento cargado.

5.1.4. Controlador de gestión de tribunales (TribunalController)

El controlador de tribunales gestiona la creación, asignación y administración de los tribunales evaluadores de TFT.

Método: index(): JsonResponse

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *JsonResponse* con lista de tribunales
- **Funcionalidad:** Lista todos los tribunales con todos sus miembros y TFT asociados cargados.

Método: searchProfesores(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con *query* de búsqueda
- **Parámetros de salida:** *JsonResponse* con profesores encontrados
- **Funcionalidad:** Busca profesores por nombre para autocompletado en la asignación de tribunales.

Método: store(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con datos del tribunal
- **Parámetros de salida:** *JsonResponse* con tribunal creado
- **Funcionalidad:** Crea un nuevo tribunal con validación de miembros y restricciones reglamentarias.

Método: show(Tribunal $tribunal): JsonResponse

- **Parámetros de entrada:** Modelo **Tribunal**
- **Parámetros de salida:** *JsonResponse* con tribunal completo
- **Funcionalidad:** Muestra un tribunal específico con todos sus miembros y TFT asociados.

Método: update(Request $request, Tribunal $tribunal): JsonResponse

- **Parámetros de entrada:** *Request* con datos a actualizar y modelo **Tribunal**
- **Parámetros de salida:** *JsonResponse* con tribunal actualizado
- **Funcionalidad:** Actualiza un tribunal existente verificando restricciones y validaciones.

5.1.5. Controlador de gestión de calendario (CalendarioController)

El controlador de calendario gestiona los eventos académicos, convocatorias y plazos del sistema.

Método: index(): JsonResponse

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *JsonResponse* con lista de calendarios
- **Funcionalidad:** Lista todos los calendarios ordenados por fecha de inicio.

Método: store(Request $request): JsonResponse

- **Parámetros de entrada:** *Request* con datos del calendario
- **Parámetros de salida:** *JsonResponse* con calendario creado
- **Funcionalidad:** Crea un nuevo evento de calendario con validación de fechas y restricciones de rol.

Método: show(Calendario $calendario): JsonResponse

- **Parámetros de entrada:** Modelo **Calendario**
- **Parámetros de salida:** *JsonResponse* con calendario completo
- **Funcionalidad:** Muestra un evento de calendario específico.

Método: update(Request $request, Calendario $calendario): JsonResponse

- **Parámetros de entrada:** *Request* con datos a actualizar y modelo **Calendario**
- **Parámetros de salida:** *JsonResponse* con calendario actualizado
- **Funcionalidad:** Actualiza un evento de calendario existente.

Método: destroy(Calendario $calendario): JsonResponse

- **Parámetros de entrada:** Modelo **Calendario**
- **Parámetros de salida:** *JsonResponse* de confirmación

- **Funcionalidad:** Elimina un evento de calendario del sistema.

5.1.6. Controlador de chat (ChatController)

El controlador de chat implementa la funcionalidad de comunicación en tiempo real entre usuarios del sistema.

Método: getConversations()

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *JsonResponse* con conversaciones del usuario
- **Funcionalidad:** Obtiene todas las conversaciones del usuario autenticado con usuarios y último mensaje cargados.

Método: getMessages(Conversation $conversation)

- **Parámetros de entrada:** Modelo **Conversation**
- **Parámetros de salida:** *JsonResponse* con mensajes de la conversación
- **Funcionalidad:** Obtiene todos los mensajes de una conversación específica y marca como leídos.

Método: sendMessage(Request $request, Conversation $conversation)

- **Parámetros de entrada:** *Request* con contenido del mensaje y modelo **Conversation**
- **Parámetros de salida:** *JsonResponse* con mensaje creado
- **Funcionalidad:** Envía un mensaje a una conversación con soporte para archivos adjuntos y *broadcasting*.

Método: startConversation(Request $request)

- **Parámetros de entrada:** *Request* con participantes
- **Parámetros de salida:** *JsonResponse* con conversación creada o existente
- **Funcionalidad:** Inicia una nueva conversación o retorna una existente entre usuarios específicos.

5.1.7. Middleware de control de roles (CheckRole)

El *middleware* implementa el control de acceso basado en roles para proteger las rutas del sistema.

Método: handle(Request $request, Closure $next, ...$roles): Response

- **Parámetros de entrada:** *Request*, *Closure* y roles permitidos
- **Parámetros de salida:** *Response* o redirección
- **Funcionalidad:** Verifica que el usuario autenticado tenga uno de los roles permitidos, registra intentos de acceso y retorna respuesta apropiada.

5.1.8. Políticas de autorización

Las políticas implementan el control granular de acceso a recursos específicos del sistema.

TftPolicy:

- **Método view($user, $tft): bool** - Verifica si el usuario puede ver un TFT específico
- **Método update($user, $tft): bool** - Verifica si el usuario puede actualizar un TFT específico
- **Método delete($user, $tft): bool** - Verifica si el usuario puede eliminar un TFT específico

ConversationPolicy:

- **Método view($user, $conversation): bool** - Verifica si el usuario puede ver una conversación
- **Método delete($user, $conversation): bool** - Verifica si el usuario puede eliminar una conversación

5.2. Implementación del frontend (Angular 17)

5.2.1. Servicio de autenticación (AuthService)

El servicio de autenticación gestiona todo el proceso de *login*, *logout* y gestión de sesión en el *frontend*.

Método: login(dni: string, password: string): Observable<LoginResponse>

- **Parámetros de entrada:** DNI y contraseña del usuario
- **Parámetros de salida:** *Observable* con respuesta de *login*
- **Funcionalidad:** Realiza la petición de autenticación al *backend*, almacena el token y datos del usuario en *localStorage*, y actualiza el estado de autenticación.

Método: logout(): Observable<any>

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Observable* de confirmación
- **Funcionalidad:** Realiza *logout* en el *backend*, limpia *localStorage* y actualiza el estado de autenticación.

Método: getCurrentUser(): any

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** Datos del usuario actual
- **Funcionalidad:** Retorna los datos del usuario actualmente autenticado.

Método: isAuthenticated(): boolean

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Boolean* indicando si está autenticado
- **Funcionalidad:** Verifica si existe un usuario autenticado en el sistema.

Método: hasRole(roles: string | string[]): boolean

- **Parámetros de entrada:** Rol o *array* de roles a verificar

- **Parámetros de salida:** Boolean indicando si tiene el rol
- **Funcionalidad:** Verifica si el usuario actual tiene uno de los roles especificados.

Método: updateProfile(profileData: any): Observable<any>

- **Parámetros de entrada:** Datos del perfil a actualizar
- **Parámetros de salida:** *Observable* con usuario actualizado
- **Funcionalidad:** Actualiza el perfil del usuario y sincroniza los datos locales.

5.2.2. Servicio de Gestión de TFT (TftService)

El servicio de TFT proporciona todas las operaciones **CRUD** y funcionalidades específicas para la gestión de trabajos de fin de titulación.

Método: getTfts(params?: any): Observable<any>

- **Parámetros de entrada:** Parámetros de filtrado opcionales
- **Parámetros de salida:** *Observable* con lista de TFT
- **Funcionalidad:** Obtiene la lista de TFT con filtros aplicados según el rol del usuario.

Método: getTft(id: number): Observable<any>

- **Parámetros de entrada:** ID del TFT
- **Parámetros de salida:** *Observable* con TFT específico
- **Funcionalidad:** Obtiene un TFT específico con todas sus relaciones cargadas.

Método: createTft(tftData: any): Observable<any>

- **Parámetros de entrada:** Datos del TFT a crear
- **Parámetros de salida:** *Observable* con TFT creado
- **Funcionalidad:** Crea un nuevo TFT en el sistema.

Método: updateTft(id: number, tftData: any): Observable<any>

- **Parámetros de entrada:** *ID* del TFT y datos a actualizar

- **Parámetros de salida:** *Observable* con TFT actualizado
- **Funcionalidad:** Actualiza un TFT existente.

Método: deleteTft(id: number): Observable<any>

- **Parámetros de entrada:** *ID* del TFT
- **Parámetros de salida:** *Observable* de confirmación
- **Funcionalidad:** Elimina un TFT del sistema.

Método: aprobarPropuesta(tftId: number): Observable<any>

- **Parámetros de entrada:** *ID* del TFT
- **Parámetros de salida:** *Observable* de confirmación
- **Funcionalidad:** Aprueba una propuesta de TFT.

Método: rechazarPropuesta(tftId: number, motivo: string): Observable<any>

- **Parámetros de entrada:** *ID* del TFT y motivo de rechazo
- **Parámetros de salida:** *Observable* de confirmación
- **Funcionalidad:** Rechaza una propuesta de TFT con motivo específico.

Método: subirDocumento(tftId: number, file: File, tipo: string): Observable<any>

- **Parámetros de entrada:** *ID* del TFT, archivo y tipo de documento
- **Parámetros de salida:** *Observable* con documento subido
- **Funcionalidad:** Sube un documento asociado al TFT.

5.2.3. Componente de Login (LoginComponent)

El componente de *login* implementa la interfaz de autenticación con validación en tiempo real y manejo de errores.

Método: onSubmit(): void

- **Parámetros de entrada:** Ninguno

- **Parámetros de salida:** *Void*
- **Funcionalidad:** Maneja el envío del formulario de *login*, valida credenciales y redirige según el resultado.

Método: ngOnInit(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicializa el componente verificando si existe un token y mostrando mensaje de seguridad si es necesario.

5.2.4. Componente de Dashboard (DashboardComponent)

El componente de *dashboard* proporciona una vista personalizada según el rol del usuario con estadísticas y acciones rápidas.

Método: ngOnInit(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicializa el dashboard cargando estadísticas específicas según el rol del usuario.

Método: abrirReglamento(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Abre el reglamento oficial de TFT en una nueva ventana.

5.2.5. Componente de Lista de TFT (TftListComponent)

El componente de lista de TFT implementa la visualización y gestión de trabajos de fin de titulación.

Método: ngOnInit(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicializa el componente cargando la lista de TFT y datos necesarios para filtros.

Método: loadTfts(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Carga la lista de TFT aplicando filtros según el rol del usuario.

Método: viewTft(id: number): void

- **Parámetros de entrada:** ID del TFT
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Navega a la vista detallada de un TFT específico.

Método: createTft(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Crea un nuevo TFT utilizando los datos del formulario modal.

5.2.6 Componente de Detalle de TFT (TftDetailComponent)

El componente de detalle de TFT implementa la visualización completa y gestión de un TFT específico.

Método: ngOnInit(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicializa el componente cargando los datos del TFT y verificando permisos de acceso.

Método: loadTft(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Carga los datos completos del TFT con todas sus relaciones.

Método: aprobarPropuesta(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Aprueba la propuesta del TFT actual.

Método: rechazarPropuesta(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Rechaza la propuesta del TFT con motivo específico.

Método: programarDefensa(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Programa la defensa del TFT con fecha, hora y lugar específicos.

Método: subirDocumento(event: any): void

- **Parámetros de entrada:** Evento de selección de archivo
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Sube un documento asociado al TFT con validación de formato.

5.2.7 Componente de Gestión de Usuarios (UsuariosComponent)

El componente de gestión de usuarios implementa la administración completa de usuarios del sistema.

Método: ngOnInit(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicializa el componente cargando la lista de usuarios y configurando filtros.

Método: loadUsers(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Carga la lista de usuarios aplicando filtros configurados.

Método: filterUsers(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Aplica filtros de búsqueda, rol y departamento a la lista de usuarios.

Método: createUser(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Crea un nuevo usuario utilizando los datos del formulario modal.

Método: editUser(user: any): void

- **Parámetros de entrada:** Datos del usuario a editar
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Abre el modal de edición con los datos del usuario seleccionado.

Método: deleteUser(user: any): void

- **Parámetros de entrada:** Datos del usuario a eliminar

- **Parámetros de salida:** *Void*
- **Funcionalidad:** Elimina un usuario del sistema con confirmación previa.

5.2.8 Componente de Chat (ChatWindowComponent)

El componente de chat implementa la comunicación en tiempo real entre usuarios del sistema.

Método: ngOnInit(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicializa el componente cargando conversaciones y configurando *listeners* de mensajes.

Método: loadConversations(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Carga todas las conversaciones del usuario autenticado.

Método: selectConversation(conversation: any): void

- **Parámetros de entrada:** Datos de la conversación
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Selecciona una conversación y carga sus mensajes.

Método: sendMessage(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Envía un mensaje a la conversación actual con soporte para archivos adjuntos.

Método: startNewConversation(): void

- **Parámetros de entrada:** Ninguno
- **Parámetros de salida:** *Void*
- **Funcionalidad:** Inicia una nueva conversación con usuarios seleccionados.

5.2.9 Guards de Autenticación

Los *guards* implementan la protección de rutas basada en autenticación y roles específicos.

Método: authGuard(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean

- **Parámetros de entrada:** *Route* y *state* del router
- **Parámetros de salida:** *Boolean* indicando si puede acceder
- **Funcionalidad:** Verifica que el usuario esté autenticado antes de permitir acceso a la ruta.

Método: roleGuard(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean

- **Parámetros de entrada:** *Route* y *state* del router
- **Parámetros de salida:** *Boolean* indicando si puede acceder
- **Funcionalidad:** Verifica que el usuario tenga el rol específico requerido para la ruta.

5.2.10 Interceptor JWT

El *interceptor JWT* añade automáticamente el *token* de autenticación a todas las peticiones HTTP.

Método: intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>

- **Parámetros de entrada:** *Request* HTTP y *handler*
- **Parámetros de salida:** *Observable* con respuesta HTTP

- **Funcionalidad:** Intercepta todas las peticiones HTTP y añade el token JWT en el *header* de autorización.

5.3. Integración y comunicación

5.3.1. API RESTful

La comunicación entre *frontend* y *backend* se realiza mediante una *API RESTful* completa que implementa todos los estándares HTTP y proporciona respuestas consistentes en formato JSON. **En el ANEXO se podrán consultar todas las rutas API RESTful y su respectiva descripción.**

Como se puede ver en la Ilustración 51, las rutas de la API para cada operación se definen en el archivo base **backend/routes/api.php**

```
// Rutas RESTful completas para TFTs
Route::apiResource('tfts', TFTController::class);

// Rutas RESTful para usuarios
Route::apiResource('users', UserController::class);

// Rutas RESTful para calendarios
Route::apiResource('calendarios', CalendarioController::class);

// Rutas RESTful para tribunales
Route::apiResource('tribunales', TribunalController::class);
```

Ilustración 51: Rutas RESTful base según tipo de operación

Donde por ejemplo tenemos la implementación estándar HTTP. A continuación tenemos un ejemplo para la ruta base de TFTs como se puede ver en la Ilustración 52.

```
GET /api/tfts - Listar TFTs (200)
POST /api/tfts - Crear TFT (201)
GET /api/tfts/{id} - Obtener TFT específico (200)
PUT /api/tfts/{id} - Actualizar TFT (200)
DELETE /api/tfts/{id} - Eliminar TFT (200)
```

Ilustración 52: Implementación de estándares HTTP para la ruta de tfts

A continuación, en la Ilustración 53 podemos ver las respuestas JSON para un caso de éxito y otro de error.

```php
// Ejemplo de respuesta exitosa
return response()->json([
    'message' => 'Login exitoso',
    'user' => $user->load('rol', 'departamento'),
    'token' => $token,
]);

// Ejemplo de respuesta de error
return response()->json([
    'message' => 'Las credenciales proporcionadas son incorrectas.'
], 422);
```

Ilustración 53:Ejemplo de una respuesta JSON de error y de éxito

5.3.2. Autenticación JWT

El sistema implementa autenticación basada en *tokens* JWT que proporcionan seguridad robusta y escalabilidad, permitiendo la **gestión de sesiones sin estado** en el servidor.

En la Ilustración 54 podemos ver a modo de ejemplo el código usado en **backend/app/Http/Controllers/AuthController.php** para implementar JWT.

```php
public function login(Request $request): JsonResponse
{
    // Validación de credenciales
    $user = User::where('dni', $request->dni)->first();

    if (!Hash::check($request->password, $user->password)) {
        return response()->json([
            'message' => 'Las credenciales proporcionadas son incorrectas.'
        ], 422);
    }

    // Generación de token JWT usando Laravel Sanctum
    $token = $user->createToken('auth_token')->plainTextToken;

    return response()->json([
        'message' => 'Login exitoso',
        'user' => $user->load('rol', 'departamento'),
        'token' => $token,
    ]);
}
```

Ilustración 54: Código de implementación en el backend de autenticación JWT

5.3.3. WebSockets para Chat

La funcionalidad de chat implementa *WebSockets* para comunicación en tiempo real, proporcionando una experiencia de usuario fluida y eficiente.

En la Ilustración 55 podemos ver la configuración de los *WebSockets* para el chat en nuestra plataforma.

```
'default' => 'reverb',

'connections' => [
    'reverb' => [
        'driver' => 'reverb',
        'key' => env('REVERB_APP_KEY', 'abcdefg12345'),
        'secret' => env('REVERB_SECRET', 'a_secure_secret_string'),
        'app_id' => env('REVERB_APP_ID', '12345'),
        'options' => [
            'host' => env('REVERB_HOST', 'localhost'),
            'port' => env('REVERB_PORT', 6001),
            'scheme' => env('REVERB_SCHEME', 'http'),
            'useTLS' => false,
        ],
    ],
]
```

Ilustración 55: Configuración de los WebSockets para el chat

Luego en la Ilustración 56 se puede apreciar el código de implementación en **backend/app/Http/Controllers/Api/ChatController.php**

```
public function sendMessage(Request $request, Conversation $conversation)
{
    // Validación y creación del mensaje
    $message = $conversation->messages()->create([
        'user_id' => Auth::id(),
        'body' => $request->body,
        'file_path' => $filePath,
        'file_name' => $fileName,
        'file_type' => $fileType,
    ]);

    // Broadcasting en tiempo real usando WebSockets
    broadcast(new \App\Events\MessageSent($message))->toOthers();

    return response()->json($message, 201);
}
```

Ilustración 56: Código implementación de WebSockets en el backend

5.3.4. Validación de Datos

Tanto el *frontend* como el *backend* implementan validación completa de datos que garantiza la integridad y consistencia de la información en el sistema.

Solo a modo de ejemplo en la Ilustración 57 se puede apreciar cómo se implementa una validación de datos en el *backend*.

```php
public function store(Request $request): JsonResponse
{
    $validator = Validator::make($request->all(), [
        'titulo' => 'required|string|max:255',
        'descripcion' => 'required|string',
        'estudiante_id' => 'required|exists:users,id',
        'tutor_id' => 'required|exists:users,id',
        'cotutor_id' => 'nullable|exists:users,id',
    ]);

    if ($validator->fails()) {
        return response()->json(['errors' => $validator->errors()], 422);
    }
}
```

Ilustración 57: Ejemplo de código de validación de datos en el backend

5.3.5 Manejo de Errores

El sistema implementa un manejo de errores robusto que proporciona mensajes informativos al usuario y registra errores para análisis y depuración.

A modo explicativo se puede ver en la Ilustración 58 un ejemplo de código que implementa gestión de errores para mostrar simplemente como se ha trabajado en este proyecto con dicho sistema de gestión de errores.

```php
public function login(Request $request): JsonResponse
{
    // Buscar usuario por DNI
    $user = User::where('dni', $request->dni)->first();

    if (!$user) {
        Log::warning('User not found with DNI: ' . $request->dni);
        return response()->json([
            'message' => 'Las credenciales proporcionadas son incorrectas.'
        ], 422);
    }

    // Verificar si el usuario está activo
    if (!$user->active) {
        Log::warning('Login failed for inactive user DNI: ' . $request->dni);
        return response()->json(['message' => 'El usuario está inactivo.'], 403)
    }
}
```

Ilustración 58: Ejemplo de implementación de gestión y manejo de errores

Con ello además buscamos con el fin de posibilitar las auditorias de seguridad y acceso que nos aparezcan en los *logs* de Laravel. Se puede ver en la Ilustración 59 un ejemplo de *log* de inicio de sesión exitoso.

```
Log::info('--- LOGIN ATTEMPT START ---');
Log::info('Request data:', $request->all());
Log::info('Login successful for user: ' . $user->name);
Log::info('--- LOGIN ATTEMPT SUCCESS ---');
```

Ilustración 59: Ejemplo de log de inicio de sesión exitoso en los logs de Laravel

Esta implementación completa demuestra la robustez y escalabilidad de la plataforma EITE TFT, proporcionando una base sólida para la gestión académica de TFT con todas las funcionalidades requeridas por el reglamento académico.

5.4. Implementación de los tests

La estrategia de *testing* se centró en dos tipos de validaciones. Primero, las validaciones de datos, donde se comprueba que los campos obligatorios estén presentes, que los formatos (**email, fecha, DNI**) sean correctos y que se devuelvan los errores 422 correspondientes cuando la información es inválida.

Por otro lado, se implementaron validaciones de la lógica de negocio. Estas pruebas verifican las reglas del sistema, como la coherencia de las fechas, las transiciones de estado de los TFT, los permisos asociados a cada rol y la unicidad de datos clave como el **DNI**.

Además, los tests fueron diseñados e implementados con el fin de manejar errores como pueden ser los errores de autenticación con usuarios no autenticados, tokens expirados o inválidos y usuarios inactivos. Adicionalmente tenemos tests de errores de autorización que abarcan casos de acceso denegado a recursos, operaciones no permitidas por rol y violaciones de políticas de seguridad.

Finalmente nos encontramos con tests de errores de validación para comprobar los datos malformados, campos requeridos faltantes y formatos inválidos.

Como se estableció en el capítulo 4 de diseño, el sistema se ha desarrollado para cubrir los casos de uso más importantes y frecuentes. Buscamos entonces que los tests abarquen la mayor cobertura de código posible.

En el ANEXO podremos ver la implementación de los tests.

5.4.1. Estructura de los tests del backend

La plataforma implementa una arquitectura de *testing* en el *backend* completa organizada en dos niveles principales:

Tests de Feature (Feature Tests):

- **Ubicación**: **backend/tests/Feature/**
- **Propósito**: Prueban funcionalidades completas del sistema desde la perspectiva del usuario
- **Cobertura***: Endpoints API*, flujos de trabajo, integración entre componentes.
- **Framework**: **PHPUnit** con **Laravel Testing Framework**

Tests Unitarios (Unit Tests):

- **Ubicación: backend/tests/Unit/**
- **Propósito**: Prueban lógica de negocio específica y políticas de autorización
- **Cobertura**: Modelos, servicios, políticas, validaciones
- **Framework**: **PHPUnit** con **Laravel Testing Framework**

5.4.2. Estructura de los tests en el frontend

Tests de Servicios (Service Tests):

- **Ubicación**: **frontend/src/app/core/services/*.spec.ts**
- **Propósito**: Prueban la lógica de servicios, comunicación HTTP y manejo de datos
- **Framework**: **Jasmine/Karma** con **Angular Testing Utilities**

Tests End-to-End (E2E Tests):

- **Ubicación**: **frontend/e2e/**
- **Propósito**: Prueban flujos completos de usuario desde la interfaz
- **Framework**: **Playwright** para automatización de navegador

6. Guía de usuario

En este capítulo vamos a abordar la explicación sobre cómo utilizar esta plataforma web además de las funciones a las que puede acceder el usuario en función de su rol que puede ser por: **administrador root, director EITE, secretario EITE, comisión TFT, Tutor, Miembro tribunal y estudiante**. Se tratará en todo momento de que el lenguaje sea lo más entendible posible para el público general debido a que está orientado al usuario evitando el uso siempre que sea posible de tecnicismos.

Además, esta guía de usuario será escrita siguiendo el flujo inicial antes de comenzar el curso académico pasando por todos los procesos y todos los usuarios hasta el final.

Comenzaremos primero con la pantalla de inicio de sesión que será la misma para todos los usuarios e iremos analizando el flujo académico de TFT.

Ilustración 60: Guía de usuario

(Imagen generada por Gemini)

Pantalla de Inicio de Sesión

La pantalla de inicio de sesión es el punto de entrada principal a la Plataforma TFT de la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria (ULPGC). Esta interfaz ha sido diseñada con un enfoque en la simplicidad y la seguridad, proporcionando un acceso rápido y confiable al sistema de gestión de Trabajos de Fin de Título. Se puede ver los componentes de la pantalla de Inicio de sesión en la Ilustración 61.

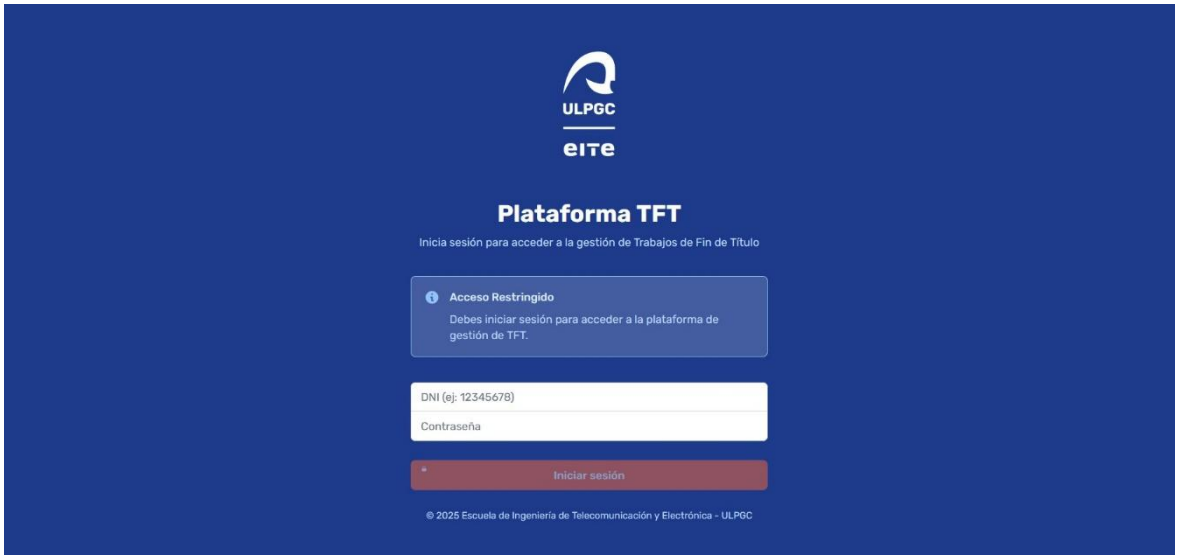

Ilustración 61: Pantalla de inicio de sesión

Una breve descripción en texto azul claro explica el propósito de la plataforma: *Inicia sesión para acceder a la gestión de Trabajos de Fin de Título*. Esta descripción ayuda a los usuarios a confirmar que están en el lugar correcto y entienden qué pueden hacer una vez autenticados.

Campo DNI

El primer campo del formulario solicita el DNI (Documento Nacional de Identidad) del usuario. Este campo incluye varias características de validación:

- Acepta únicamente 8 dígitos numéricos (formato: 12345678).
- Incluye validación en tiempo real que verifica el formato correcto
- Muestra mensajes de error específicos si el formato no es válido
- El campo se resalta en rojo cuando hay errores de validación
- Incluye un *placeholder* que muestra un ejemplo del formato esperado

Campo contraseña

El segundo campo solicita la contraseña del usuario. Este campo:

- Oculta automáticamente los caracteres ingresados por seguridad
- Es obligatorio para completar el proceso de autenticación
- No incluye validación de formato específico, permitiendo diversos tipos de contraseñas

Mensaje de acceso restringido

Cuando un usuario intenta acceder a la plataforma sin estar autenticado, aparece un mensaje informativo en un recuadro azul que explica que el acceso está restringido y que es necesario iniciar sesión. Este mensaje incluye un icono de información y texto explicativo sobre la necesidad de autenticación. Se puede ver en la Ilustración 62.

Ilustración 62:Mensaje de acceso restringido para usuarios no autenticados

Mensajes de error

En caso de problemas durante el inicio de sesión, el sistema muestra mensajes de error claros y específicos en un recuadro rojo con icono de advertencia. Estos errores pueden darse si el usuario no existe en la base de datos o si se introduce el campo DNI mal:

- Errores de formato del DNI. Visible en la Ilustración 63.
- Errores de credenciales de autenticación. Visible en la Ilustración 64.

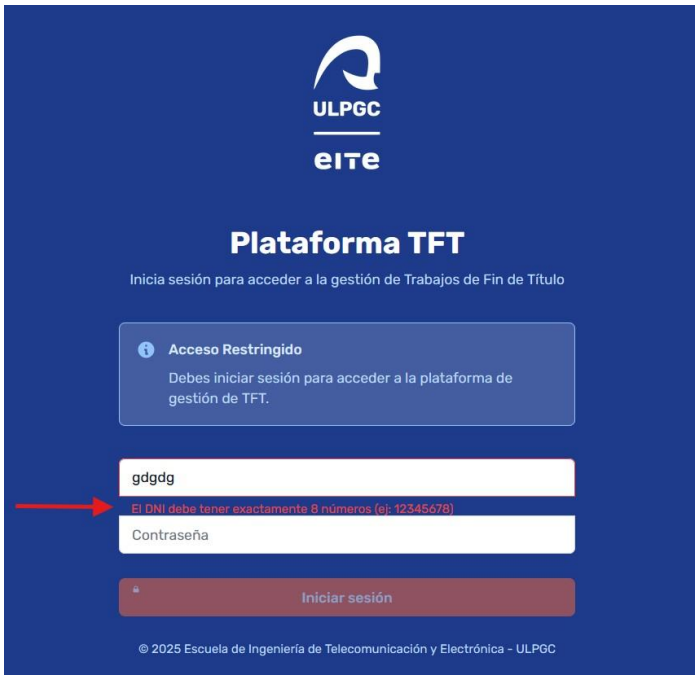

Ilustración 63: Mensaje de error de formato de DNI

Ilustración 64: Mensaje de error de credenciales de autenticación

Botón de inicio de sesión

El botón principal *Iniciar sesión* presenta las siguientes características:

- Se encuentra centrado y ocupa todo el ancho disponible
- Utiliza el color naranja institucional de la ULPGC
- Incluye un icono de candado que refuerza la idea de seguridad
- Durante el proceso de carga, el texto cambia a *Iniciando sesión...* y se muestra un indicador de carga animado. Podemos verlo en la Ilustración 65.
- Incluye efectos visuales de *hover* y *focus* para mejorar la experiencia del usuario
- Se deshabilita automáticamente cuando:
    - El formulario no es válido
    - Hay errores de validación del DNI
    - El proceso de autenticación está en curso

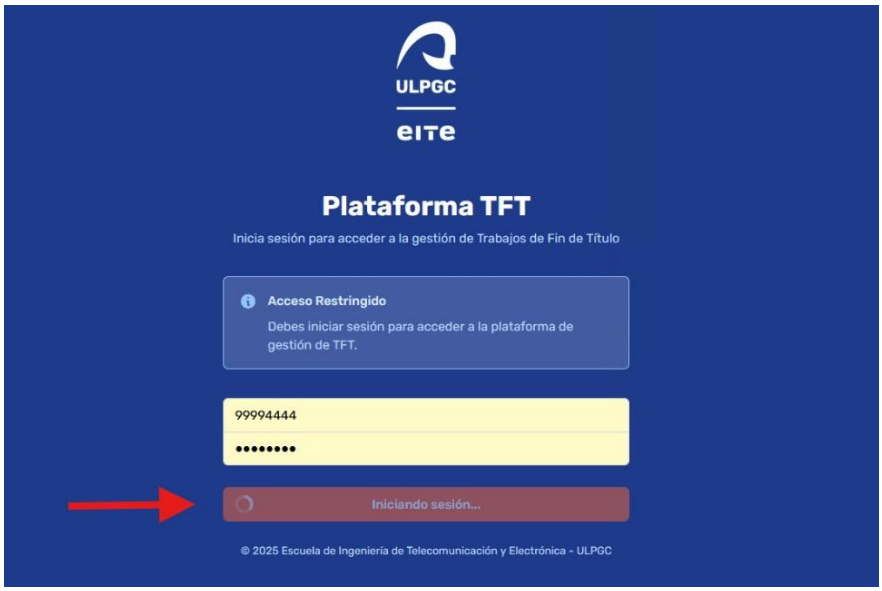

Ilustración 65: Indiador de carga

Flujo académico

Creación de nuevos usuarios

Una vez se han hecho las matrículas por parte de los estudiantes y se sabe quién va a cursar la asignatura de TFT. El **administrador root** en primera instancia será quien creará los usuarios manualmente en la pantalla de gestión de usuario que podemos ver en la Ilustración 66.

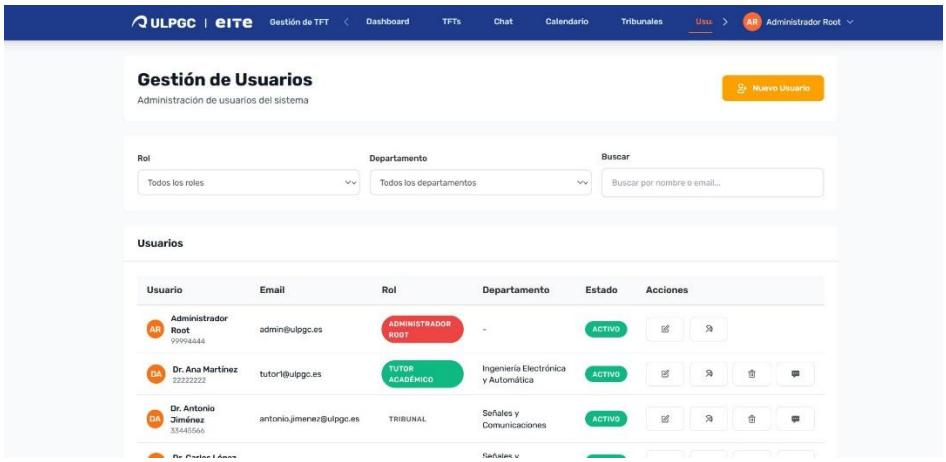

Ilustración 66: Pantalla de gestión de usuarios

Al darle al botón superior derecho que dice *Nuevo usuario* nos aparecerá un formulario para rellenar al con los datos del usuario y abajo del todo de dicho formulario nos cambiará a **VALID** cuando todo el formato de datos sea correcto y podamos crear al nuevo usuario, lo podemos ver en la Ilustración 67. Para simular una tarea habitual de inicio de curso, como es la de registrar nuevos estudiantes, a continuación se procederá a crear un nuevo perfil. Entonces en dicho formulario rellenamos los datos del estudiante.

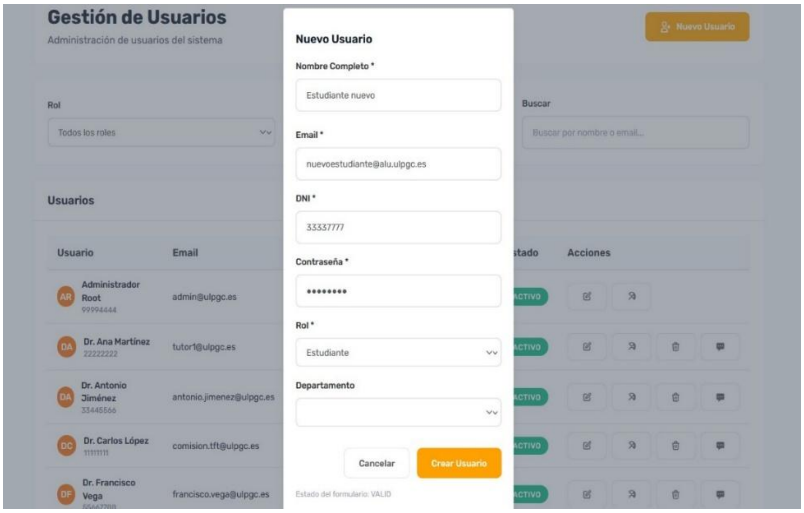

Ilustración 67: Formulario de creación de nuevo usuario

Una vez confirmado y enviado el formulario volveremos a la pantalla de gestión de usuarios y nos saldrá una notificación confirmándonos que todo ha salido correctamente y el usuario se encuentra ya registrado en la base de datos. Podemos ver esto en la Ilustración 68.

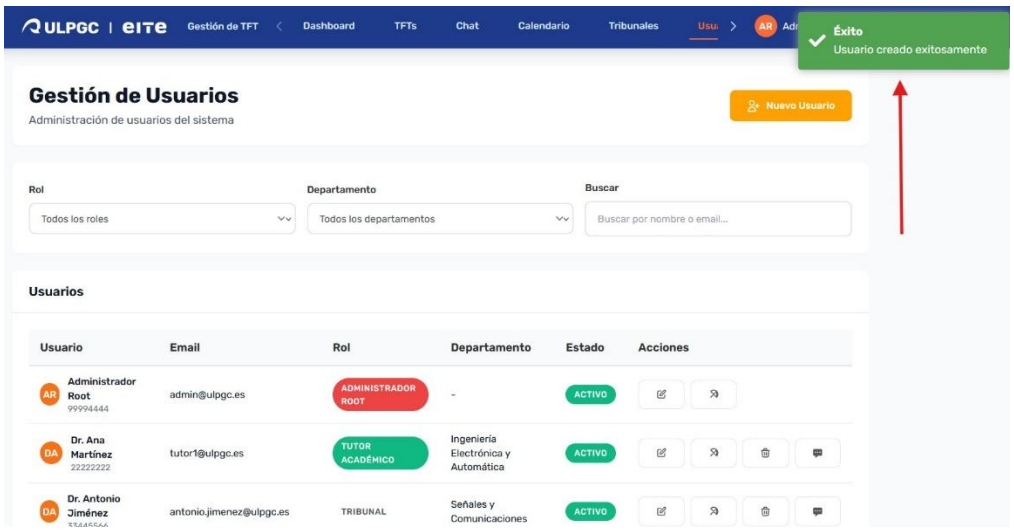

Ilustración 68: Confirmación de usuario creado correctamente

Creación de convocatorias

Una vez se han añadido los estudiantes y otros perfiles de usuario al sistema, el siguiente paso es crear las nuevas convocatorias. Esto se puede hacer desde la pantalla de gestión de convocatorias. La Ilustración 69 muestra una convocatoria que se ha creado a modo de ejemplo.

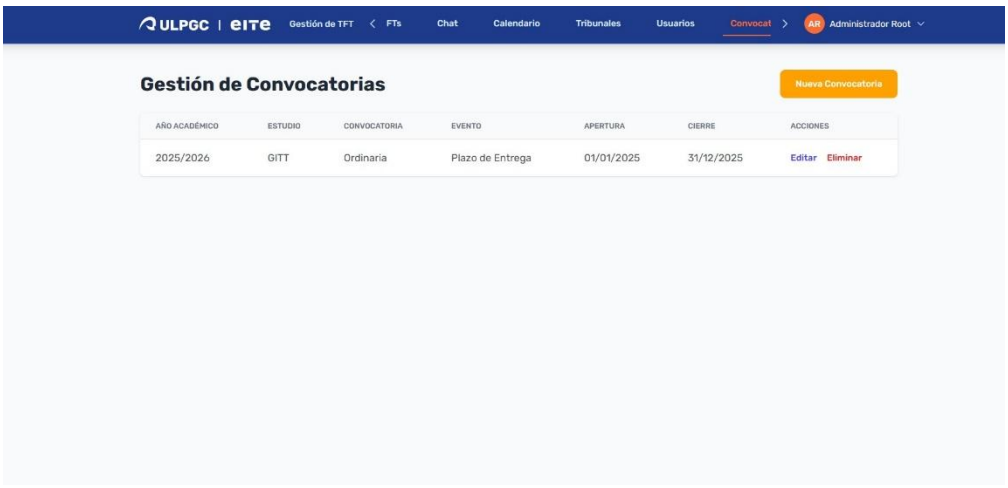

Ilustración 69: Pantalla de gestión de convocatorias

Al pulsar sobre el botón de *Nueva convocatoria* nos saldrá un formulario que nos permitirá en primer lugar poner el año académico de la convocatoria. Una característica que tiene nuestro formulario en cuanto a este campo es que al escribir el primer año se autocompleta con el segundo. Es decir, si se escribe 2024, automáticamente se autocompletará con 2025. Tras eso podremos añadir el tipo de estudio. Como estamos en la EITE, solo aparecerá el GITT, MUIT, GIB y DB-GITT-ADE. La Ilustración 70 muestra la vista general del formulario de creación de convocatorias.

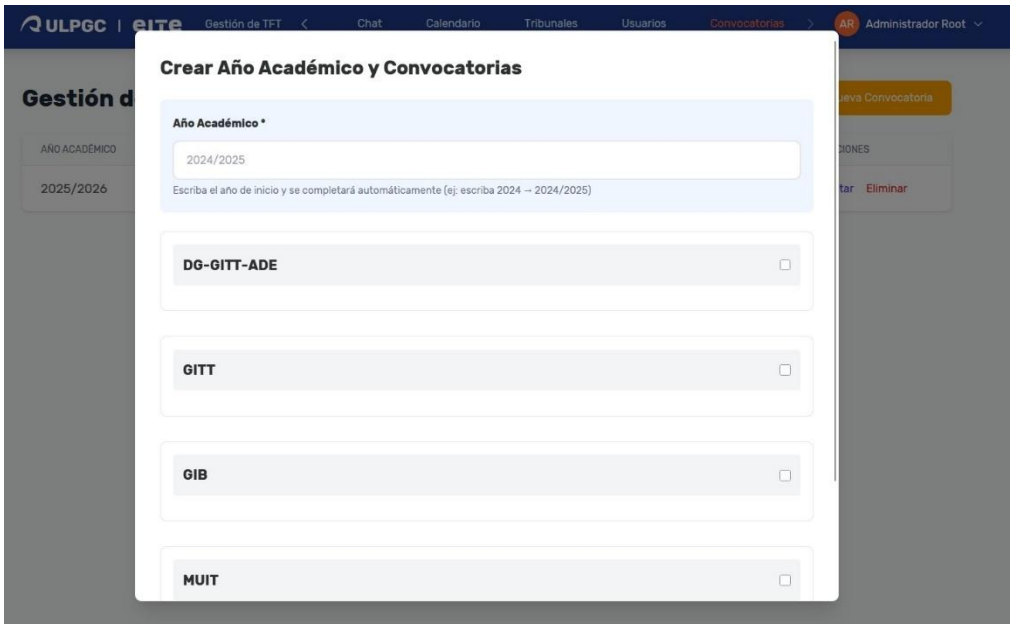

Ilustración 70: Formulario de creación de nuevo año y convocatoria

A modo de ejemplo, y para simular una tarea habitual de inicio de curso, se creará una nueva convocatoria para el año académico 2025/2026. En la Ilustración 71 podemos ver el nuevo formulario dentro del actual que nos sale para añadir las fechas importantes de la convocatoria de la EITE. El formulario se ha diseñado para ser muy fácil de usar. Por ejemplo, permite establecer fechas importantes de forma independiente para cada convocatoria de cada estudio. En el menú desplegable salen las fechas importantes de la EITE para ser puestas por el usuario, como puede haber solo una de esas fechas por convocatoria, al elegirla se elimina del desplegable para ayudar al usuario a identificar más fácilmente las demás fechas.

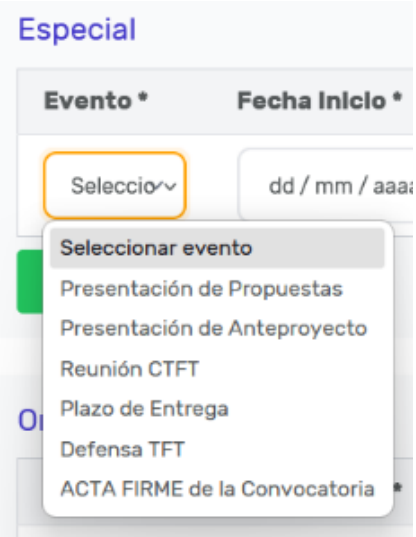

Ilustración 71: Menú desplegable de eventos de formulario de creación de convocatorias

Vamos a establecer el evento de **Presentación de propuestas** con su intervalo de fechas y le damos *click* al botón verde de *+Añadir evento*. En la Ilustración 72 se puede ver el calendario de selección de fechas para elegir una de las fechas clave del evento que es la de fin.

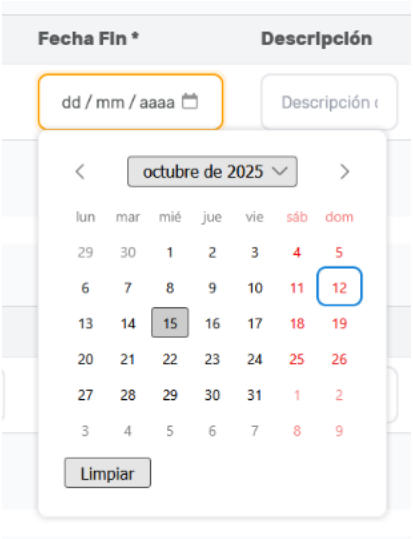

Ilustración 72: Calendario del formulario de creación de eventos para la selección de fechas

Tras eso y confirmar la creación del evento nos aparecerá en la lista siendo editable como se puede ver en la Ilustración 73.

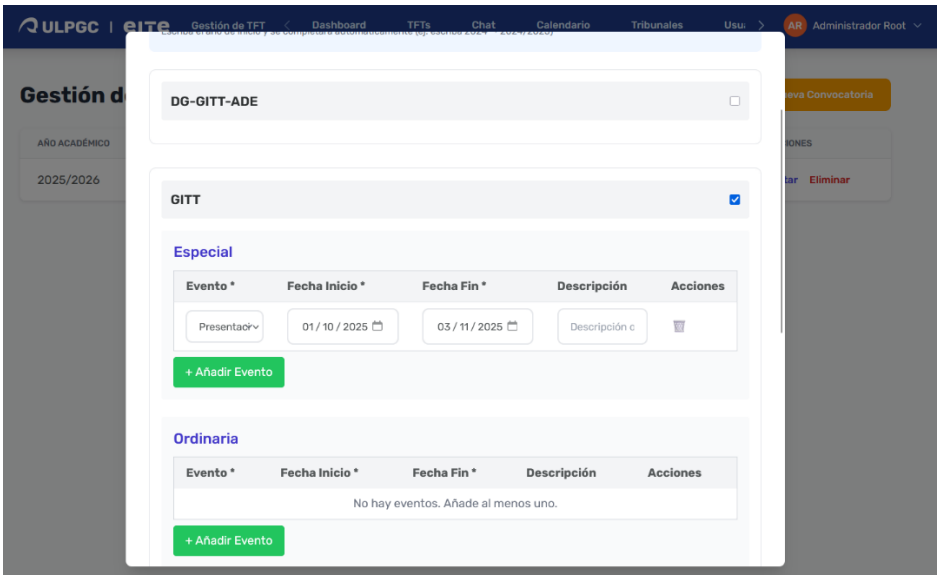

Ilustración 73: Evento ya añadido en el formulario de creación de convocatorias

Para guardar la configuración, se pulsa el botón *Guardar año académico completo* al final del formulario. Posteriormente, este **año académico** se puede modificar en cualquier momento para añadir, eliminar o cambiar eventos, convocatorias y estudios, tal y como se muestra en la Ilustración 74.

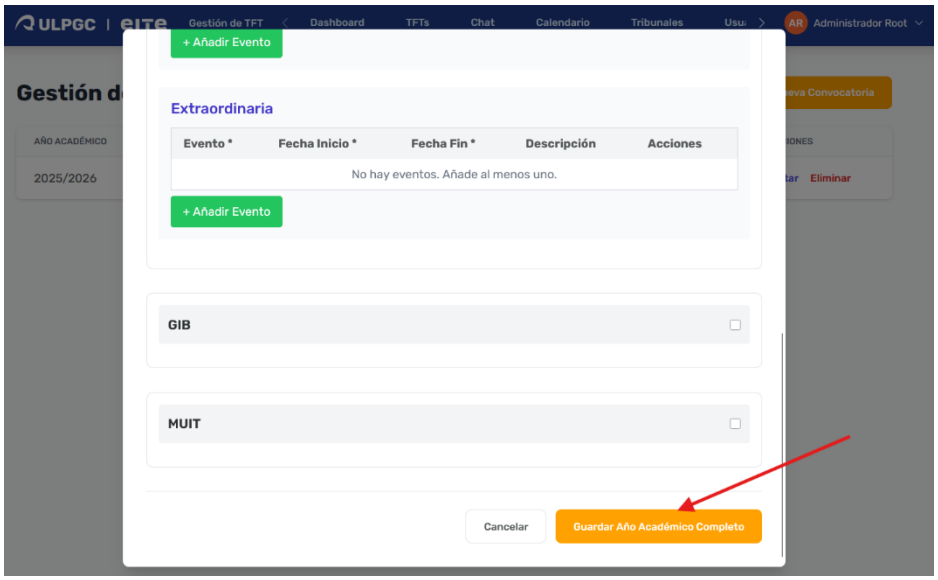

Ilustración 74: Botón para confirmar el guardado de año académico

Podemos ver la convocatoria con su evento en la pantalla de gestión de convocatorias. Se puede editar en cualquier momento. Podemos ver además el mensaje de que se ha añadido con éxito en la Ilustración 75.

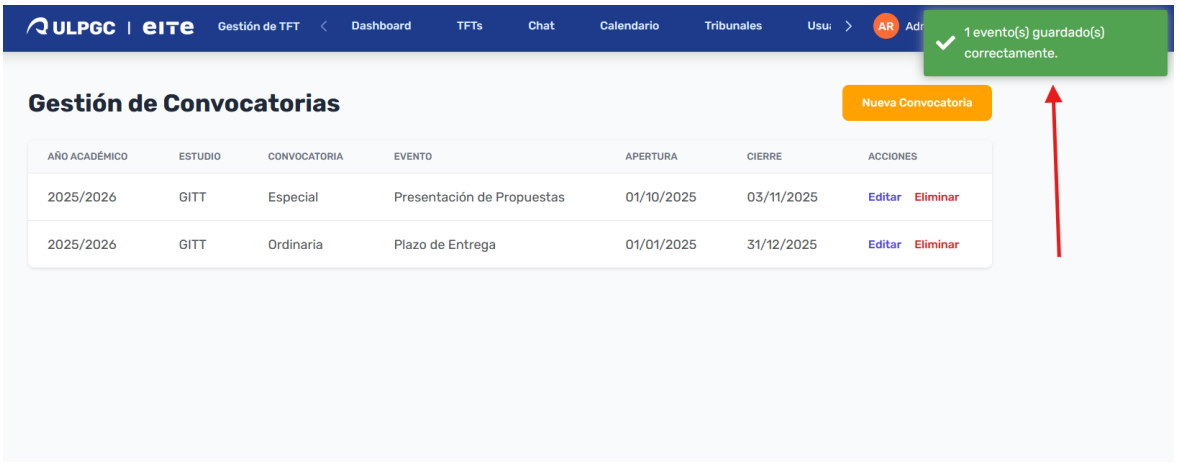

Ilustración 75: Convocatoria añadida correctamente

Creación de los TFT

Una vez añadidos los usuarios y las convocatorias con sus correspondientes eventos se pueden crear los TFT para cada alumno. Aun como **administrador root** vamos al apartado de gestión de TFT. Podemos ver dicha pantalla en la Ilustración 76 con TFGs que ya habían sido creados previamente.

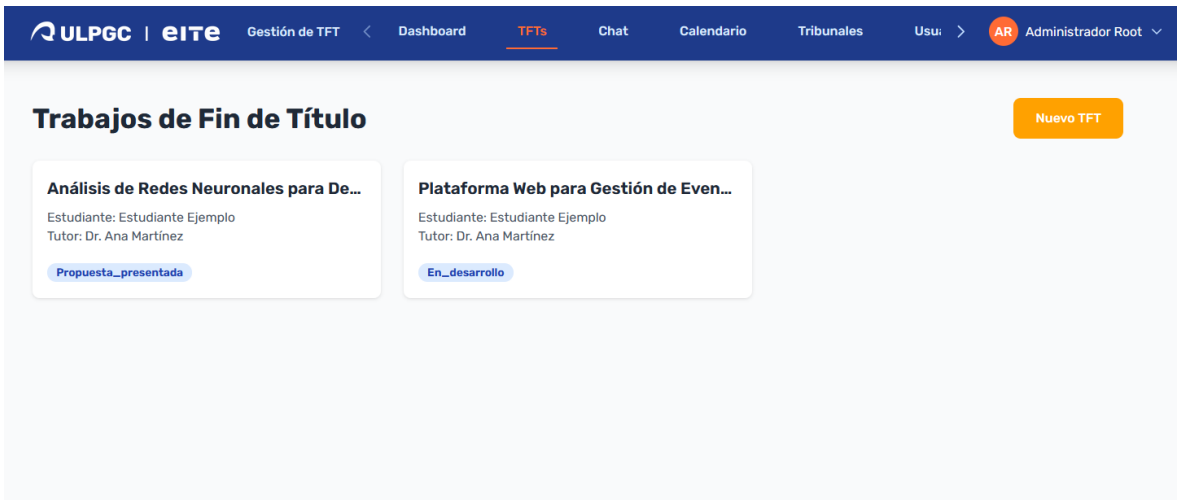

Ilustración 76: Pantalla de TFTs con ejemplos previamente creados

Al hacer *click* sobre el botón naranja de *Nuevo TFT* nos saldrá el correspondiente formulario para crearlo. Podemos verlo en la Ilustración 77.

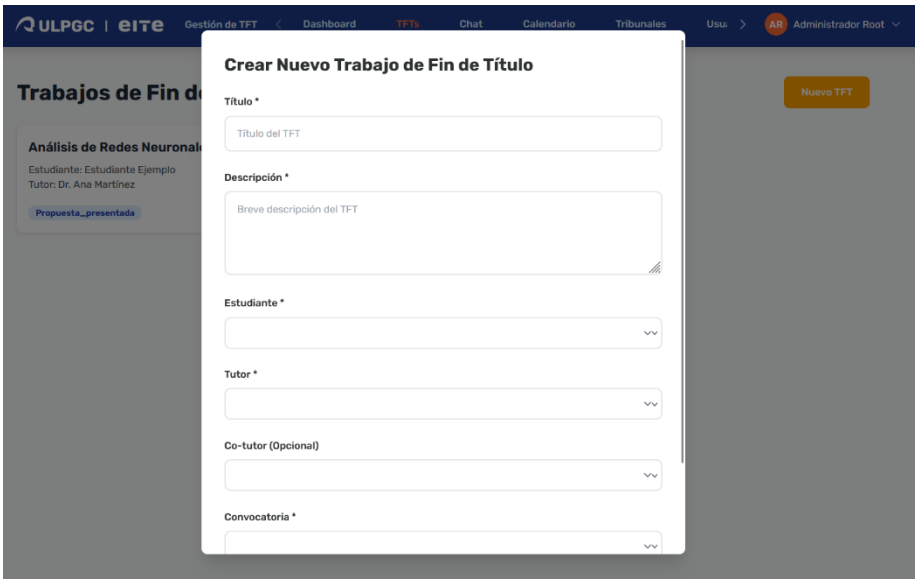

Ilustración 77: Formulario para crear un nuevo TFT

Esta interfaz permite crear un nuevo TFT, vinculándolo con un **estudiante**, un **tutor** y la **convocatoria** a la que pertenece. Si bien existe la opción de añadir un **co-tutor**, esta funcionalidad no se utilizará en la versión actual. Una vez relleno correctamente se seleccionará la opción de confirmar la creación de TFT como podemos ver en la Ilustración 78.

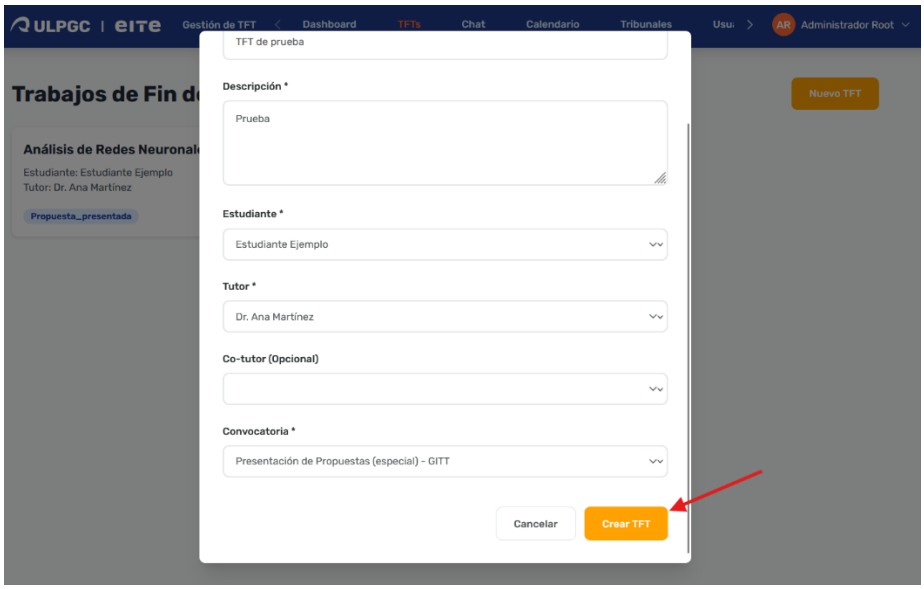

Ilustración 78: Confirmación de creación de TFT

Una vez confirmado nos saldrá un mensaje de confirmación y el TFG aparecerá en la lista de todos los TFG existentes de la aplicación encontrándose en el estado de **anteproyecto** como podemos ver en la Ilustración 79.

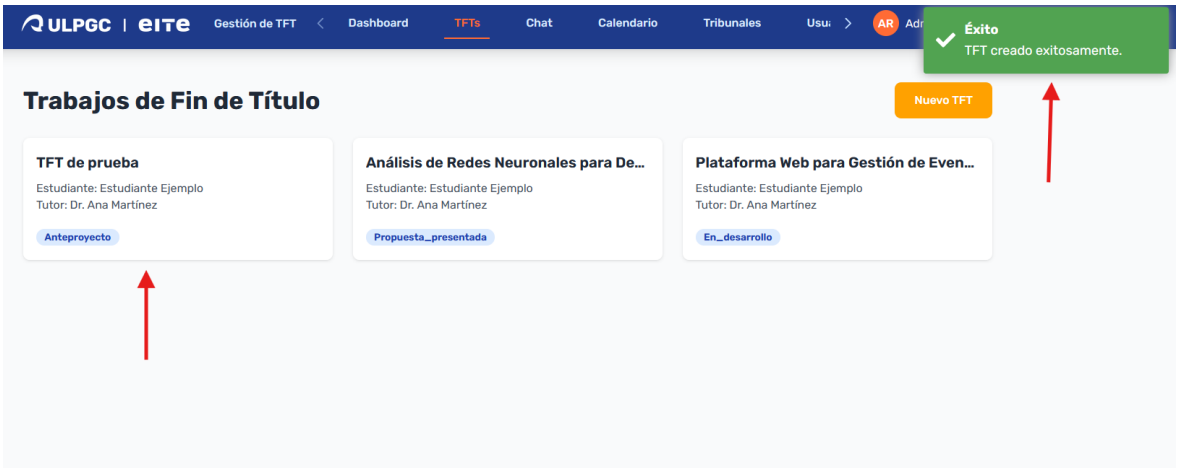

Ilustración 79: TFT creado correctamente

Para que un **estudiante** pueda presentar su **propuesta de TFT**, primero deben cumplirse una serie de condiciones: los usuarios implicados deben estar registrados en el sistema, debe existir una convocatoria activa y la fecha actual debe estar dentro del plazo establecido para ello. Una vez se cumplen estos requisitos, el **estudiante**, al acceder a su panel, podrá ver la información relativa a su TFT y el estado actual del proceso. En la Ilustración 80 podemos ver una vista general de la pantalla del TFT.

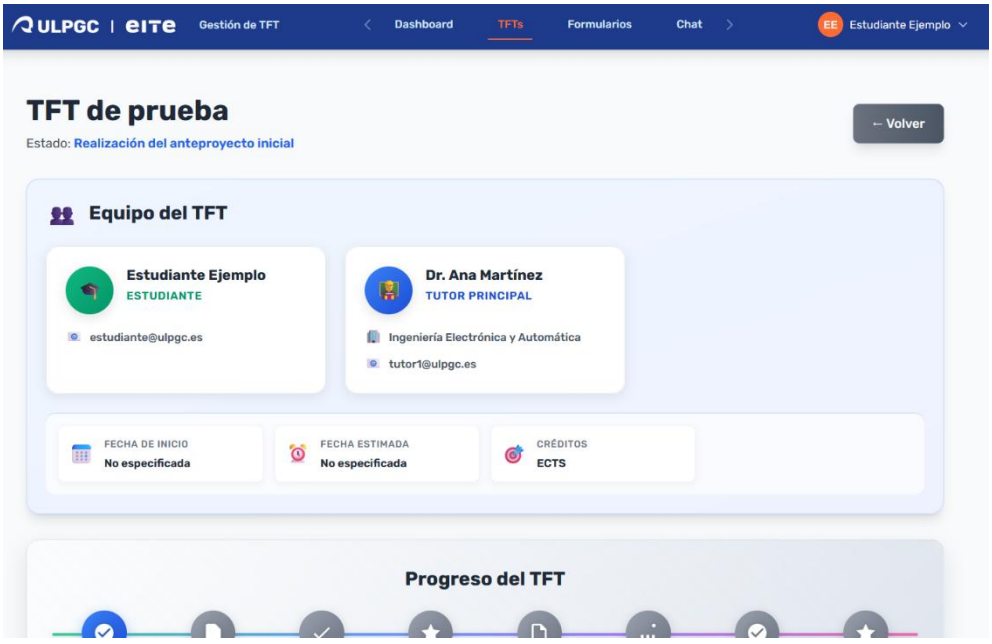

Ilustración 80: Vista general de la pantalla de detalles del TFT

La interfaz incluye un indicador de estado visual que utiliza iconos específicos para representar cada fase del proceso de desarrollo del TFT. Podemos verla en la Ilustración 81 en detalle.

Ilustración 81: Barra de progresión del estado del TFT

Como estudiante si bajamos en dicha página podemos ver que ya se encuentra habilitada la plataforma de carga de documentación para la documentación del TFT separada una para el formulario de solicitud de anteproyecto y por el otro lado la documentación del anteproyecto. La Ilustración 82 muestra en detalle la pantalla del TFT. En ella se observa la sección para la entrega de documentación y, a su lado, un mensaje informativo que guía al usuario indicando el estado actual del proyecto.

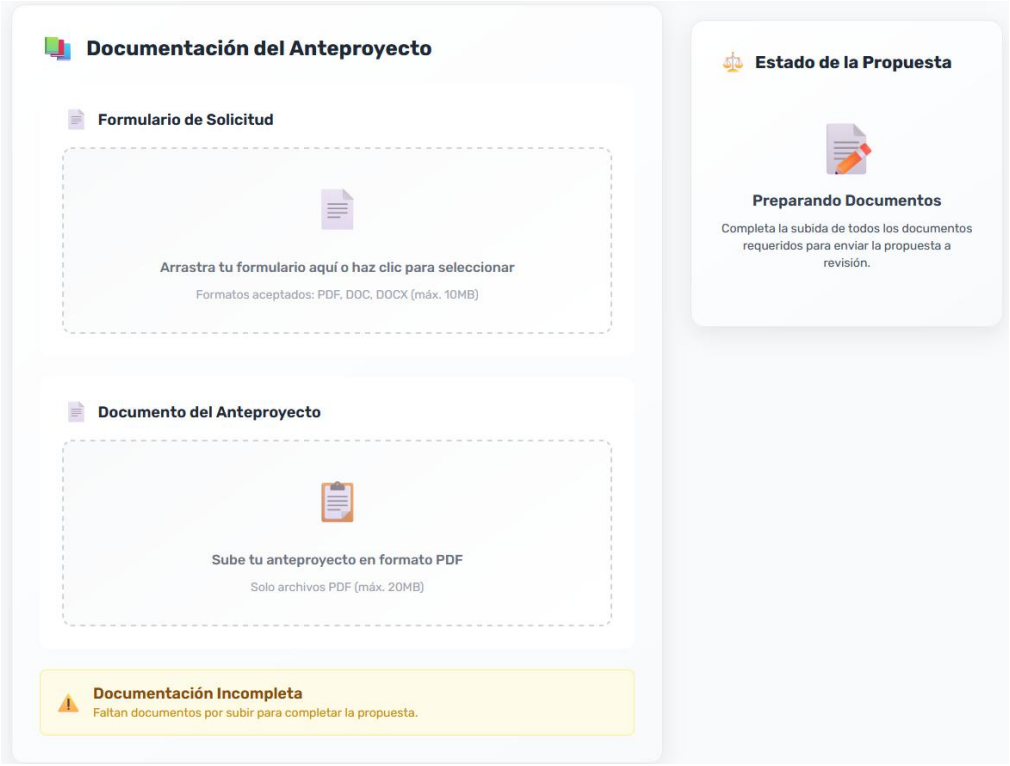

Ilustración 82: Plataforma de recepción de documentación de anteproyectos y mensaje informativo a modo de guía

Se puede subir la documentación correspondiente de manera sencilla arrastrando el archivo al módulo correspondiente o haciendo *click* en él, con ello se sube automáticamente. Una vez subidos ambos documentos el estado del proyecto cambiará a **PROPUESTA PRESENTADA**. Podemos ver esto en la Ilustración 83.

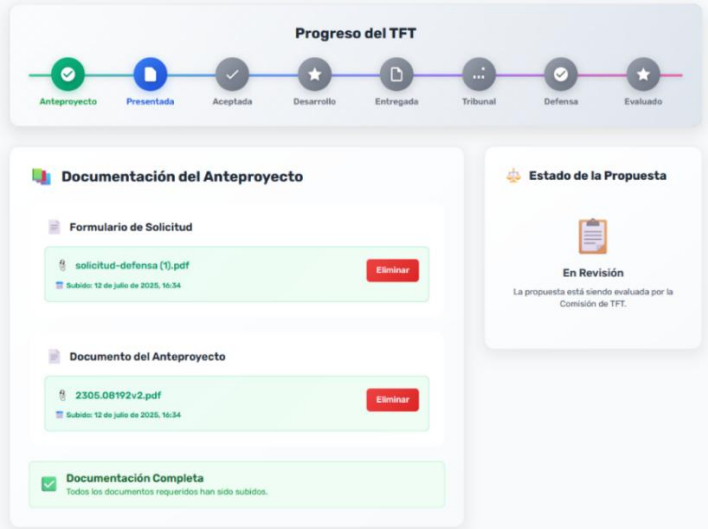

Ilustración 83: Plataforma de carga de documentación de anteproyecto con la documentación subida

Tras enviar la documentación, el usuario puede ver los detalles de la entrega, como el nombre del archivo y la fecha y hora exactas de la subida. El sistema también le ofrece la opción de eliminar el documento para reemplazarlo por uno nuevo. Asimismo, el mensaje de estado se actualiza para confirmar la recepción del archivo.

En el flujo académico ahora le toca el turno a la comisión de TFT que podrá ver dicha documentación subida para examinarla y aceptar o rechazar dicha propuesta. Al acceder a la pantalla de detalle de un TFT con el perfil de miembro de la comisión, el usuario dispone de dos funcionalidades principales: puede consultar el documento directamente en la plataforma y tiene acceso al panel para aprobar o rechazar la propuesta. Podemos ver esto último en la Ilustración 84.

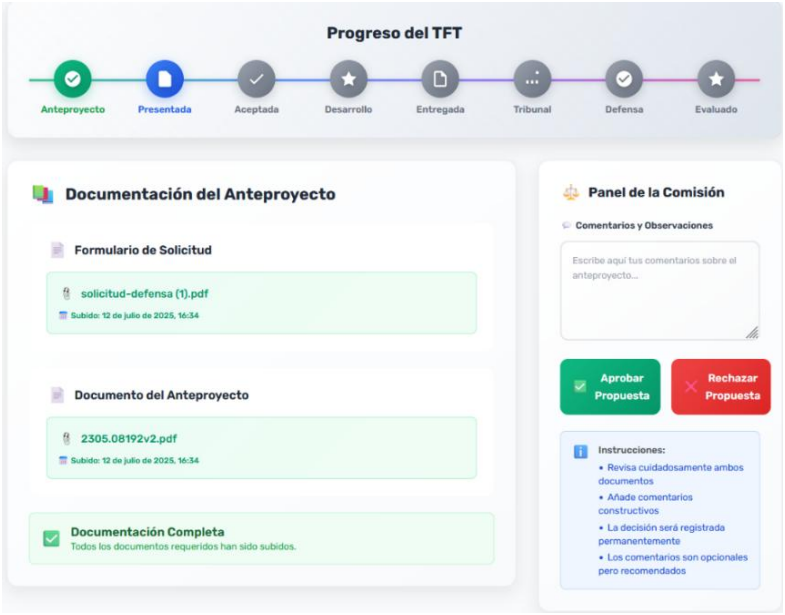

Ilustración 84: Panel de aceptación o rechazo de anteproyectos

La comisión aceptará dicha propuesta, por lo que se pulsará el botón de *Aprobar propuesta* entonces nos aparecerá el mensaje que se puede ver en la Ilustración 85.

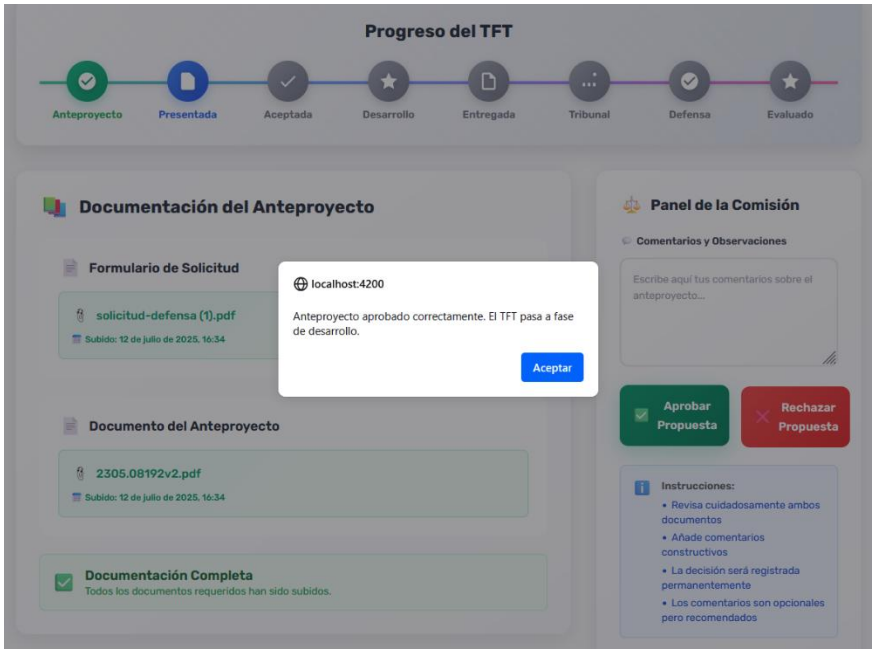

Ilustración 85: Mensaje de confirmación de anteproyecto aprobado

Ya con el anteproyecto aprobado el estudiante podrá trabajar en el desarrollo de su TFT pasando al estado en **DESARROLLO**. Cuando el TFT alcanza este estado, aparece una nueva sección para la entrega final, como se muestra en la Ilustración 86. En esta sección, el estudiante deberá subir toda la documentación de su proyecto en un único archivo .zip, siguiendo el procedimiento actual de la EITE.

Ilustración 86: Entregable de documentación de TFT

De forma similar a la entrega del anteproyecto, el estudiante deberá enviar la documentación final en un archivo comprimido, ya que es el único formato que admite la plataforma. Una vez realizada la entrega, el estado del TFT cambiará automáticamente a **ENTREGADA**. En la Ilustración 87 podemos ver el mensaje de confirmación de documentación de TFT subida.

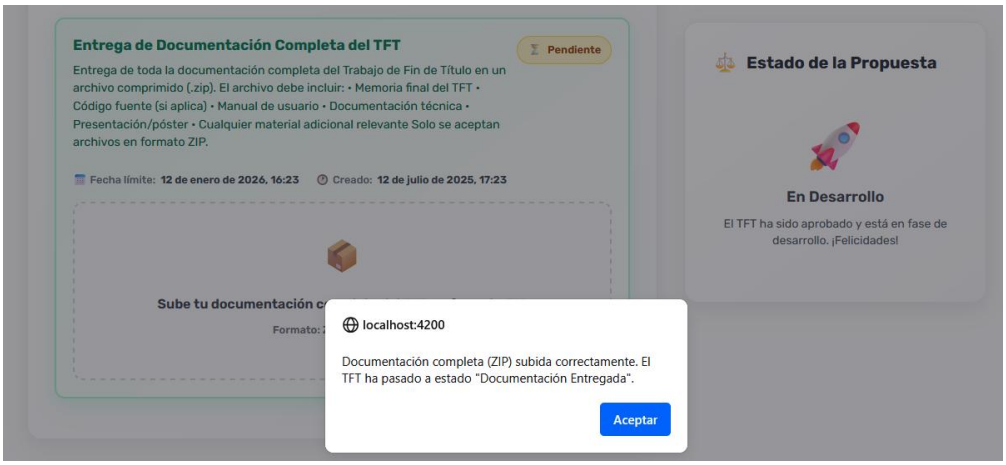

Ilustración 87: Mensaje de confirmación de documentación de TFT enviado

Una vez entregada la documentación, se asignará un tribunal evaluador al trabajo, tal como establece el reglamento de TFT de la EITE. Volviendo al **administrador root** procederemos a crear un tribunal en la pantalla de gestión de TFT en la Ilustración 88.

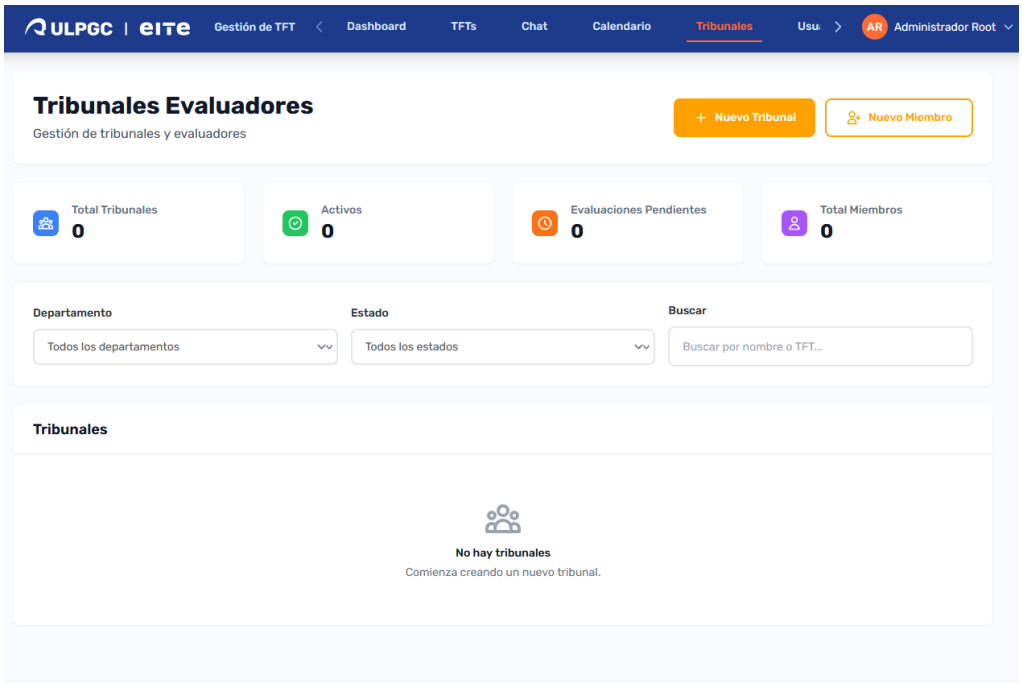

Ilustración 88: Pantalla de gestión de tribunales+3

Una característica destacada del formulario para añadir tribunales es que, al empezar a escribir el nombre de un miembro, el sistema muestra automáticamente una lista de coincidencias. Esta funcionalidad de autocompletado facilita enormemente el trabajo al usuario. Podemos ver al formulario de creación de tribunales en la Ilustración 89.

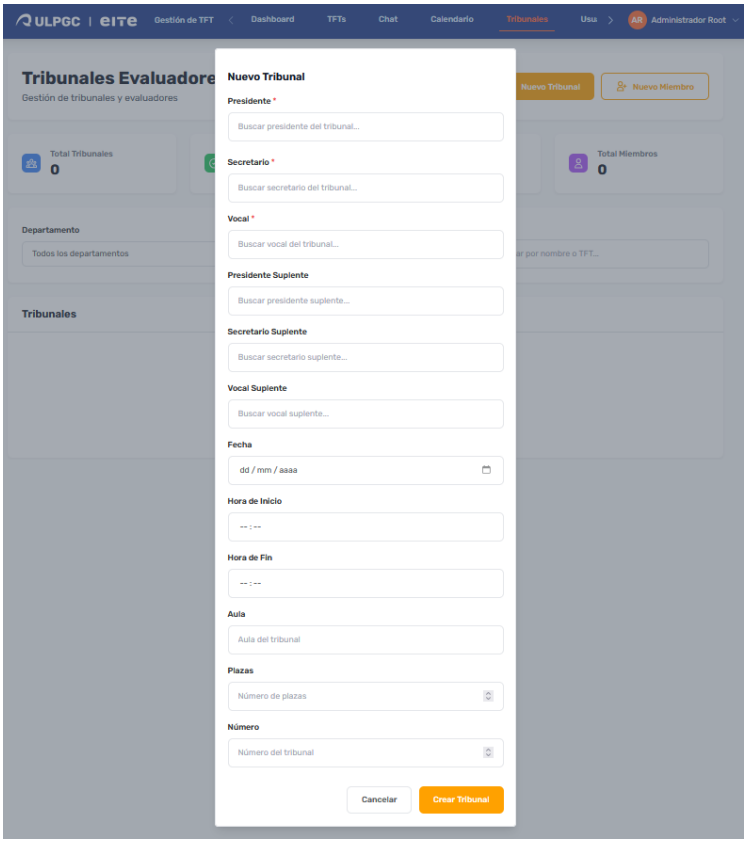

Ilustración 89: Formulario de creación de tribunal

Al pulsar el botón *Crear Tribunal*, la interfaz muestra inmediatamente el tribunal recién creado en la misma pantalla, tal y como se observa en la Ilustración 90.

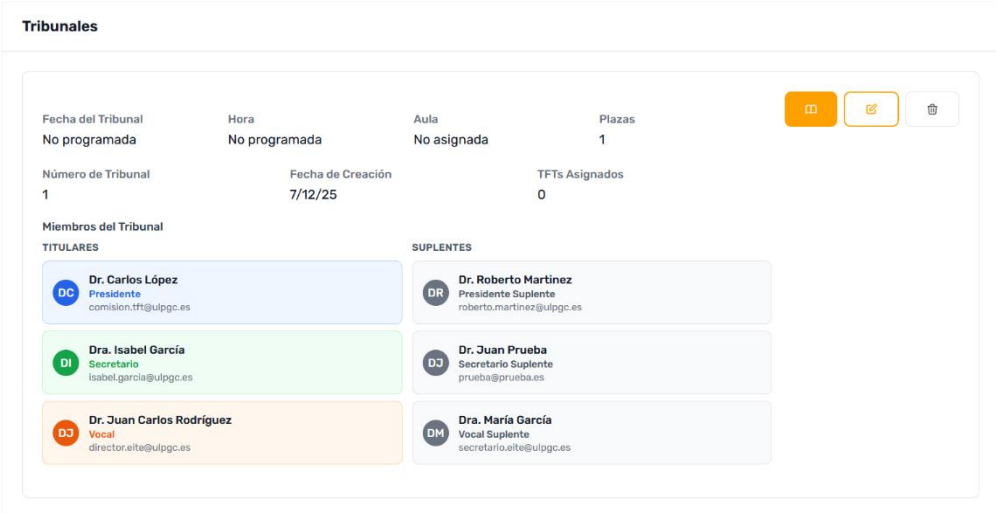

Ilustración 90: Tribunal creado

La pantalla para asignar un TFT a un tribunal se divide en dos columnas: una para los trabajos ya asignados a ese tribunal y otra para los TFTs disponibles para asignar. Para realizar la asignación, el usuario debe pulsar el icono + situado junto al TFT de prueba. Al hacerlo, el trabajo se moverá automáticamente a la columna de *TFTs Asignados*. Como resultado de esta acción, el estado del TFT también cambiará a **TRIBUNAL ASIGNADO**, tal y como se muestra en la Ilustración 91.

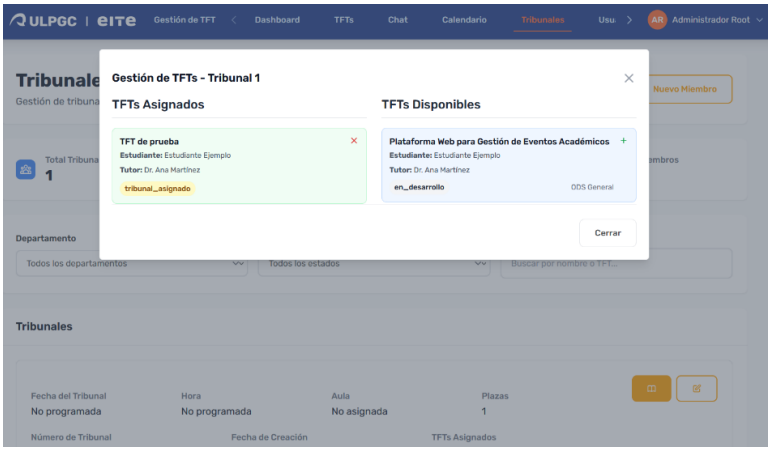

Ilustración 91: Tribunal asignado a un TFT

Si vamos a los detalles del TFT de prueba ahora aparecerán los miembros del tribunal asignado y en el caso de entrar como miembro de tribunal aparecerá la interfaz de programación de fecha y hora de la defensa como se puede ver en la Ilustración 92.

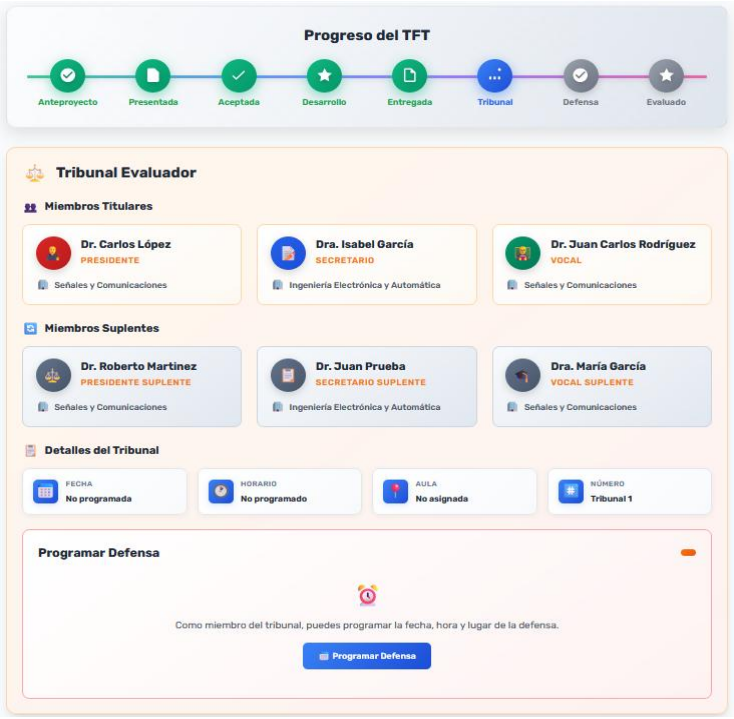

Ilustración 92: Interfaz que muestra al tribunal asignado al TFT y programación de defensa

Al darle a *Programar defensa* nos saldrá un formulario con los campos para programar la defensa como se ve en la Ilustración 93.

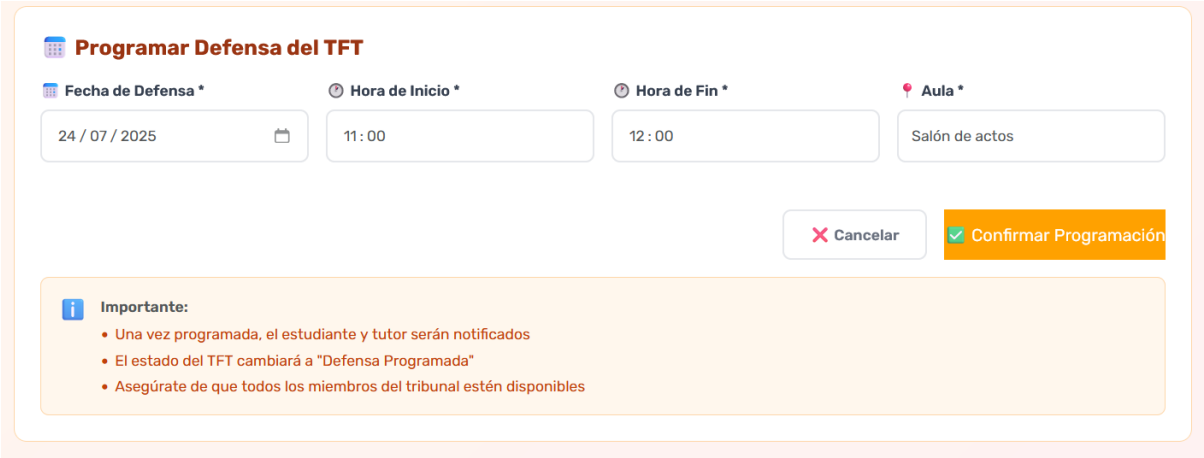

Ilustración 93: Formulario de programación de defensa

Al confirmarlo nos saldrá un mensaje de confirmación que se puede ver en la Ilustración 94.

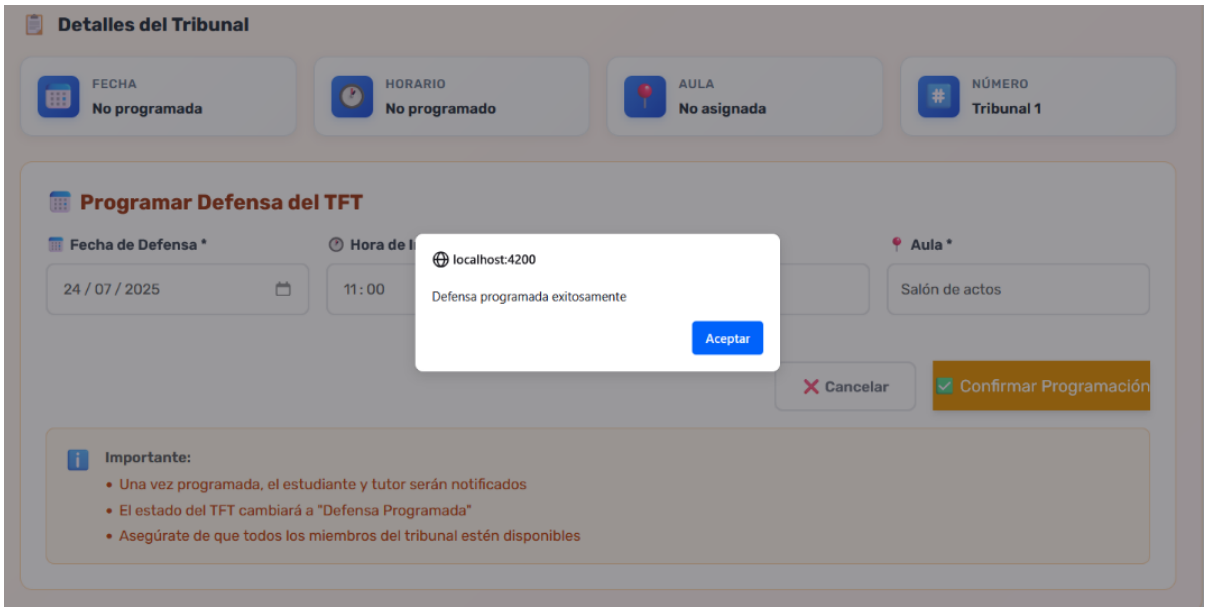

Ilustración 94: Mensaje de confirmación de defensa

La interfaz cambiará a como se puede ver en la Ilustración 95.

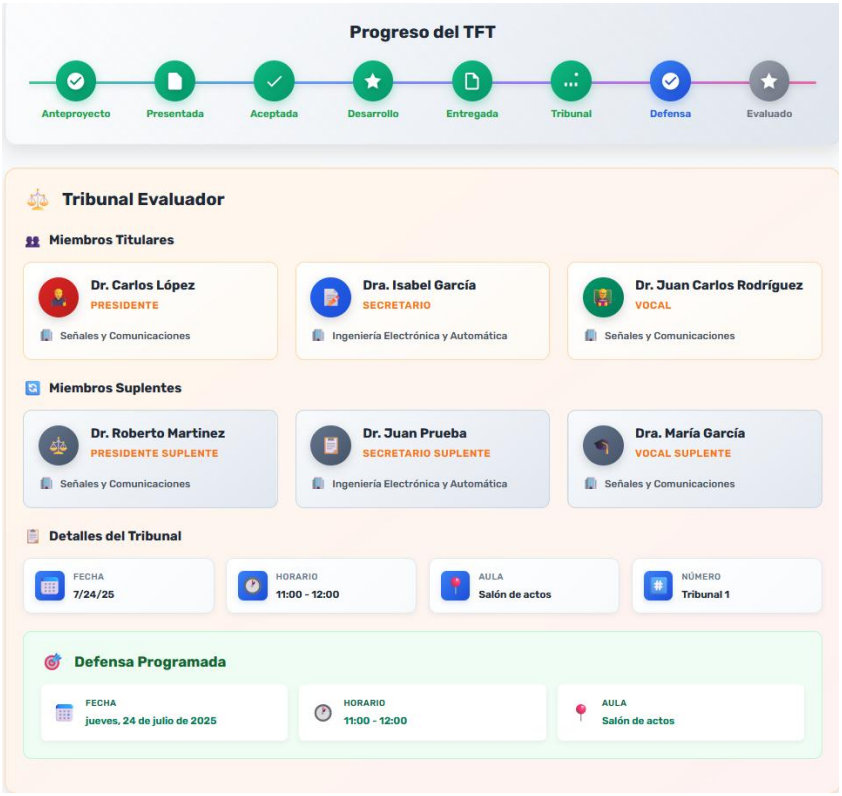

Ilustración 95: TFT programado para defensa

Una vez transcurrida la fecha y hora de la defensa, se habilita una nueva interfaz para que cada **miembro del tribunal** pueda registrar la **calificación del TFT**, como se muestra en la Ilustración 96.

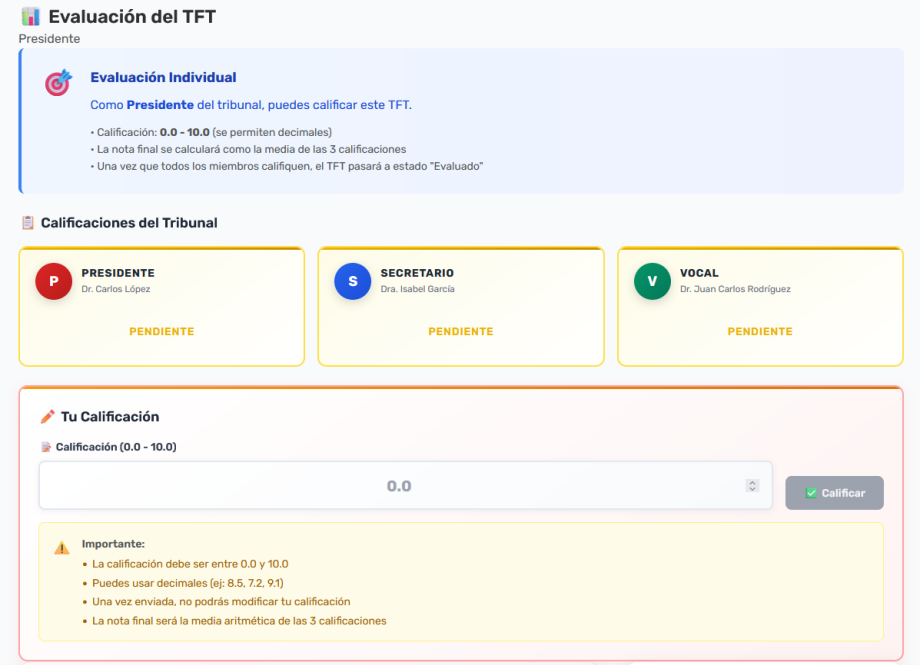

Ilustración 96: Panel de calificación del TFT

En este paso todos los **miembros del tribunal** deben añadir su nota individual, en la Ilustración 97 se puede ver como el **presidente del tribunal** ha puesto su nota.

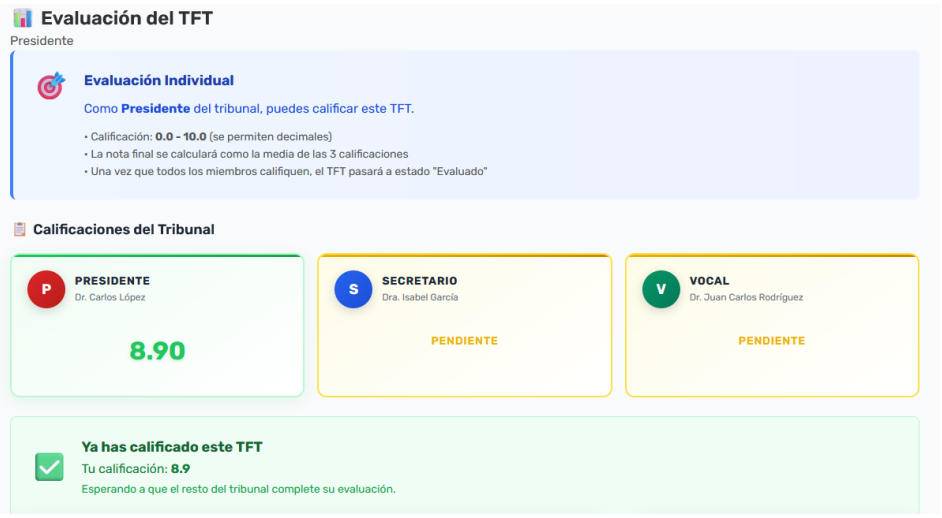

Ilustración 97: Calificación del presidente del tribunal puesta

Una vez puesta las notas por parte de todos los **miembros del tribunal**, automáticamente se mostrará la nota final que resultará de la media de las 3 notas de cada miembro del tribunal. Se puede ver en la ilustración 98.

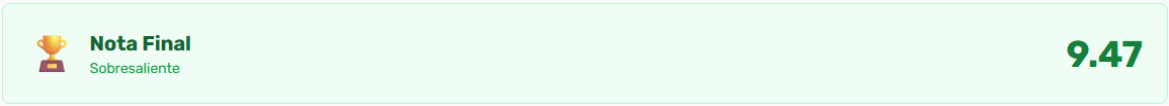

Ilustración 98: TFT calificado

Con todas las calificaciones puestas por los **miembros del tribunal** se ha acabado el flujo académico de los TFT de la EITE. En la Ilustración 99 se puede ver la barra de progreso de TFT completa.

Ilustración 99: Flujo completo de TFT

Pantalla de chat

En este apartado describiremos brevemente la funcionalidad del chat. Al acceder a esta interfaz por primera vez, el usuario la encontrará vacía. Lo único visible será el área de conversación, que se mostrará sin contenido, ya que aún no se ha iniciado ninguna conversación. Esto se ilustra en la Ilustración 100.

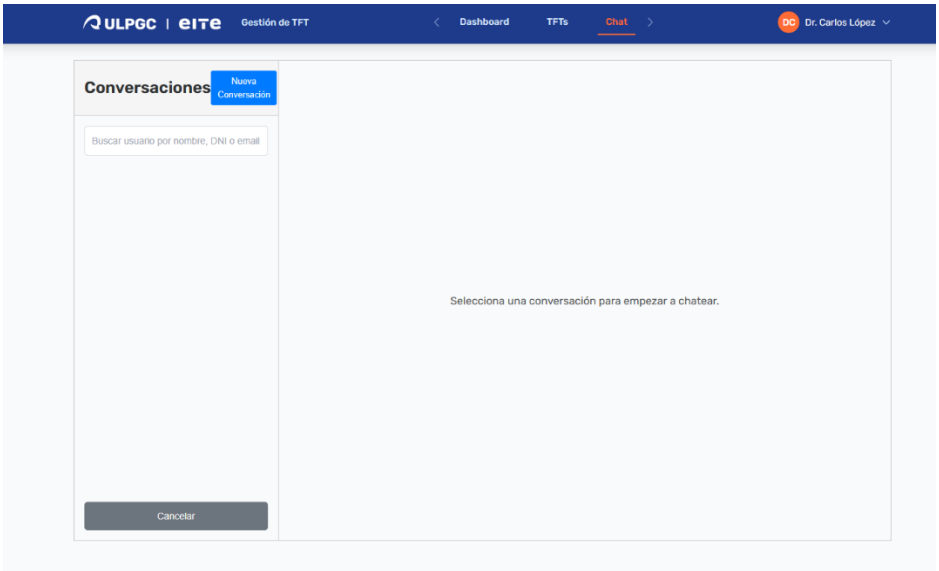

Ilustración 100: Interfaz de la pantalla de chat vacía

Ahora para crear una nueva conversación haremos *click* en el botón que dice *Nueva Conversación* y nos aparecerá un formulario de búsqueda donde podremos buscar usuarios para iniciar un chat. Escribimos al **admin root** y nos sale su chat. Hacemos una prueba con el mandándole un mensaje como se puede ver en la Ilustración 101.

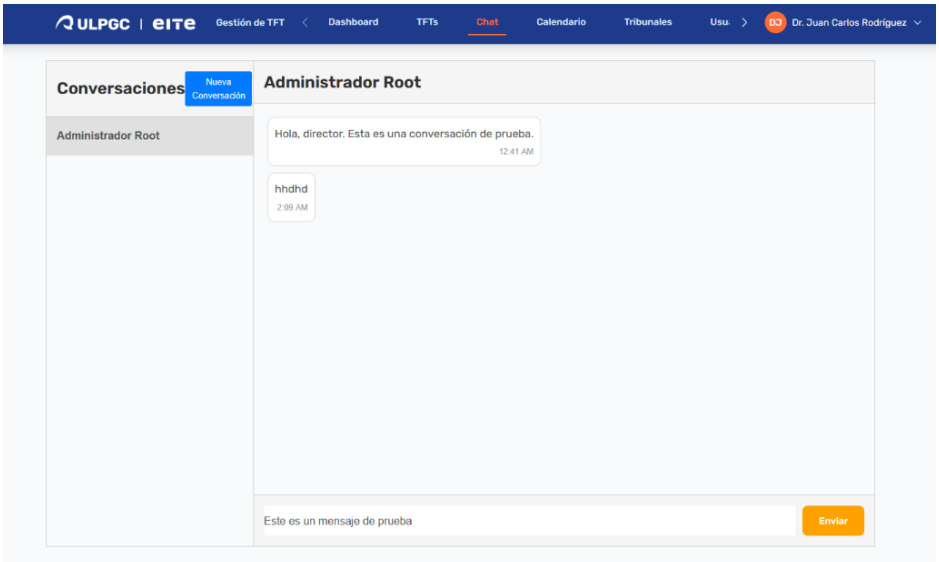

Ilustración 101: Chat establecido

Tras enviar un mensaje, se puede verificar su correcta recepción accediendo a la cuenta del **administrador root**. La Ilustración 102 muestra cómo aparece el mensaje recibido en la bandeja de entrada del administrador.

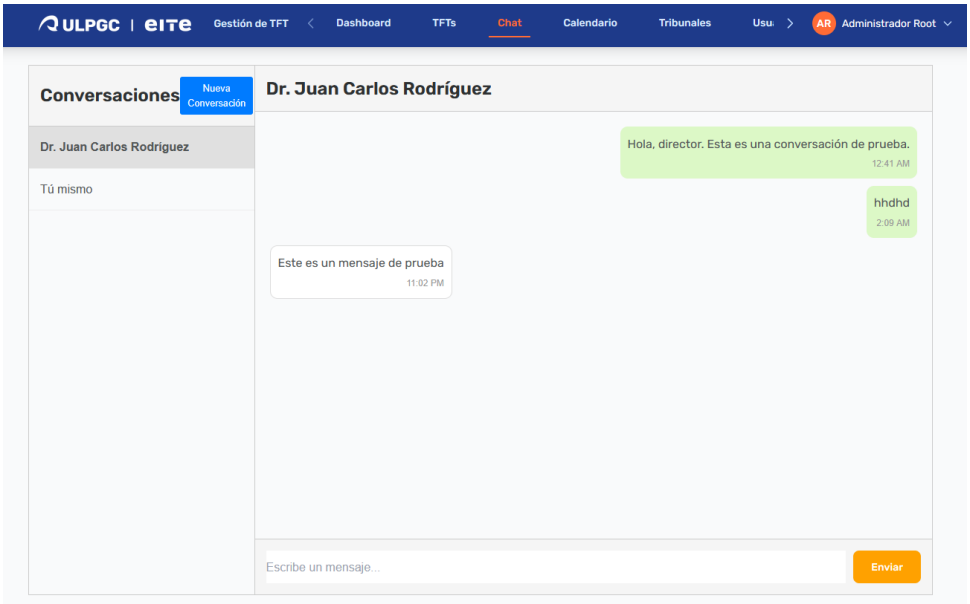

Ilustración 102: Mensaje recibido correctamente

Pantalla de formularios

Este apartado será especialmente útil para los estudiantes. Entonces accediendo como uno a la plataforma nos iremos a la pantalla de formularios y en ella podremos ver un listado de todos los formularios y documentos útiles para el usuario en cuestión. El sistema ofrece la opción de previsualizar la plantilla directamente en la plataforma, sin necesidad de descargar el documento. Esta funcionalidad evita la acumulación de archivos innecesarios en el equipo local del usuario. Podemos ver esta pantalla en la Ilustración 103.

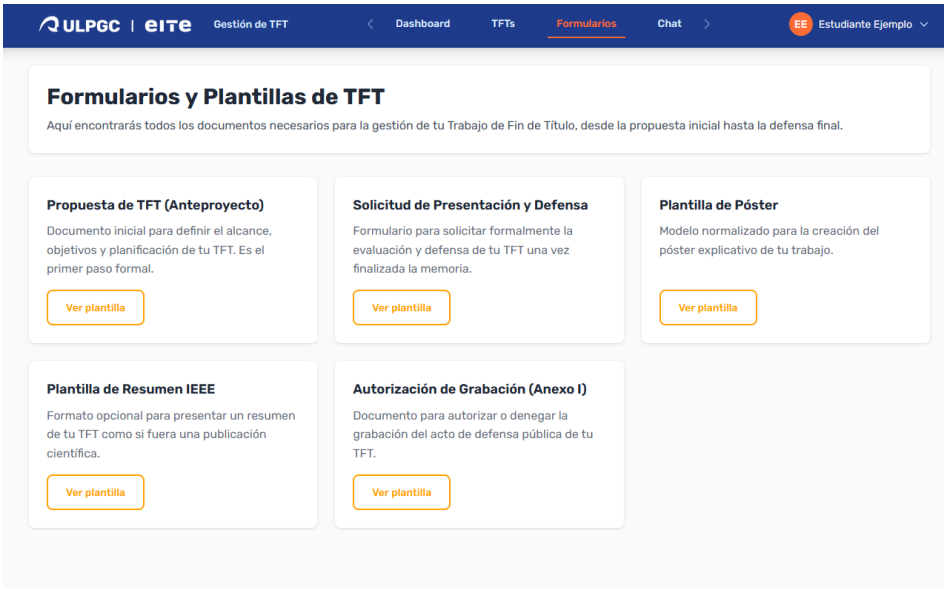

Ilustración 103: Pantalla de formularios

Si le damos al botón de *Ver plantilla* nos aparecerá una pantalla con el documento en PDF y con la opción de rellenarlo como se puede ver en la Ilustración 104.


ULPGC Universidad de Las Palmas de Gran Canaria | eite ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

Propuesta de TFT	PCC06

D./Dª. **con DNI nº** **, Domiciliado en** **, Localidad** **, Provincia** **, email** **, Teléfono de contacto** **, matriculado/a en la titulación** GIB **, en la mención (si la hay)** SISTEMAS DE TELECOMUNICACIÓN

EXPONE: Que reuniendo las condiciones requeridas para comenzar la elaboración del **Trabajo Fin de Título**, desea realizar el mismo con el título siendo su/s tutor/es o tutora/as , para lo cual cuenta con el visto bueno de los mismos reflejado en el presente documento, por lo que,

SOLICITA: Que a la vista del **anteproyecto** que se acompaña, le sean

Ilustración 104: Formulario disponible en la plataforma

7. Conclusiones y líneas futuras

En este capítulo procederemos a realizar una reflexión tras haber realizado los pasos previos que son los de diseño, implementación y correspondiente guía de usuario de los objetivos que se propusieron inicialmente para especificar su grado de cumplimiento. Además, vamos a proceder con las líneas futuras.

7.1. Conclusiones

El presente proyecto ha logrado cumplir satisfactoriamente todos los objetivos planteados, desarrollando una plataforma completa de gestión de Trabajos de Fin de Título (TFT) que abarca desde el análisis inicial hasta la implementación final.

El proyecto ha resultado en una plataforma completa y funcional que automatiza todo el proceso de gestión de TFT, desde la propuesta inicial hasta la evaluación final. La implementación técnica ha sido exitosa, utilizando tecnologías modernas y siguiendo las mejores prácticas de desarrollo. La plataforma proporciona valor significativo tanto para la institución académica como para todos los usuarios involucrados, mejorando la eficiencia operativa y reduciendo errores administrativos.

Por lo que cumplimos el objetivo general de **conseguir desarrollar una plataforma que ofrezca una solución que permita la gestión centralizada de los TFT de la EITE y autenticación de los usuarios. Entonces la finalidad de este trabajo es proporcionar la implementación del procedimiento administrativo completo que un alumno, tutor o miembro de la comisión de TFT ha de realizar desde que inicio el curso hasta que se presenta y se defienden sus TFT.**

La combinación de **Laravel** para el *backend* y **Angular** para el *frontend* ha demostrado ser una elección acertada, proporcionando una base sólida para futuras expansiones y mejoras del sistema. El cumplimiento de todos los objetivos planteados confirma el éxito del proyecto y su capacidad para satisfacer las necesidades reales de gestión de TFT en un entorno académico.

Objetivo 1: Analizar los requisitos funcionales y casos de uso para la plataforma

Se ha realizado un análisis exhaustivo de los requisitos funcionales, identificando y documentando los casos de uso principales que abarcan todo el ciclo de vida de un TFT en la EITE. El análisis incluyó la gestión de usuarios con diferentes roles (**estudiante, tutor, comisión, administrador root**…), la gestión completa de TFT desde su propuesta inicial hasta la evaluación final, la creación y gestión de tribunales evaluadores, el sistema de entregables y documentación, y la gestión del calendario académico. Este análisis sirvió como base fundamental para el diseño de la arquitectura del sistema.

Objetivo 2: Diseñar la arquitectura del sistema en donde se encuentra el *frontend*, *backend*, base de datos y los tests

Se diseñó una arquitectura robusta y escalable que separa claramente las responsabilidades entre *frontend*, *backend*, base de datos y tests. La arquitectura implementada sigue el **patrón de separación de capas**, donde el *frontend* desarrollado en **Angular 19** se encarga de la presentación y experiencia de usuario, el *backend* en **Laravel 11** maneja toda la lógica de negocio y seguridad, la base de datos **MySQL** almacena la información de manera estructurada y relacional, y los **tests automatizados** garantizan la calidad y funcionamiento correcto del código. Esta separación permite un mantenimiento eficiente y una escalabilidad independiente de cada componente.

Objetivo 3: Llevar a cabo la creación del *backend* usando PHP. Con el fin de desarrollar toda la lógica de la aplicación que será implementada totalmente desde el *backend* por motivos de seguridad. Además, se creará una API para que el *frontend* pueda solicitar y mostrar los datos. Para controlar el acceso de los usuarios, implementaremos el sistema de autenticación en el *backend* (la parte de la aplicación que funciona en el servidor) aprovechando las soluciones de seguridad que ya vienen incluidas en Laravel.

Se desarrollo completamente el código del *backend* utilizando PHP con el *framework* **Laravel 11**, implementando toda la lógica de aplicación en el *backend* por motivos de seguridad. El *backend* desarrollado incluye un sistema completo de autenticación y autorización, gestión de usuarios y roles, **APIs RESTful** para todas las funcionalidades, validación robusta de datos, y manejo seguro de archivos y documentos. La implementación de la lógica de negocio en el *backend* garantiza que las operaciones críticas no puedan ser manipuladas desde el cliente, proporcionando un alto nivel de seguridad.

Además, se desarrollaron servicios API completos que son consultados desde el *frontend* para proporcionar los datos necesarios. Los servicios implementados incluyen *endpoints* para gestión de usuarios, TFT, tribunales, evaluaciones, entregables y calendario académico. Cada API sigue las mejores prácticas **RESTful**, incluyen validación de datos, manejo de errores consistente, y respuestas JSON estandarizadas. El sistema de APIs permite una comunicación eficiente y segura entre el *frontend* y *backend*.

Finalmente, se implementó un sistema de autenticación robusto utilizando las herramientas proporcionadas por **Laravel**, específicamente **Laravel Sanctum** para la autenticación de APIs. El sistema incluye autenticación JWT, *middleware* de autorización basado en roles, políticas de acceso granulares, y gestión segura de sesiones. La implementación garantiza que solo los usuarios autorizados puedan acceder a las

funcionalidades correspondientes a su rol, proporcionando un alto nivel de seguridad en toda la aplicación.

Objetivo 4: Desarrollar un *frontend* responsive con Angular para ofrecer una interfaz de usuario intuitiva y eficiente que permita a los distintos usuarios de la plataforma realizar sus gestiones.

Se desarrolló un *frontend responsive* utilizando la librería Angular para proporcionar una interfaz de usuario intuitiva que permite a los distintos usuarios de la plataforma realizar las gestiones correspondientes. El *frontend* implementado incluye componentes modulares con *lazy loading*, interfaz *responsive* con **Tailwind CSS**, formularios con validación en tiempo real, y navegación intuitiva. La aplicación se adapta a diferentes dispositivos y proporciona una experiencia de usuario moderna y funcional para todos los roles del sistema.

Objetivo 5: Verificación del código empleado mediante los tests desarrollados para comprobar el correcto funcionamiento del código

Se implementó un sistema de tests exhaustivo para verificar el correcto funcionamiento del código. Los tests incluyen tests unitarios para modelos y servicios, tests de integración para las APIs, y tests de aceptación para los flujos críticos de usuario. Esta verificación automatizada garantiza la calidad del código, detecta errores tempranamente, y facilita el mantenimiento y evolución del sistema.

En la Ilustración 105 podemos ver como los tests del *frontend* pasan exitosamente todas y cada una de las pruebas.

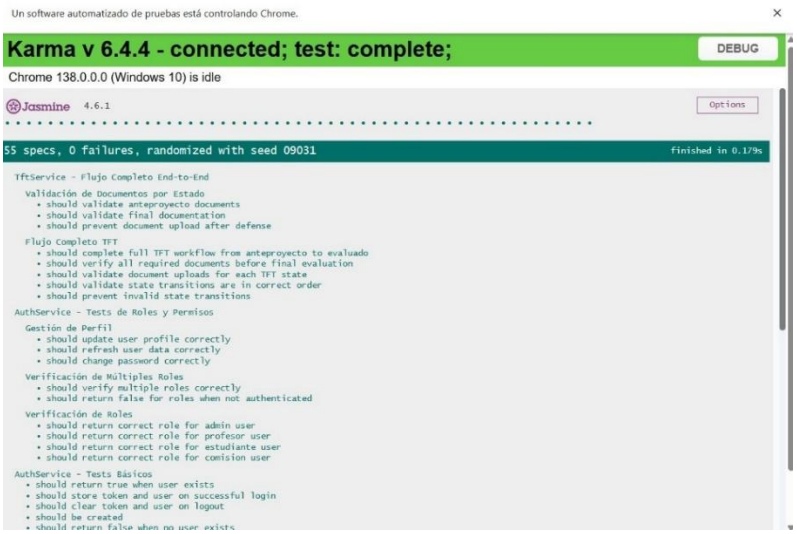

Ilustración 105: Tests del frontend pasan

Por otro lado, en la Ilustración 106 podemos observar cómo los test del *backend* pasan las pruebas.

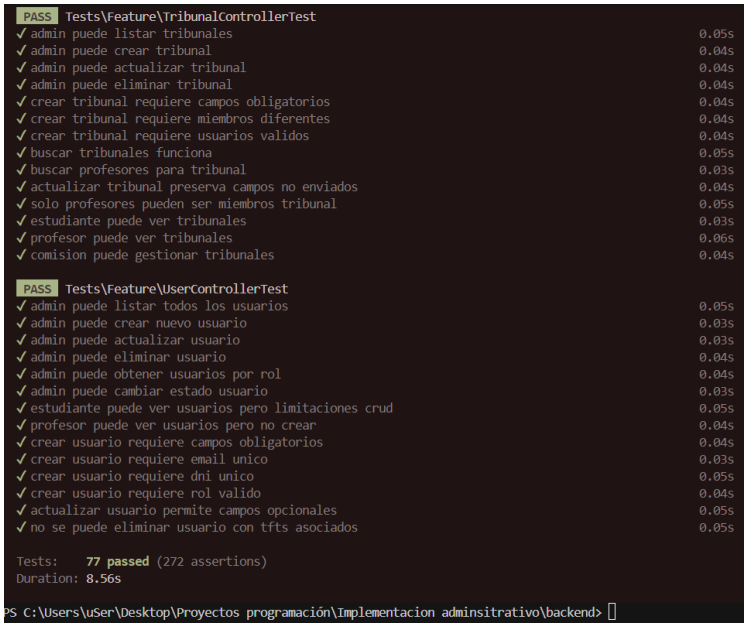

Ilustración 106: Tests del backend pasan

7.2. Líneas futuras

El presente proyecto de *Plataforma de Gestión de Trabajos de Fin de Título* (TFT) ha establecido una base sólida y funcional que permite identificar múltiples líneas de desarrollo futuro para expandir sus capacidades y mejorar su funcionalidad.

Línea Futura 1: Expansión de funcionalidades académicas

Se plantea la implementación de funcionalidades adicionales que complementen el proceso actual de gestión de TFT. Estas mejoras incluirían la integración de un sistema de gestión de bibliografía y referencias bibliográficas automáticas, la implementación de un módulo de seguimiento de horas de trabajo del estudiante, y la creación de un sistema de alertas y notificaciones avanzadas que informen sobre fechas límite, cambios de estado y requerimientos pendientes. Asimismo, se contempla la integración con sistemas externos de la **Universidad** como el sistema de matrículas, el repositorio institucional y las plataformas de gestión académica existentes, así como del sistema de autenticación propio de la universidad, así como el que implica certificado digital.

Línea Futura 2: Mejoras en la Experiencia de Usuario

Se propone el desarrollo de mejoras significativas en la interfaz de usuario y la experiencia del usuario. Estas mejoras incluirían la implementación de un **dashboard** personalizado para cada tipo de usuario con métricas y estadísticas relevantes, la creación de un sistema de búsqueda avanzada con filtros múltiples para localizar TFT específicos, y el desarrollo de una aplicación móvil nativa que permita a los usuarios acceder a las funcionalidades principales desde dispositivos móviles.

Línea Futura 3: Análisis de Datos y Business Intelligence

Se plantea la implementación de un sistema de análisis de datos y *business intelligence* que proporcione *insights* valiosos sobre el proceso de gestión de TFT. Este sistema incluiría la creación de informes automáticos sobre el rendimiento académico, estadísticas de aprobación y tiempos promedio de completación, análisis predictivo para identificar TFT en riesgo de no cumplir los plazos, y la generación de métricas de calidad y satisfacción del usuario. Asimismo, se contempla la implementación de un sistema de auditoría avanzado que registre todas las acciones realizadas en la plataforma para fines de trazabilidad y cumplimiento normativo. Todo esto podría hacerse con *IA*.

Línea Futura 4: Integración con herramientas externas

Se propone la integración con herramientas y servicios externos que enriquezcan la funcionalidad de la plataforma. Estas integraciones incluirían la conexión con sistemas de detección de plagio para validar la originalidad de los trabajos, la integración con herramientas de gestión de proyectos como **Trello** o **Asana** para el seguimiento de tareas, la conexión con sistemas de videoconferencia para realizar defensas virtuales, y la integración con plataformas de almacenamiento en la nube como **OneDrive** con la que trabaja la ULPGC para la gestión de documentos. Igualmente se puede integrar con el correo institucional de la universidad para cuando se genere un nuevo usuario, este reciba las credenciales automáticamente allí.

Línea Futura 5: Mejoras en seguridad y cumplimiento

Se plantea el desarrollo de mejoras significativas en el ámbito de seguridad y cumplimiento normativo con el fin de optimizar el cumplimiento del Esquema Nacional de Seguridad. Estas mejoras incluirían la implementación de **autenticación multifactorial** (MFA) para todos los usuarios, la creación de un sistema de gestión de permisos más granular basado en políticas de acceso, la implementación de cifrado *end-to-end* para documentos sensibles, y el desarrollo de un sistema de *backup* automático y recuperación ante incidentes. Asimismo, se contempla la implementación de auditorías de seguridad

regulares y la conformidad con estándares internacionales como *GDPR* para la protección de datos personales.

Línea Futura 6: Escalabilidad y optimización técnica

Se propone el desarrollo de mejoras técnicas que permitan la escalabilidad del sistema y la optimización del rendimiento. Estas mejoras incluirían la optimización de consultas de base de datos y la implementación de caché distribuido, la migración a contenedores *Docker* para facilitar el despliegue y la gestión de la aplicación, y la implementación de un sistema de monitoreo y alertas en tiempo real. Además, se contempla la implementación de pruebas de carga y estrés para garantizar el rendimiento bajo condiciones de alta demanda.

Impacto esperado de las líneas futuras

La implementación de estas líneas futuras permitiría transformar la plataforma actual en un sistema integral de gestión académica que no solo automatice los procesos existentes, sino que también proporcione herramientas avanzadas de análisis, colaboración y gestión de calidad. Estas mejoras resultarían en una mayor eficiencia operativa, una mejor experiencia de usuario, y una plataforma más robusta y escalable que pueda adaptarse a las necesidades cambiantes de las instituciones educativas. La combinación de funcionalidades avanzadas, mejoras técnicas y capacidades de integración posicionaría la plataforma como una solución líder en el mercado de gestión de trabajos académicos.

Bibliografía

[1] T. T. Phuong Thao, T.-T. Nguyen y N. Danh Nam, «Digital transformation in education: a bibliometric analysis using Scopus,» *European Science Editing,* 2023.

[2] S. Saini, K. Gomis, Y. Polychronakis, M. Saini y S. Sapountzis, «Identifying challenges in implementing digital transformation in UK higher education,» *Quality Assurance in Education,* 2024.

[3] P. Permata Sari y S. Mas Ayu, «Digital Transformation of Educational Management: Accelerating Students' Digital Literacy Through Educational Technology,» *Vol. 2 No. 1 (2025): Raden Intan International Conference on Islamic Studies (RIICIS 2024),* 2025.

[4] Fundación CYD, «¿Cómo afrontan las universidades españolas su transformación digital?,» 19 octubre 2023. [En línea]. Available: https://www.fundacioncyd.org/como-afrontan-las-universidades-espanolas-su-transformacion-digital/. [Último acceso: 2025 julio 7].

[5] Gobierno de Canarias, «Estrategia de Especialización Inteligente de Canarias (RIS3 ampliada,» 2023. [En línea]. Available: https://www.redpoliticasidi.es/system/files/repositorio-archivos/RIS3-ampliada_marzo-2023.pdf. [Último acceso: 7 julio 2025].

[6] Universidad de La Laguna, «La ULL presenta su Plan de Modernización Administrativa con 48 proyectos sustentados en un proceso participativo,» 25 septiembre 2024. [En línea]. Available: https://www.ull.es/portal/noticias/2024/la-ull-presenta-su-plan-de-modernizacion-administrativa-con-48-proyectos-sustentados-en-un-proceso-participativo/. [Último acceso: 7 julio 2025].

[7] Fundación Universitaria de Las Palmas, «Programa Diginnova | Formación Gratuita Certificada por la Universidad de Las Palmas de Gran Canaria,» 2024. [En línea]. Available: https://www.fulp.es/programas/diginnova/beneficiarios/. [Último acceso: 7 julio 2025].

[8] Escuela Superior de Ingeniería - UCA, «Trabajo Fin de Grado/Máster (TFG/M),» 25 septiembre 2024. [En línea]. Available: https://esingenieria.uca.es/docencia/tfgm/. [Último acceso: 8 julio 2025].

[9] Universidad de Málaga, Escuela de Ingenierías Industriales, «Trabajo Fin de Grado,» 2025. [En línea]. Available: https://www.uma.es/escuela-de-ingenierias-industriales/info/104573/informacion-general-sobre-el-trabajo-fin-de-grado/. [Último acceso: 8 julio 2025].

[10] Universidad de Alcalá, «Aplicación de gestión de Trabajos Fin de Grado (TFG) y Trabajos Fin de Máster (TFM),» 2025. [En línea]. Available: https://gestiontfx.uah.es/GestTFx/. [Último acceso: 8 julio 2025].

[11] Universidad Rey Juan Carlos, «Trabajo Fin de Grado,» 2025. [En línea]. Available: https://gestion2.urjc.es/tfg/. [Último acceso: 8 julio 2025].

[12] Universidad Pablo de Olavide, «TFG-APP: Aplicación para la gestión integral de Trabajos Fin de Grado y Máster,» 2024. [En línea]. Available: https://www.upo.es/plan-transformacion-digital/proyectos/TFG-APP-Aplicacion-para-la-gestion-integral-de-Trabajos-Fin-de-Grado-y-Master/. [Último acceso: 8 julio 2025].

[13] DynFile Sistemas, «ExDin® TFGM: Gestión y presentación de trabajos fin de grado y máster,» [En línea]. Available: https://www.dynfile.es/exdintfgm-gestion-y-presentacion-de-trabajos-fin-de-grado-y-master/. [Último acceso: 8 julio 2025].

[14] The PHP Group, «PHP Manual,» [En línea]. Available: https://www.php.net/manual/es/. [Último acceso: 8 julio 2025].

[15] Microsoft, «TypeScript Documentation,» [En línea]. Available: https://www.typescriptlang.org/docs/. [Último acceso: 8 julio 2025].

[16] A. Beaulieu, Learning SQL, 3rd ed, O'Reilly Media, 2020.

[17] Oracle Corporation, «MySQL 8.0 Reference Manual,» [En línea]. Available: https://dev.mysql.com/doc/refman/8.0/en/. [Último acceso: 9 julio 2025].

[18] T. Otwell, «Laravel Documentation,» [En línea]. Available: https://laravel.com/docs/11.x. [Último acceso: 9 julio 2025].

[19] Google, «Angular Documentation,» [En línea]. Available: https://angular.dev/overview. [Último acceso: 9 julio 2025].

[20] Microsoft, «Visual Studio Code Docs,» [En línea]. Available: https://code.visualstudio.com/docs. [Último acceso: 9 julio 2025].

[21] GitHub, Inc, «GitHub Docs,» [En línea]. Available: https://docs.github.com/es. [Último acceso: 9 julio 2025].

[22] Figma, Inc, «Figma Help Center,» [En línea]. Available: https://help.figma.com/hc/en-us. [Último acceso: 9 julio 2025].

[23] Postman, Inc, «Postman Learning Center,» [En línea]. Available: https://learning.postman.com/docs/introduction/overview/. [Último acceso: 7 julio 2025].

[24] T. Otwell, «Laravel Sanctum,» [En línea]. Available: https://laravel.com/docs/11.x/sanctum. [Último acceso: 9 julio 2025].

[25] T. Otwell, «Laravel Reverb,» [En línea]. Available: https://laravel.com/docs/11.x/reverb. [Último acceso: 9 julio 2025].

[26] Pusher, «pusher-http-php,» [En línea]. Available: https://github.com/pusher/pusher-http-php. [Último acceso: 9 julio 2025].

[27] S. Bergmann, «PHPUnit Manual,» [En línea]. Available: https://docs.phpunit.de/en/11.2/index.html. [Último acceso: 10 julio 2025].

[28] ngx-toastr, «npmjs.com,» [En línea]. Available: https://www.npmjs.com/package/ngx-toastr. [Último acceso: 9 julio 2025].

[29] Google, «Angular CLI,» [En línea]. Available: https://angular.dev/cli. [Último acceso: 10 julio 2025].

[30] The Karma Team, «Karma Documentation,» [En línea]. Available: https://karma-runner.github.io/6.4/index.html. [Último acceso: 10 julio 2025].

[31] The Jasmine Team, «Jasmine Documentation,» [En línea]. Available: https://jasmine.github.io/pages/docs_home.html. [Último acceso: 10 julio 2025].

Presupuesto

Aquí en este apartado realizaremos la estimación del supuesto del coste de realización de este Trabajo de Fin de Grado.

1. Desglose del presupuesto

Con el fin de desglosar el presupuesto de este TFG vamos a seguir las pautas dadas por el Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT) siendo los siguientes:

- Recursos materiales.
- Trabajo tarificado por tiempo empleado.
- Costes de redacción del documento.
- Derechos del visado del COITT.
- Gastos de tramitación y envio.
- Aplicación de impuestos.

2. Recursos materiales

Para el caso de los recursos materiales vamos a tener en cuenta tanto a aquellos que sean de software como de hardware.

Por un lado, como los recursos de software usados a lo largo de este proyecto son gratuitos, el coste será de 0€.

Pero los recursos hardware si tendrán un coste que será de un estimado basado a su valor de adquisición y serán ponderados en función de su valor residual y años de vida útil. Siendo calculado el coste de amortización con la siguiente formula:

$$Coste\ de\ amortización = \frac{Valor\ de\ adquisición - Valor\ residual}{Años\ de\ vida\ útil}$$

Siendo el valor de adquisición en este caso el precio de costo al momento de comprar el recurso. Por otro lado, el valor residual es el valor de los recursos tras haber pasado el tiempo de vida útil estipulado.

Además, el coste de amortización se dividirá entre 4 ya que aproximadamente hemos estado haciendo uso de estos recursos poco más de un cuarto de año.

Recurso	Valor de adquisición	Valor residual	Años de vida útil	Coste de amortización
Ordenador sobremesa Windows 11	1310,14€	900€	4	25,64€
Ordenador sobremesa macOS	1517,68€	1000€	4	32,36€
Ordenador portátil	573,90€	400€	4	10,87€
Teléfono móvil personal	1051,44€	800€	4	15,72€
			TOTAL	**84,59€**

Tabla 2: Coste de amortización de los recursos hardware

Los recursos hardware que han sido usados durante la elaboración de este trabajo y sus cosas asociados se detallan en la Tabla 2 que da como resultado total de **84,59€**

3. Trabajo tarificado por tiempo empleado

La realización de este trabajo de fin de grado tal y como establece el proyecto docente ha resultado en un total de 300 horas que incluyen todas las tareas propuestas durante aproximadamente 4 meses de trabajo.

Siendo necesario conocer estos valores para calcular el importe recibido por las horas trabajadas de un Ingeniero de Telecomunicaciones. Siguiendo las recomendaciones del COITT se calcularía con las siguiente formula:

$$H = C_t \cdot 74{,}88 \cdot H_n + 96{,}72 \cdot H_e$$

Siendo en esta fórmula el importe recibido (H) calculado a partir de los siguientes componentes:

- Factor de corrección en función de las horas trabajadas (C_t).
- Horas trabajadas dentro de la jornada laboral (horas normales) (H_n).
- Horas trabajadas fuera de la jornada laboral (horas especiales) (H_e).

Siendo el cálculo del factor de corrección dado por las horas trabajadas. Siendo en nuestro caso unas 300 horas entonces el factor de corrección será de 0,6. Con esto el resultado de la ecuación sería el siguiente:

$$\boldsymbol{H} = 0{,}6 \cdot 74{,}88 \cdot 300 + 96{,}72 \cdot 0 = \mathbf{13\,478{,}40}€$$

Entonces dado el resultado total dado por la ecuación, los honorarios totales ascienden a una cantidad de **13 478,40€.**

4. Costes asociados con la redacción del documento

El coste de redacción será calculado en función de la siguiente formula:

$$R = 0{,}07 \cdot P \cdot C_n$$

Siendo los valores de los cuales se calcula el coste serán:

- Presupuesto del proyecto (P).
- Coeficiente de ponderación en función del presupuesto (C_n).

En el COITT se encuentra establecido un valor del coeficiente de ponderación de 1 para el caso de proyectos con presupuesto menor de 30 050€ como el nuestro.

Entonces los costes asociados con la redacción del documento son:

$$\boldsymbol{R} = 0{,}07 \cdot 13\,562{,}99 \cdot 1 = \mathbf{949{,}41}€$$

A raíz del resultado de la formula anterior tenemos un coste asociado con la redacción del documento de **945,98€.**

5. Derechos de visado COITT

El coste del visado del COITT se pueden calcular usando la siguiente formula:

$$V = 0{,}006 \cdot P \cdot C_v$$

Teniendo la formula los siguientes elementos:

- Presupuesto del trabajo (P).
- Coeficiente reductor en función del presupuesto del trabajo (C_v).

El presupuesto P por ahora suma el monto total de **14 427,81€.** Por lo que siguiendo las directrices del COITT el coeficiente reductor será de 1 ya que el presupuesto no supera los 30 050€.

Entonces el coste total de los derechos de visado del COITT serán los siguientes:

$$\boldsymbol{V} = 0{,}006 \cdot 14\,427{,}81 \cdot 1 = \mathbf{85{,}57}€$$

De acuerdo con el resultado a esta fórmula vemos que el coste total de los derechos del visado del COITT queda en **85,57€.**

6. Gastos de tramitación y envio

Los gastos relativos a la tramitación y envio están fijados en **7€**.

7. Aplicación de impuestos

Para concluir el coste total de este TFG se encuentra en **14 513,38€.** Nos encontramos en Canarias que cuenta con un sistema fiscal especial, entonces se sumara el Impuesto General Indirecto Canario (IGIC) que se encuentra en un 7% sobre el coste total. Entonces el resultado es el siguiente:

Recurso	Coste (€)
Recursos materiales	84,59
Trabajo tarificado por tiempo empleado	13 478,40
Costes asociados a la redacción del documento	945,98
Derecho de visado COITT	85,57
Gastos de tramitación y envío	7
Aplicación de impuestos (7%)	1 015,94
Total	**15 529,32**

Tabla 3: Cálculo total del presupuesto del TFG

Entonces a raíz de todos los cálculos y de la Tabla 3 el presupuesto total asciende a la suma de **15 529,32€.**

Las Palmas de Gran Canaria, a 15 de julio de 2025

Fdo.: Freddy Alexander López Gutiérrez

ANEXOS

1. ENDPOINTS API RESTFUL

AUTENTICACIÓN Y PERFIL

1.1 Login de Usuario
- Ruta: POST /api/login
- Descripción: Autentica un usuario con DNI y contraseña
- Autenticación: No requerida
- Content-Type: application/json
- Body:

```json
{
    "dni": "12345678A",
    "password": "password123"
}
```

- **Respuesta Exitosa (200):**

```json
{
    "message": "Login exitoso",
    "user": {
        "id": 1,
        "name": "Usuario Ejemplo",
        "email": "usuario@example.com",
        "dni": "12345678A",
        "role": "estudiante"
    },
    "token": "1|abc123def456..."
}
```

1.2 Registro de Usuario
- Ruta: POST /api/register
- Descripción: Registra un nuevo usuario (solo administradores)
- Autenticación: No requerida
- Content-Type: application/json
- Body:

```json
{
    "name": "Nuevo Usuario",
    "email": "nuevo@example.com",
    "dni": "87654321B",
    "password": "password123",
    "password_confirmation": "password123",
    "role": "estudiante",
    "departamento_id": 1,
```

```
    "certificado_firmado": false
}
```

1.3 Logout de Usuario
- Ruta: POST /api/logout
- Descripción: Cierra la sesión del usuario autenticado
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Body: Vacío
- Respuesta (200):
```json
{
    "message": "Logout exitoso"
}
```

1.4 Obtener Usuario Autenticado
- Ruta: GET /api/user
- Descripción: Obtiene información del usuario autenticado
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Respuesta (200):
```json
{
    "id": 1,
    "name": "Usuario Ejemplo",
    "email": "usuario@example.com",
    "dni": "12345678A",
    "role": "estudiante"
}
```

1.5 Actualizar Perfil
- Ruta: PUT /api/profile
- Descripción: Actualiza el perfil del usuario autenticado
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Body:
```json
{
    "name": "Nombre Actualizado",
    "email": "nuevo@email.com"
}
```

1.6 Cambiar Contraseña
- Ruta: POST /api/change-password
- Descripción: Cambia la contraseña del usuario autenticado
- Autenticación: Bearer Token requerido
- Content-Type: application/json

- Body:
```json
{
   "current_password": "password123",
   "password": "newpassword123",
   "password_confirmation": "newpassword123"
}
```

2. SISTEMA DE CHAT

2.1 Obtener Conversaciones
- Ruta: GET /api/chat/conversations
- Descripción: Obtiene todas las conversaciones del usuario autenticado
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Respuesta (200):
```json
[
   {
      "id": 1,
      "name": "Conversación con Tutor",
      "created_at": "2024-01-01T10:00:00Z",
      "updated_at": "2024-01-01T15:30:00Z"
   }
]
```

2.2 Obtener Mensajes de Conversación
- Ruta: GET /api/chat/conversations/{conversation}/messages
- Descripción: Obtiene todos los mensajes de una conversación específica
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Respuesta (200):
```json
[
   {
      "id": 1,
      "body": "Hola, ¿cómo va el proyecto?",
      "user_id": 1,
      "conversation_id": 1,
      "created_at": "2024-01-01T10:00:00Z"
   }
]
```

2.3 Enviar Mensaje
- Ruta: POST /api/chat/conversations/{conversation}/messages
- Descripción: Envía un mensaje a una conversación específica
- Autenticación: Bearer Token requerido
- Content-Type: multipart/form-data

- Body:
```json
{
   "body": "Mensaje de texto",
   "file": "archivo.pdf" // Opcional
}
```

2.4 Iniciar Conversación
- Ruta: POST /api/chat/conversations/start
- Descripción: Inicia una nueva conversación con otro usuario
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Body:
```json
{
   "user_id": 2
}
```

3. GESTIÓN DE USUARIOS

3.1 Listar Usuarios
- Ruta: GET /api/users
- Descripción: Obtiene lista de usuarios con filtros opcionales
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Query Parameters:
  - `role`: Filtrar por rol
  - `departamento_id`: Filtrar por departamento
  - `search`: Búsqueda por nombre, email o DNI
- Respuesta (200):
```json
[
   {
      "id": 1,
      "name": "Usuario Ejemplo",
      "email": "usuario@example.com",
      "dni": "12345678A",
      "role": "estudiante",
      "active": true
   }
]
```

3.2 Crear Usuario
- Ruta: POST /api/users
- Descripción: Crea un nuevo usuario
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:

```json
{
   "name": "Nuevo Usuario",
   "email": "nuevo@example.com",
   "dni": "87654321B",
   "password": "password123",
   "role": "estudiante",
   "departamento_id": 1,
   "active": true,
   "certificado_firmado": false
}
```

3.3 Obtener Usuario Específico
-Ruta: GET /api/users/{id}
- Descripción: Obtiene información de un usuario específico
- Autenticación: Bearer Token requerido
- Content-Type: application/json`

3.4 Actualizar Usuario
- Ruta: PUT /api/users/{id}
- Descripción: Actualiza información de un usuario
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
   "name": "Nombre Actualizado",
   "email": "nuevo@email.com",
   "role": "profesor",
   "departamento_id": 2
}
```

3.5 Eliminar Usuario
- Ruta: DELETE /api/users/{id}
- Descripción: Elimina un usuario del sistema
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

3.6 Usuarios por Rol
- Ruta: GET /api/users/role/{roleName}
- Descripción: Obtiene usuarios filtrados por rol específico
- Autenticación: Bearer Token requerido
- Content-Type: application/json

3.7 Cambiar Estado de Usuario
- Ruta: POST /api/users/{id}/toggle-status
- Descripción: Activa/desactiva un usuario
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

3.8 Importar Usuarios
- Ruta: POST /api/users/import
- Descripción: Importa usuarios desde archivo CSV
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: multipart/form-data
- Body:
```json
{
   "file": "usuarios.csv"
}
```

4. GESTIÓN DE TFTs

4.1 Listar TFTs
- Ruta: GET /api/tfts
- Descripción: Obtiene lista de TFTs según el rol del usuario
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Query Parameters:
  - `estado`: Filtrar por estado
  - `calendario_id`: Filtrar por convocatoria
- **Respuesta (200):**
```json
{
   "data": [
      {
         "id": 1,
         "titulo": "Sistema de Gestión TFT",
         "descripcion": "Desarrollo de sistema web",
         "estado": "en_desarrollo",
         "estudiante_id": 1,
         "tutor_id": 2
      }
   ],
   "current_page": 1,
   "per_page": 15
}
```

4.2 Crear TFT
- Ruta: POST /api/tfts
- Descripción: Crea un nuevo TFT
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
   "titulo": "Nuevo TFT",
   "descripcion": "Descripción del proyecto",
```

```
    "estudiante_id": 1,
    "tutor_id": 2,
    "cotutor_id": 3
}
```

4.3 Obtener TFT Específico
- Ruta:GET /api/tfts/{id}
- Descripción: Obtiene información detallada de un TFT
- Autenticación: Bearer Token requerido
- Content-Type: application/json

4.4 Actualizar TFT
- Ruta: PUT /api/tfts/{id}
- Descripción: Actualiza información de un TFT
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Body:
```json
{
    "titulo": "Título Actualizado",
    "descripcion": "Descripción actualizada",
    "objetivos": "Objetivos del proyecto",
    "tareas_realizar": "Tareas a realizar"
}
```

4.5 Eliminar TFT
- Ruta: DELETE /api/tfts/{id}
- Descripción: Elimina un TFT del sistema
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

4.6 Estadísticas de TFTs
- Ruta: GET /api/tfts/stats
- Descripción: Obtiene estadísticas generales de TFTs
- Autenticación: Bearer Token requerido
- Content-Type: application/json

4.7 TFTs Recientes
- Ruta: GET /api/tfts/recent
- Descripción: Obtiene los TFTs más recientes
- Autenticación: Bearer Token requerido
- Content-Type: application/json

4.8 TFTs por Estudiante
- Ruta: GET /api/tfts/student/{studentId}
- Descripción: Obtiene TFTs de un estudiante específico
- Autenticación: Bearer Token requerido
- Content-Type: application/json

5. GESTIÓN DE DOCUMENTOS

5.1 Listar Documentos de TFT
- Ruta: GET /api/tfts/{tft}/documentos
- Descripción: Obtiene todos los documentos de un TFT
- Autenticación: Bearer Token requerido
- Content-Type: application/json

5.2 Subir Documento
- Ruta: POST /api/tfts/{tft}/documentos
- Descripción: Sube un documento a un TFT
- Autenticación: Bearer Token requerido
- Content-Type: multipart/form-data
- Body:
```json
{
   "file": "documento.pdf",
   "tipo": "anteproyecto",
   "descripcion": "Documento de anteproyecto"
}
```

5.3 Obtener Documento Específico
- Ruta: GET /api/tfts/{tft}/documentos/{documento}
- Descripción: Obtiene información de un documento específico
- Autenticación: Bearer Token requerido
- Content-Type: application/json

5.4 Actualizar Documento
- Ruta: PUT /api/tfts/{tft}/documentos/{documento}
- Descripción: Actualiza información de un documento
- Autenticación: Bearer Token requerido
- Content-Type: application/json

5.5 Eliminar Documento
- Ruta: DELETE /api/tfts/{tft}/documentos/{documento}
- Descripción: Elimina un documento
- Autenticación: Bearer Token requerido
- Content-Type: application/json

5.6 Subir Documento Individual
- Ruta: POST /api/tfts/{tft}/subir-documento-individual
- Descripción: Sube un documento individual con validaciones específicas
- Autenticación: Bearer Token requerido
- Content-Type: multipart/form-data

6. GESTIÓN DE ANTEPROYECTOS

6.1 Subir Anteproyecto
- Ruta: POST /api/tfts/{tft}/subir-anteproyecto
- Descripción: Sube el documento de anteproyecto

- Autenticación: Bearer Token requerido
- Content-Type: multipart/form-data
- Body:
```json
{
   "anteproyecto": "anteproyecto.pdf",
   "descripcion": "Descripción del anteproyecto"
}
```

6.2 Descargar Anteproyecto
- Ruta: GET /api/tfts/{tft}/descargar-anteproyecto
- Descripción: Descarga el documento de anteproyecto
- Autenticación: Bearer Token requerido
- Content-Type: application/octet-stream

6.3 Aprobar Anteproyecto
- Ruta: POST /api/tfts/{tft}/aprobar-anteproyecto
- Descripción: Aprueba un anteproyecto (solo comisión)
- Autenticación: Bearer Token requerido (Comisión)
- Content-Type: application/json
- Body:
```json
{
   "observaciones": "Anteproyecto aprobado sin observaciones"
}
```

6.4 Rechazar Anteproyecto
- Ruta: POST /api/tfts/{tft}/rechazar-anteproyecto
- Descripción: Rechaza un anteproyecto (solo comisión)
- Autenticación: Bearer Token requerido (Comisión)
- Content-Type: application/json
- Body:
```json
{
   "motivo_rechazo": "Falta documentación requerida",
   "fecha_limite_correccion": "2024-12-31"
}
```

7. GESTIÓN DE TRIBUNALES

7.1 Listar Tribunales
- Ruta: GET /api/tribunales
- Descripción: Obtiene lista de todos los tribunales
- Autenticación: Bearer Token requerido
- Content-Type: application/json

7.2 Crear Tribunal
- Ruta: POST /api/tribunales

- Descripción: Crea un nuevo tribunal
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
   "presidente_id": 1,
   "secretario_id": 2,
   "vocal_id": 3,
   "presidente_suplente_id": 4,
   "secretario_suplente_id": 5,
   "vocal_suplente_id": 6,
   "fecha": "2024-06-15",
   "hora_inicio": "09:00",
   "hora_fin": "14:00",
   "aula": "Aula 101",
   "plazas": 5,
   "numero": 1
}
```

7.3 Obtener Tribunal Específico
- Ruta: GET /api/tribunales/{id}
- Descripción: Obtiene información detallada de un tribunal
- Autenticación: Bearer Token requerido
- Content-Type: application/json

7.4 Actualizar Tribunal
- Ruta: PUT /api/tribunales/{id}
- Descripción: Actualiza información de un tribunal
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

7.5 Eliminar Tribunal
- Ruta: DELETE /api/tribunales/{id}
- Descripción: Elimina un tribunal
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

7.6 Buscar Profesores
- Ruta: GET /api/tribunales/search-profesores
- Descripción: Busca profesores para asignar a tribunales
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Query Parameters:
  - `q`: Término de búsqueda

7.7 Estadísticas de Tribunales
- Ruta: GET /api/tribunales/estadisticas
- Descripción: Obtiene estadísticas de tribunales
- Autenticación: Bearer Token requerido

- Content-Type: application/json

7.8 TFTs Disponibles
- Ruta: GET /api/tfts/disponibles
- Descripción: Obtiene TFTs disponibles para asignar a tribunales
- Autenticación: Bearer Token requerido
- Content-Type: application/json

7.9 Asignar TFT a Tribunal
- Ruta: POST /api/tribunales/{tribunal}/tfts/asignar
- Descripción: Asigna un TFT a un tribunal
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
   "tft_id": 1
}
```

7.10 Desasignar TFT de Tribunal
- Ruta: POST /api/tribunales/{tribunal}/tfts/desasignar
- Descripción: Desasigna un TFT de un tribunal
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
   "tft_id": 1
}
```

7.11 Programar Defensa
- Ruta: POST /api/tribunales/{tribunal}/tfts/programar-defensa
- Descripción: Programa la defensa de un TFT
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
   "tft_id": 1,
   "fecha_defensa": "2024-06-15",
   "hora_defensa": "10:00"
}
```

8. GESTIÓN DE CALENDARIOS

8.1 Listar Calendarios
- Ruta: GET /api/calendarios
- Descripción: Obtiene lista de todos los calendarios

- Autenticación: Bearer Token requerido
- Content-Type: application/json

8.2 Crear Calendario
- Ruta: POST /api/calendarios
- Descripción: Crea un nuevo calendario
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json
- Body:
```json
{
    "titulo": "Convocatoria 2024-2025",
    "fecha_inicio": "2024-09-01",
    "fecha_fin": "2025-06-30",
    "descripcion": "Calendario académico",
    "activo": true,
    "tipo_evento": "plazo_entrega",
    "titulacion": "GITT",
    "anio_academico": "2024/2025",
    "estudio": "gitt",
    "convocatoria": "ordinaria"
}
```

8.3 Obtener Calendario Específico
- Ruta: GET /api/calendarios/{id}
- Descripción: Obtiene información de un calendario específico
- Autenticación: Bearer Token requerido
- Content-Type: application/json

8.4 Actualizar Calendario
- Ruta: PUT /api/calendarios/{id}
- Descripción: Actualiza información de un calendario
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

8.5 Eliminar Calendario
- Ruta: DELETE /api/calendarios/{id}
- Descripción: Elimina un calendario
- Autenticación: Bearer Token requerido (Admin)
- Content-Type: application/json

9. GESTIÓN DE EVALUACIONES

9.1 Obtener Evaluación de TFT
- Ruta: GET /api/tfts/{tft}/evaluacion
- Descripción: Obtiene la evaluación de un TFT específico
- Autenticación: Bearer Token requerido (Tribunal)
- Content-Type: application/json

9.2 Calificar TFT

- Ruta: POST /api/tfts/{tft}/calificar
- Descripción: Califica un TFT (solo miembros del tribunal)
- Autenticación: Bearer Token requerido (Tribunal)
- Content-Type: application/json
- Body:
```json
{
   "calificacion": 9.5,
   "observaciones": "Excelente trabajo",
   "criterios": {
      "presentacion": 9,
      "contenido": 10,
      "metodologia": 9,
      "resultados": 10
   }
}
```

10. GESTIÓN DE SEGUIMIENTOS

10.1 Listar Seguimientos
- Ruta: GET /api/seguimientos
- Descripción: Obtiene lista de seguimientos
- Autenticación: Bearer Token requerido
- Content-Type: application/json

10.2 Crear Seguimiento
- Ruta: POST /api/seguimientos
- Descripción: Crea un nuevo seguimiento
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Body:
```json
{
   "tft_id": 1,
   "fecha": "2024-01-15",
   "descripcion": "Reunión de seguimiento",
   "observaciones": "Progreso satisfactorio"
}
```

10.3 Obtener Seguimiento Específico
- Ruta: GET /api/seguimientos/{id}
- Descripción: Obtiene información de un seguimiento específico
- Autenticación: Bearer Token requerido
- Content-Type: application/json

10.4 Actualizar Seguimiento
- Ruta: PUT /api/seguimientos/{id}
- Descripción: Actualiza información de un seguimiento
- Autenticación: Bearer Token requerido

- Content-Type: application/json

10.5 Eliminar Seguimiento
- Ruta: DELETE /api/seguimientos/{id}
- Descripción: Elimina un seguimiento
- Autenticación: Bearer Token requerido
- Content-Type: application/json

11. ENDPOINTS DE PRUEBA

11.1 Test de Conectividad
- Ruta: GET /api/test-search
- Descripción: Endpoint de prueba para verificar conectividad
- Autenticación: No requerida
- Content-Type: application/json
- Respuesta (200):

```json
[
  {
    "id": 1,
    "name": "Test Profesor",
    "email": "test@test.com",
    "role": "profesor"
  }
]
```

11.2 Test de API
- Ruta: GET /api/test
- Descripción: Verifica que la API funciona correctamente
- Autenticación: Bearer Token requerido
- Content-Type: application/json
- Respuesta (200):

```json
{
  "message": "API funcionando correctamente"
}
```

12. CÓDIGOS DE RESPUESTA HTTP

12.1 Respuestas Exitosas
- 200 OK: Operación exitosa
- 201 Created: Recurso creado exitosamente
- 204 No Content: Operación exitosa sin contenido

12.2 Respuestas de Error del Cliente
- 400 Bad Request: Datos de entrada inválidos
- 401 Unauthorized: No autenticado
- 403 Forbidden: No autorizado
- 404 Not Found: Recurso no encontrado

- 422 Unprocessable Entity: Datos de validación incorrectos

12.3 Respuestas de Error del Servidor
- 500 Internal Server Error: Error interno del servidor

13. AUTENTICACIÓN

13.1 Headers Requeridos
Para endpoints protegidos, incluir en el header:

```
Authorization: Bearer {token}
Content-Type: application/json
```

13.2 Formato de Token
Los tokens se obtienen del endpoint de login y deben incluirse en todas las peticiones protegidas.

13.3 Roles y Permisos
- admin: Acceso completo a todas las funcionalidades
- comision: Gestión de TFTs y aprobaciones
- profesor/tutor: Gestión de TFTs asignados
- estudiante: Acceso limitado a sus propios TFTs

2. Código

El código usado en este proyecto puede encontrarse en **Github** en el siguiente enlace: https://github.com/FreddyLopez208/sistema-gestion-tft-eite

Hay que tener en cuenta que al ser un proyecto que debe tener seguridad, se ha establecido en privado dicho repositorio, pero se puede acceder al código pidiendo la autorización.

3. Implementación de los tests

Tests de autenticación y autorización

AuthControllerTest.php

Propósito: Tiene el propósito de validar el sistema completo de autenticación y gestión de usuarios.

Métodos Principales:

test_login_exitoso_con_credenciales_validas()

- **Funcionalidad:** Verifica que un usuario con credenciales válidas puede autenticarse exitosamente. Crea un usuario de prueba con DNI y contraseña, realiza una petición POST al endpoint /api/login y valida que la respuesta incluye el token de autenticación, datos del usuario y mensaje de éxito. También verifica que la estructura JSON de la respuesta contiene todos los campos requeridos.

test_login_falla_con_dni_inexistente()

- **Funcionalidad:** Valida que el sistema rechaza correctamente intentos de login con DNIs que no existen en la base de datos. Realiza una petición POST con un DNI inexistente y verifica que se devuelve un error 422 con el mensaje apropiado indicando credenciales incorrectas.

test_login_falla_con_usuario_inactivo()

- **Funcionalidad:** Verifica que los usuarios marcados como inactivos no pueden autenticarse. Crea un usuario con el campo active en false, intenta el login y valida que se devuelve un error 403 con el mensaje indicando que el usuario está inactivo.

test_logout_exitoso()

- **Funcionalidad:** Prueba el proceso de logout de un usuario autenticado. Crea un token de autenticación para un usuario, realiza una petición POST al endpoint /api/logout con el token en los headers y verifica que se devuelve un mensaje de logout exitoso.

test_actualizar_perfil_usuario()

- **Funcionalidad:** Valida la actualización del perfil de usuario. Autentica un usuario, envía datos de actualización (nombre y email) al endpoint /api/profile y verifica que los cambios se aplican correctamente tanto en la respuesta como en la base de datos.

test_cambiar_password_exitoso()

- **Funcionalidad:** Prueba el cambio de contraseña de un usuario. Autentica un usuario, envía la contraseña actual y la nueva contraseña con su confirmación al endpoint /api/change-password y verifica que el cambio se procesa exitosamente.

RolesPermisosTest.php

Propósito: Valida el sistema de roles y permisos en toda la aplicación.

Métodos Principales:

test_admin_tiene_acceso_completo()

- **Funcionalidad:** Verifica que los usuarios con rol 'admin' tienen acceso completo a todas las funcionalidades del sistema. Crea un usuario administrador y prueba el acceso a todos los endpoints principales (TFTs, usuarios, calendarios, tribunales) validando que todas las peticiones devuelven respuestas exitosas.

test_estudiante_acceso_limitado()

- **Funcionalidad:** Valida que los estudiantes tienen acceso limitado según las políticas de seguridad. Crea un usuario estudiante y verifica que solo puede acceder a sus propios TFTs, no puede gestionar usuarios ni calendarios, y tiene acceso restringido a ciertas funcionalidades administrativas.

test_profesor_acceso_intermedio()

- **Funcionalidad:** Prueba que los profesores tienen acceso intermedio al sistema. Verifica que pueden gestionar TFTs asignados, acceder a calendarios para consulta, pero tienen restricciones en la gestión de usuarios y configuraciones administrativas.

Tests de gestión de TFT

TftTest.php

Propósito: Valida todas las operaciones CRUD y flujos de trabajo relacionados con los TFT.

Métodos Principales:

admin_can_view_all_tfts()

- **Funcionalidad:** Verifica que los administradores pueden ver todos los TFTs en el sistema. Crea un usuario admin, autentica la sesión y realiza una petición GET al endpoint /api/tfts. Valida que la respuesta incluye todos los TFTs con la estructura JSON correcta que contiene campos como id, título, descripción, estado, estudiante_id y tutor_id.

student_can_view_own_tfts()

- **Funcionalidad:** Valida que los estudiantes solo pueden ver sus propios TFTs. Crea un estudiante y un TFT asociado a ese estudiante, autentica la sesión del estudiante y verifica que la respuesta del endpoint /api/tfts solo incluye el TFT del estudiante autenticado, no otros TFTs del sistema.

admin_can_create_tft()

- **Funcionalidad:** Prueba la creación de TFTs por parte de administradores. Crea usuarios admin, estudiante y tutor, autentica al admin y envía datos de un nuevo TFT al endpoint POST /api/tfts. Valida que el TFT se crea exitosamente con estado 201, que la respuesta contiene los datos correctos y que el registro se almacena en la base de datos con las relaciones apropiadas.

tft_creation_validates_required_fields()

- **Funcionalidad:** Valida que el sistema requiere campos obligatorios para crear un TFT. Autentica un admin y envía una petición POST vacía al endpoint /api/tfts. Verifica que se devuelve un error 422 con validaciones específicas para los campos requeridos como título, descripción, estudiante_id y tutor_id.

student_can_update_own_tft_in_borrador_state()

- **Funcionalidad:** Verifica que los estudiantes pueden actualizar sus TFTs solo cuando están en estado 'borrador'. Crea un TFT en estado borrador para un estudiante, autentica al estudiante y envía datos de actualización al endpoint PUT /api/tfts/{id}. Valida que la actualización se procesa exitosamente y que los cambios se reflejan en la base de datos.

student_cannot_update_tft_not_in_borrador_state()

- **Funcionalidad:** Valida que los estudiantes no pueden actualizar TFTs que no están en estado 'borrador'. Crea un TFT en estado 'en_desarrollo' para un estudiante, autentica al estudiante e intenta actualizar el TFT. Verifica que se devuelve un error apropiado indicando que la actualización no está permitida en ese estado.

TftFlujoCompletoTest.php

Propósito: Valida el flujo completo de un TFT desde su creación hasta la evaluación final.

Métodos Principales:

test_flujo_completo_tft_desde_anteproyecto_hasta_calificacion()

- **Funcionalidad:** Simula el flujo completo de un TFT desde su presentación como anteproyecto hasta la calificación final. El test se divide en múltiples fases: 1) El estudiante presenta el anteproyecto y sube documentación inicial, 2) La comisión revisa y aprueba la propuesta, 3) El director cambia el estado a 'en_desarrollo', 4) El estudiante entrega documentación final (memoria, código, presentación), 5) Se asigna tribunal y programa defensa, 6) Se realiza la defensa y evaluación. En cada fase valida las transiciones de estado, la subida de documentos requeridos y las autorizaciones necesarias.

test_subida_documentacion_en_todos_los_estados()

- **Funcionalidad:** Valida que la subida de documentación funciona correctamente en todos los estados del TFT. Crea un TFT y prueba la subida de diferentes tipos de documentos (anteproyecto, borrador, memoria, código, presentación) en los estados correspondientes. Verifica que cada documento se almacena correctamente, que se validan los tipos de archivo permitidos y que se mantiene la integridad de los datos.

TftPolicyTest.php (Unit Test)

Propósito: Valida las políticas de autorización específicas para TFTs.

Métodos Principales:

admin_can_view_any_tft()

- **Funcionalidad:** Verifica que la política TftPolicy permite a los administradores ver cualquier TFT. Crea un usuario admin y un TFT, instancia la política TftPolicy y llama al método view() con el admin y el TFT. Valida que el método retorna true, confirmando que el admin tiene permisos para ver el TFT.

student_can_view_own_tft()

- **Funcionalidad:** Valida que los estudiantes pueden ver solo sus propios TFTs según la política. Crea un estudiante y un TFT asociado a ese estudiante, instancia la política y verifica que el método view() retorna true cuando el estudiante intenta ver su propio TFT.

student_cannot_view_other_student_tft()

- **Funcionalidad:** Verifica que la política impide que los estudiantes vean TFTs de otros estudiantes. Crea dos estudiantes y un TFT asociado al primer estudiante, instancia la política y valida que el método view() retorna false cuando el segundo estudiante intenta ver el TFT del primero.

tutor_can_update_assigned_tft()

- **Funcionalidad:** Valida que los tutores pueden actualizar TFTs que les han sido asignados. Crea un tutor y un TFT asignado a ese tutor, instancia la política y verifica que el método update() retorna true cuando el tutor intenta actualizar su TFT asignado.

Tests de gestión de calendarios

CalendarioTest.php

Propósito: Valida la gestión completa de calendarios académicos y eventos.

Métodos Principales:

admin_can_view_all_calendarios()

- **Funcionalidad:** Verifica que los administradores pueden ver todos los calendarios del sistema. Crea un usuario admin y múltiples calendarios de prueba, autentica al admin y realiza una petición GET al endpoint /api/calendarios. Valida que la respuesta incluye todos los calendarios con la estructura JSON correcta que contiene campos como id, nombre, descripción, fecha_inicio, fecha_fin y activo.

admin_can_create_calendario()

- **Funcionalidad:** Prueba la creación de calendarios por parte de administradores. Autentica un admin y envía datos de un nuevo calendario al endpoint POST /api/calendarios. Valida que el calendario se crea exitosamente con estado 201, que la respuesta contiene los datos correctos y que el registro se almacena en la base de datos con las fechas y estado activo apropiados.

calendario_fecha_fin_must_be_after_fecha_inicio()

- **Funcionalidad:** Valida que el sistema impide crear calendarios con fechas de fin anteriores a las fechas de inicio. Autentica un admin y envía datos de calendario con fecha_fin anterior a fecha_inicio. Verifica que se devuelve un error 422 con validación específica para el campo fecha_fin.

only_one_calendario_can_be_active()

- **Funcionalidad:** Verifica que solo puede haber un calendario activo en el sistema en un momento dado. Crea un calendario activo, luego intenta crear un segundo calendario activo. Valida que el sistema maneja correctamente esta restricción de negocio, ya sea desactivando automáticamente el anterior o rechazando la creación del nuevo.

calendario_events_can_be_managed()

- **Funcionalidad:** Prueba la gestión de eventos dentro de los calendarios. Crea un calendario y prueba la creación, actualización y eliminación de eventos asociados. Valida que los eventos se almacenan correctamente con sus fechas, horas, descripciones y tipos de evento, y que las operaciones CRUD funcionan según las expectativas.

CalendarioEndpointsTest.php

Propósito: Valida específicamente los *endpoints* de la API de calendarios.

Métodos Principales:

test_get_calendarios_endpoint()

- **Funcionalidad:** Prueba el endpoint GET /api/calendarios para obtener la lista de calendarios. Crea múltiples calendarios de prueba, autentica un usuario con permisos y realiza la petición. Valida que la respuesta incluye todos los calendarios con paginación correcta, filtros aplicados y estructura JSON apropiada.

test_create_calendario_endpoint()

- **Funcionalidad:** Valida el endpoint POST /api/calendarios para crear nuevos calendarios. Autentica un usuario con permisos de administración y envía datos de un nuevo calendario. Verifica que se crea exitosamente, que se aplican todas las validaciones de negocio y que la respuesta contiene los datos del calendario creado.

Tests de sistema de chat

ChatTest.php

Propósito: Valida el sistema completo de mensajería y conversaciones.

Métodos Principales:

user_can_get_conversations()

- **Funcionalidad:** Verifica que los usuarios pueden obtener sus conversaciones. Crea un usuario y una conversación asociada, autentica al usuario y realiza una petición GET al endpoint /api/chat/conversations. Valida que la respuesta incluye todas las conversaciones del usuario con la estructura JSON correcta que contiene id, nombre, created_at y updated_at.

user_can_start_conversation()

- **Funcionalidad:** Prueba la creación de nuevas conversaciones entre usuarios. Crea dos usuarios, autentica al primero y envía una petición POST al endpoint /api/chat/conversations/start con el ID del segundo usuario. Valida que se crea la conversación exitosamente, que ambos usuarios están asociados a la conversación y que la respuesta contiene los datos de la conversación creada.

user_cannot_start_conversation_with_self()

- **Funcionalidad:** Valida que un usuario no puede iniciar una conversación consigo mismo. Crea un usuario, autentica la sesión e intenta crear una conversación usando el mismo ID de usuario. Verifica que se devuelve un error 422 indicando que la operación no está permitida.

user_can_send_message()

- **Funcionalidad:** Prueba el envío de mensajes en conversaciones. Crea un usuario y una conversación asociada, autentica al usuario y envía un mensaje al endpoint POST /api/chat/conversations/{id}/messages. Valida que el mensaje se almacena correctamente en la base de datos, que la respuesta contiene los datos del mensaje y que se mantienen las relaciones apropiadas entre usuario, conversación y mensaje.

user_cannot_send_message_to_unrelated_conversation()

- **Funcionalidad:** Verifica que los usuarios no pueden enviar mensajes a conversaciones en las que no participan. Crea dos usuarios y una conversación asociada solo al segundo usuario, autentica al primer usuario e intenta enviar un mensaje a esa conversación. Valida que se devuelve un error 403 indicando que no tiene permisos.

message_requires_body()

- **Funcionalidad:** Valida que los mensajes requieren contenido. Crea un usuario y una conversación, autentica al usuario e intenta enviar un mensaje vacío. Verifica que se devuelve un error 422 con validación específica para el campo 'body'.

ChatControllerSimpleTest.php

Propósito: Valida funcionalidades específicas del controlador de chat.

Métodos Principales:

test_send_message_with_notification()

- **Funcionalidad:** Prueba que el envío de mensajes genera notificaciones apropiadas. Crea usuarios y una conversación, envía un mensaje y verifica que se disparan los eventos de notificación correspondientes, asegurando que los usuarios reciben notificaciones en tiempo real de nuevos mensajes.

Tests de gestión de usuarios

UserTest.php

Propósito: Valida todas las operaciones relacionadas con la gestión de usuarios.

Métodos Principales:

admin_can_view_all_users()

- **Funcionalidad:** Verifica que los administradores pueden ver todos los usuarios del sistema. Crea múltiples usuarios con diferentes roles, autentica un admin y realiza una petición GET al endpoint /api/users. Valida que la respuesta incluye todos los usuarios con información completa incluyendo roles, departamentos y estados.

admin_can_create_user()

- **Funcionalidad:** Prueba la creación de usuarios por parte de administradores. Autentica un admin y envía datos de un nuevo usuario al endpoint POST /api/users. Valida que el usuario se crea exitosamente, que la contraseña se hashea correctamente, que se asignan los roles apropiados y que se envían emails de bienvenida.

user_creation_validates_required_fields()

- **Funcionalidad:** Valida que la creación de usuarios requiere campos obligatorios. Autentica un admin y envía una petición POST vacía al endpoint /api/users. Verifica

que se devuelve un error 422 con validaciones específicas para campos como nombre, email, DNI, rol y departamento.

admin_can_update_user()

- **Funcionalidad:** Prueba la actualización de usuarios por parte de administradores. Crea un usuario, autentica un admin y envía datos de actualización al endpoint PUT /api/users/{id}. Valida que los cambios se aplican correctamente, que se mantienen las relaciones con roles y departamentos, y que la respuesta contiene los datos actualizados.

admin_can_deactivate_user()

- **Funcionalidad:** Verifica que los administradores pueden desactivar usuarios. Crea un usuario activo, autentica un admin y envía una petición para desactivar al usuario. Valida que el campo active se cambia a false, que el usuario no puede autenticarse después de la desactivación y que se mantiene la integridad de los datos.

CreacionUsuariosPorTipoTest.php

Propósito: Valida la creación de usuarios según diferentes tipos y roles.

Métodos Principales:

test_crear_estudiante_con_datos_completos()

- **Funcionalidad:** Prueba la creación específica de usuarios estudiantes. Autentica un admin y envía datos completos de un estudiante incluyendo información académica, departamento y datos personales. Valida que el estudiante se crea correctamente con el rol apropiado, que se asignan las relaciones necesarias y que se generan las credenciales de acceso.

test_crear_profesor_con_departamento()

- **Funcionalidad:** Valida la creación de usuarios profesores con asignación de departamento. Crea un departamento, autentica un admin y envía datos de un profesor incluyendo el departamento asignado. Verifica que el profesor se

crea con el rol correcto, que se establece la relación con el departamento y que se configuran los permisos apropiados.

test_crear_comision_con_permisos_especiales()

- **Funcionalidad:** Prueba la creación de usuarios de comisión con permisos especiales. Autentica un admin y crea un usuario con rol 'comision'. Valida que se asignan los permisos especiales para gestión de TFTs, que puede aprobar y rechazar propuestas, y que tiene acceso a funcionalidades administrativas específicas.

Tests de gestión de tribunales

TribunalTest.php

Propósito: Valida la gestión completa de tribunales de evaluación.

Métodos Principales:

admin_can_view_all_tribunales()

- **Funcionalidad:** Verifica que los administradores pueden ver todos los tribunales del sistema. Crea múltiples tribunales con diferentes composiciones, autentica un admin y realiza una petición GET al endpoint /api/tribunales. Valida que la respuesta incluye todos los tribunales con información completa de presidente, secretario, vocal y TFTs asignados.

admin_can_create_tribunal()

- **Funcionalidad:** Prueba la creación de tribunales por parte de administradores. Crea usuarios para los roles del tribunal (presidente, secretario, vocal), autentica un admin y envía datos del tribunal al endpoint POST /api/tribunales. Valida que el tribunal se crea exitosamente, que se establecen las relaciones correctas y que se asignan los permisos apropiados a cada miembro.

tribunal_creation_validates_members()

- **Funcionalidad:** Valida que la creación de tribunales requiere miembros válidos. Autentica un admin e intenta crear un tribunal con usuarios inexistentes o con roles inapropiados. Verifica que se devuelven errores de validación específicos para cada campo requerido y que se validan las restricciones de roles.

tribunal_can_be_assigned_to_tft()

- **Funcionalidad:** Prueba la asignación de tribunales a TFTs. Crea un tribunal y un TFT, autentica un admin y asigna el tribunal al TFT. Valida que la asignación se establece correctamente, que el TFT cambia a estado 'tribunal_asignado' y que los miembros del tribunal pueden acceder al TFT asignado.

tribunal_members_can_evaluate_tft()

- **Funcionalidad:** Verifica que los miembros del tribunal pueden evaluar TFTs asignados. Crea un tribunal asignado a un TFT, autentica cada miembro del tribunal y permite que realicen evaluaciones. Valida que las evaluaciones se almacenan correctamente, que se calculan las calificaciones finales y que el TFT cambia a estado 'evaluado' cuando se completan todas las evaluaciones.

Tests del frontend – servicios

auth-roles.service.spec.ts

Propósito: Valida el servicio de autenticación y gestión de roles en el *frontend*.

Métodos Principales:

should return correct role for admin user

- **Funcionalidad:** Verifica que el servicio AuthService identifica correctamente el rol de usuario administrador. Configura un usuario admin en localStorage, reinicializa el servicio y valida que las propiedades userRole y hasRole() retornan los valores correctos para el rol 'admin', confirmando que el sistema de roles funciona apropiadamente en el frontend.

should verify multiple roles correctly

- **Funcionalidad:** Prueba la verificación de múltiples roles simultáneamente. Configura un usuario admin y valida que los métodos hasRole() y hasAnyRole() funcionan correctamente con arrays de roles. Verifica que retorna true cuando el usuario tiene al menos uno de los roles especificados y false cuando no tiene ninguno de los roles del array.

should update user profile correctly

- **Funcionalidad:** Valida la actualización del perfil de usuario desde el frontend. Configura datos de perfil actualizados, llama al método updateProfile() del servicio y simula la respuesta del servidor. Verifica que la respuesta se procesa correctamente, que los datos se actualizan en localStorage y que se mantiene la consistencia de los datos del usuario.

should change password correctly

- **Funcionalidad:** Prueba el cambio de contraseña desde el frontend. Configura datos de cambio de contraseña (contraseña actual, nueva contraseña y confirmación), llama al método changePassword() del servicio y simula la respuesta del servidor. Valida que la petición HTTP se realiza correctamente con los datos apropiados y que se procesa la respuesta exitosa.

tft-flujo-completo.service.spec.ts

Propósito: Valida el flujo completo de TFT desde la perspectiva del frontend.

Métodos Principales:

should complete full TFT workflow from anteproyecto to evaluado

- **Funcionalidad:** Simula el flujo completo de un TFT desde el frontend. Crea un nuevo TFT con estado 'anteproyecto', luego simula las transiciones de estado siguiendo el flujo correcto: propuesta_presentada, propuesta_aceptada, en_desarrollo, documentacion_entregada, tribunal_asignado, defensa_programada, defendido y evaluado. En cada paso valida que las peticiones HTTP se realizan correctamente y que las respuestas se procesan apropiadamente.

should validate document uploads for each TFT state

- **Funcionalidad:** Valida la subida de documentos en diferentes estados del TFT. Simula la subida de diferentes tipos de documentos (anteproyecto, borrador, memoria, código, presentación) en los estados correspondientes del TFT. Verifica que cada tipo de documento se asocia correctamente con el estado apropiado y que las peticiones HTTP incluyen los datos correctos.

should verify all required documents before final evaluation

- **Funcionalidad:** Verifica que todos los documentos requeridos están presentes antes de la evaluación final. Simula un TFT con todos los documentos necesarios (anteproyecto, memoria, código, presentación) y valida que el sistema reconoce que todos los documentos están completos. Confirma que el TFT puede proceder a la evaluación final cuando todos los documentos requeridos están presentes.

should validate state transitions are in correct order

- **Funcionalidad:** Valida que las transiciones de estado del TFT siguen el orden correcto. Simula cada transición de estado en el orden predefinido y verifica que cada cambio se procesa exitosamente. Confirma que el sistema mantiene la integridad del flujo de trabajo y que no se permiten saltos de estado inválidos.

should prevent invalid state transitions

- **Funcionalidad:** Verifica que el sistema impide transiciones de estado inválidas. Intenta cambiar el estado de un TFT de 'anteproyecto' directamente a 'defendido', simulando un salto de estado no permitido. Valida que el sistema devuelve un error 422 indicando que la transición de estado es inválida, confirmando que se mantienen las reglas de negocio del flujo de trabajo.

Tests end-to-end (E2E)

tft-workflow.e2e-spec.ts

Propósito: Valida flujos completos de usuario desde la interfaz web.

Métodos Principales:

Complete TFT workflow from creation to evaluation

- **Funcionalidad:** Simula el flujo completo de un TFT desde la interfaz de usuario. Inicia con el login como administrador, navega a la gestión de TFTs, crea un nuevo TFT completando el formulario, edita el TFT para actualizar información, envía la propuesta, cambia a usuario comisión para aprobar la propuesta, cambia a usuario estudiante para finalizar el TFT, presenta la defensa subiendo documentos y completa el proceso de evaluación. En cada paso valida que los elementos de la interfaz responden correctamente y que los datos se procesan según las expectativas.

TFT rejection workflow

- **Funcionalidad:** Prueba el flujo de rechazo de un TFT. Crea un TFT con información incompleta, envía la propuesta, cambia a usuario comisión y rechaza la propuesta proporcionando razones de rechazo y fecha límite para correcciones. Valida que el sistema maneja correctamente el rechazo, que se registran las razones y que se establecen las fechas de corrección apropiadas.

TFT search and filtering

- **Funcionalidad:** Valida las funcionalidades de búsqueda y filtrado de TFTs. Navega a la lista de TFTs, realiza búsquedas por texto específico, aplica filtros por estado, verifica que los resultados se muestran correctamente y limpia los filtros para confirmar que la funcionalidad de reset funciona apropiadamente.

TFT document upload

- **Funcionalidad:** Prueba la subida de documentos desde la interfaz de usuario. Crea un TFT y sube diferentes tipos de documentos (anteproyecto, memoria) usando los controles de archivo de la interfaz. Valida que los archivos se suben correctamente, que se muestran en la lista de documentos y que se mantiene la integridad de los datos.

convocatorias-calendar-workflow.e2e-spec.ts

Propósito: Valida flujos de trabajo relacionados con convocatorias y calendarios.

Métodos Principales:

Complete convocatoria workflow

- **Funcionalidad:** Simula el flujo completo de gestión de convocatorias. Inicia con login como administrador, navega a la gestión de convocatorias, crea una nueva convocatoria con fechas y eventos específicos, programa eventos del calendario, asigna TFTs a la convocatoria y valida que todo el flujo funciona correctamente desde la perspectiva del usuario final.

Integración y comunicación entre tests

WorkflowTest.php

Propósito: Valida la integración entre diferentes módulos del sistema.

Métodos Principales:

test_integration_tft_calendario_tribunal()

- **Funcionalidad:** Prueba la integración completa entre TFTs, calendarios y tribunales. Crea un calendario activo, un TFT asociado a ese calendario, un tribunal y asigna el tribunal al TFT. Valida que todas las relaciones se establecen correctamente, que los eventos del calendario se reflejan en el TFT, que el tribunal puede acceder al TFT asignado y que el flujo completo funciona sin errores de integración.

test_notification_system_integration()

- **Funcionalidad:** Valida la integración del sistema de notificaciones con otros módulos. Simula diferentes eventos del sistema (creación de TFT, cambio de estado, asignación de tribunal, envío de mensajes) y verifica que se generan las notificaciones apropiadas, que se envían a los usuarios correctos y que el sistema de notificaciones funciona en conjunto con todos los módulos.

ProfesorSearchTest.php

Propósito: Valida funcionalidades específicas de búsqueda y filtrado.

Métodos Principales:

test_search_profesores_by_departamento()

- **Funcionalidad:** Prueba la búsqueda de profesores por departamento. Crea múltiples profesores en diferentes departamentos, realiza búsquedas específicas por departamento y valida que los resultados incluyen solo los profesores del departamento especificado, que la paginación funciona correctamente y que los filtros se aplican apropiadamente.