Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación para la gestión de tribunales de TFG/TFM en la ETSINF

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Parreño Collado, Iván

Tutor: Pelechano Ferragud, Vicente

2019/2020

Resumen

El proyecto que a continuación se presenta tiene como objetivo el desarrollo de una aplicación web que permita al usuario, crear y gestionar los tribunales de TFG y TFM de la escuela, centrándose en la rapidez, eficiencia, escalabilidad y usabilidad de esta. El sistema a partir de los datos referentes a profesores, titulaciones y las ramas de estas últimas, almacenados en la base de datos, permite realizar al usuario, en este caso el Jefe de Estudios, crear automáticamente todos los tribunales asociados a una convocatoria de TFG y TFM. Además, permite la gestión de los tribunales, pudiendo en caso de que así lo necesitara el usuario, crear tribunales a mano o gestionar a los profesores asociados a dicho tribunal. También ofrece otras funcionalidades que facilitan diversas tareas relacionadas con profesores, ramas, titulaciones, gestión de usuarios, etc...

El desarrollo de esta aplicación web se ha realizado siguiendo un proceso de desarrollo software iterativo de forma incremental, por medio de diferentes versiones del sistema. Esto lo que ha conllevado es que el Jefe de Estudios haya podido probar y validar las diferentes versiones del sistema, facilitando la tarea de desarrollo.

Las tecnologías usadas en el desarrollo de este sistema han sido un Framework de PHP, y un Framework de JavaScript. El Stack utilizado ha sido desde Visual Estudio Code, Git, Homestead y Gitlab como repositorio.

Palabras clave: PHP, aplicación web, JavaScript, escalabilidad, desarrollo iterativo incremental, gestión de los tribunales, Visual Estudio Code, Git, Homestead, Gitlab, Stack.

Abstract

The project presented below aims at developing a web application which allows the user to create and manage dissertations tribunal focusing on its swiftness, efficiency, scalability and usability. This system lets the user, in this case the Chief of Studies, create all the tribunals associated with an announcement automatically based on teachers' data, degrees and their branches of knowledge that are previously stored up in the database. Moreover, it permits the administration of the tribunals being able to create tribunals or manage their associated teachers manually if it's needed. Also, it offers other uses which facilitate several tasks related to teachers, user management, degrees and their branches of knowledge, etc...

The development of this web application has been carried out following an iterative software development process incrementally, through different versions of the system. This fact entails that the Chief of Studies has been able to test and validate different versions of the system, making easier the development task.

A PHP Framework and a JavaScript Framework were the technology utilized in the development task. In the other hand, the Stack used were Visual Estudio Code, Git, Homestead and Gitlab as repository.

Keywords: PHP, web application, JavaScript, scalability, iterative software development process incrementally, management of dissertations tribunal, Visual Estudio Code, Git, Homestead,Gitlab, Stack.

Glosario

- **PHP:** Hypertext Preprocessor. Es un lenguaje de programación de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.
- **Laravel:** Es un Framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5 y PHP 7. Su filosofía es desarrollar código PHP de forma elegante y simple**.**
- **REST:** Representational state transfer. Es una arquitectura software para enviar u obtener datos mediante el protocolo HTTP.
- **JSON:** JavaScript Object Notation. Es un formato ligero de intercambio de datos.
- **Eloquent:** Es un sistema que nos permite llevar la capa de persistencia en bases de datos por medio de objetos.
- **Back-End:** Es la parte del desarrollo web encargada de que toda la lógica de una página web funcione.
- **Front-End:** Es la parte que ve el usuario, es decir, donde va el diseño y los elementos gráficos de la página web.
- **API:** Application programming interface. Es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software.
- **GIT:** Es un software de control de versiones diseñado por Linus Torvalds.
- **Vuetify:** Es un Framework que combina la potencia del popular VueJs con la estética de Material Desing.
- **Vue.js:** Es un Framework progresivo para construir interfaces de usuario.
- **Framework:** Es un conjunto estandarizado de conceptos, prácticas y criterios para enforcar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Tabla de contenidos

Tabla de figuras

1. Introducción

El presente proyecto tiene como objetivo el desarrollo de una aplicación web para la creación y gestión de tribunales de TFG y TFM, que facilite al usuario de forma sencilla y rápida la creación de los tribunales de todas las titulaciones y sus especialidades/ramas asociadas a estas en el caso de que las tuvieran, además de diversas tareas asociadas a las convocatorias y profesores de la Escuela Técnica Superior de Ingeniería Informática.

1.1 Motivación

La realización de este proyecto surge a partir de la necesidad de automatizar de forma eficiente la creación y gestión de tribunales y de todas las tareas asociadas a estos.

Actualmente el software que se utiliza para llevar a cabo todas las tareas de mantenimiento y creación de los tribunales no cubre todas las necesidades que el usuario que utiliza el sistema necesita y requiere para automatizar todas las tareas, los profesores no se almacenan en ninguna base de datos, por lo que cada vez que se quiere crear un tribunal en una convocatorias concreta, se debe introducir en el sistema un fichero con los datos de todos los profesores de la Escuela, por lo que resulta muy ineficiente. Tampoco hay un registro de las titulaciones ni de sus ramas, solo pudiendo crear tribunales de la titulación Ingeniería Informática. Además de ser una aplicación de escritorio que necesita ser ejecutada en el equipo para poder ser usada.

Por estas razones y otras varias se ha visto necesario actualizar y dotar de una interfaz de usuario más amigable para llevar a cabo todas estas tareas. El sistema que se va a desarrollar en el presente proyecto, permitirá ahorrar tiempo y ofrecer una alternativa en la web, más moderna, eficiente, escalable y usable. Con este software se pretende que el usuario pueda, actualizar profesores, titulaciones, convocatorias e incluso delegar en distintos usuarios la creación de los tribunales, y poder hacerlo desde cualquier parte, despreocupándose de tener instalado el sistema en su equipo.

1.2 Objetivos

El objetivo principal es la realización de una aplicación web, sencilla y fácil de mantener, a la par que fácil de usar e intuitiva. Para ello se tendrán en cuenta los detalles estéticos y la experiencia del usuario al utilizar la aplicación. Es necesario que sea usable, es decir, la facilidad que tendrá el usuario para interactuar con el sistema y llevar a cabo las tareas sin la ayuda de manuales. Esto se podrá conseguir usando botones e iconos para el fácil reconocimiento de dichas tareas. Este objetivo se consigue utilizando librerías y estilos CSS, usando una distribución de los elementos y colores acordes a las guías propuestas por el Marerial Design usado.

Otros de los objetivos fundamentales es que la aplicación sea escalable. Este punto es muy importante puesto que, si en un futuro no muy lejano la ESTINF añade una nueva titulación a su oferta educativa, esta pueda ser incluida en el sistema y se puedan crear los tribunales asociados con los profesores que imparten clase en esa nueva titulación.

Otros de los objetivos necesarios del sistema son también es necesidad de poder crear profesores, ramas, convocarías y usuarios utilizando un sencillo formulario, otorgándole sencillez y eficiencia a la realización de estas tareas.

Un objetivo no menos importante es una buena estructura del código fuente, utilizando técnicas y buenas prácticas de Clean Code. Laravel es un Framework basado en código abierto para el desarrollo de aplicaciones web en PHP que sigue un patrón de arquitectura MVC (Model View Controller) que utiliza tres componentes y separa la lógica de la aplicación de la lógica de la vista en una aplicación. Este Framework facilita mucho el trabajo de desarrollo proporcionando múltiples herramientas para escribir código limpio.

1.3 Estructura

El presente proyecto se divide en los siguientes bloques:

- En el primer punto, se explica brevemente el problema a resolver y las motivaciones que han impulsado la creación del proyecto y los objetivos del mismo.
- En el segundo, se hablará sobre varias aplicaciones similares a la aplicación que se desarrollará en este proyecto.
- En el tercer punto se hará un pequeño análisis del problema, y las diferentes viabilidades técnicas a la hora de comenzar con el proyecto.
- En el cuarto punto, se describirá el proceso de desarrollo software usado en el proyecto y las diferentes fases que tiene.
- En el quinto punto, se definen los requisitos funcionales y no funcionales y los diferentes casos de uso de la aplicación.
- En el sexto punto, se define el diseño, tanto de la interfaz de usuario como del modelo de datos y la arquitectura usada.
- En el séptimo punto Se hablará sobre las herramientas y la tecnología usada para el desarrollo del proyecto.
- En el octavo punto, se explica cómo se ha implementado la aplicación y la estructura del proyecto. También se mostrará un fragmento de la documentación API.
- En el noveno punto, se enseñará como ha quedado la aplicación final y una breve explicación de la funcionalidad que ofrece.
- Para finalizar en los dos últimos puntos se incluye un punto de conclusiones donde se hace una pequeña reflexión acerca de los problemas que han ido sucediendo en el desarrollo y como se han ido solucionando. Y para acabar se incluye una bibliografía con todas las referencias a las fuentes consultadas.

2. Estado del Arte

Existen multitud de aplicaciones en el mercado para realizar la gestión de documentos de todo tipo, ya sean documentos Excel con extensión CSV o documentos de texto plano (TXT). El problema al que daremos solución en este proyecto pertenece un área de aplicación muy específico, concretamente al sector académico. Esta aplicación pretende dar un paso más allá de la simple generación de tribunales a través del tratamiento de un documento con toda la información necesaria para la constitución de estos tribunales, por lo que se intentara comparar el modelo de negocio de la aplicación con otros que realicen las mismas tareas.

2.1 Aplicaciones similares

En este apartado nos centraremos en comparar, diversas aplicaciones y sistemas dentro del ámbito académico que constituyen tribunales de TFG y TFM. Al ser sistemas no comerciales y de uso académico interno de cada universidad y titulación, se ha hecho muy dificultoso encontrar sistemas parecidos al que se va a desarrollar en este TFG.

2.1.1 Escuela Técnica Superior de Ingeniería Informática

Actualmente en la Escuela Técnica Superior de Ingeniería Informática (ETSINF), el Jefe de Estudios es el encargado de llevar a cabo las tareas de constitución de tribunales. Estas tareas se están realizando a cabo a través de una aplicación de escritorio realizada en Java.

Este sistema tiene varias limitaciones de usabilidad, rendimiento, escalabilidad, entre otras. Para la resolución de las tareas de constitución de los tribunales, en esta aplicación en concreto solo hay que pasarle un fichero CSV con todos los profesores que cumplen los requisitos para la conformación de los tribunales.

Figura 1. Pantalla inicio aplicación GenerarTribuanles.jar

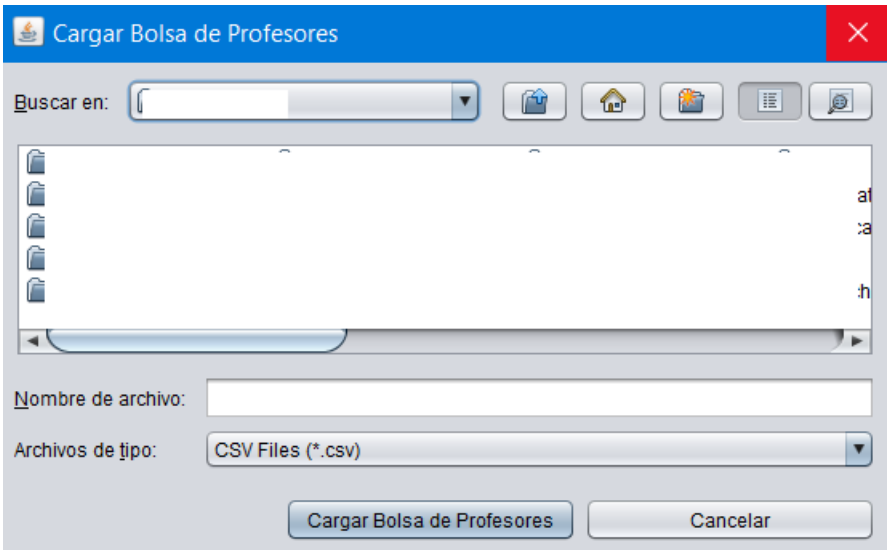


Figura 2. Vista Cargar Bolsa de profesores, aplicación GenerarTribunales.jar

A través de una interfaz de usuario simple, el usuario introduce la convocatoria y el número de tribunales que se tienen que conformar por rama de la titulación Ingeniería Informática. Una vez el sistema ha procesado la información en relación a unos requisitos, la devuelve en forma de fichero TXT.

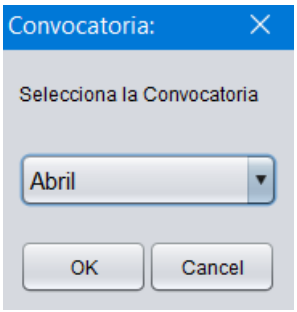


Figura 3. Vista convocatoria, aplicación GenerarTribunales.jar

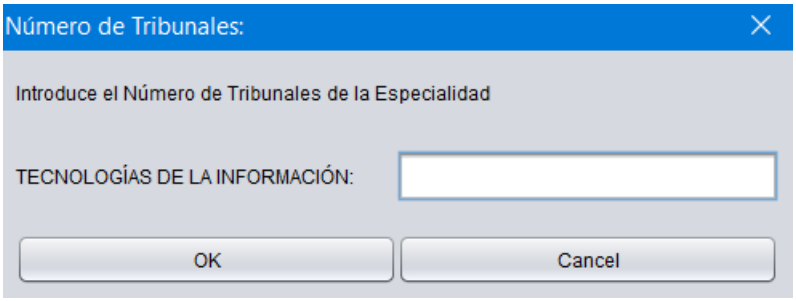


Figura 4. Vista número tribunales, aplicación GenerarTribunales.jar

2.1.2 Universidad de Murcia.

La Universidad de Murcia (UM) es una universidad española situada en la región de Murcia. Dentro de su plataforma educativa nos podemos encontrar integrada en el sistema la posibilidad a través de la secretaria del Decanato y/o el Jefe de la Secretaria la confección de los tribunales, su constitución y la asignación de los alumnos. En nuestra aplicación nuestra competencia solo llegaría hasta la constitución de esos tribunales.

Mediante la opción “Tribunales” -> “Constitución de tribunales” se elaborarán los tribunales pulsando el botón “Crear nuevo tribunal”. A partir de este punto se indicará el nombre del tribunal y la fecha de su constitución. Para facilitar su creación también se permitirá realizar diversos tipos de búsquedas, por titulación o por “DNI”, “Nombre y Apellidos” o “Correo electrónico”.

Una vez seleccionado el profesor se añadirá y podremos elegir el cargo del tribunal que ejercerá. Como particularidad la aplicación permite dar de alta a un tribunal a una persona ajena a la Universidad que previamente se ha debido proceder a su alta en el sistema [2].

2.2.3 Conclusiones

La tarea de buscar aplicaciones similares a la del presente proyecto ha sido ardua, puesto que los centros universitarios que utilizan estos tipos de sistemas lo hacen de forma interna, por lo que se dificulta en gran medida la obtención de información sobre estos. La mayoría de los centros universitarios cuentan con plataformas online en la que, en el mejor de los casos están integrados los sistemas para la conformación y confección de los tribunales como lo hace la Universidad de Murcia. En cambios otros como la ETSINF cuenta con una aplicación externa. En el peor de los casos la conformación de estos tribunales para la defensa de los trabajos se lleva de forma manual, con las numerosas desventajas que esto conlleva.

En el caso de que la Universidad conforme los tribunales de forma manual es poco práctico y seguro, puesto que se manejan datos sensibles relacionados con los profesores que imparten clase los centros. Se podría caer en el error de pensar que dichos datos se podrían tratar mejor en ficheros almacenados en dispositivos personales, esto a su vez no es seguro, puesto que nos podrían robar ese dispositivo o simplemente extraviarse la información. Sin embargo, si se cuenta con una aplicación robusta, eficiente y segura, los datos se almacenan de forma segura en una base de datos y nos despreocuparíamos de ese tipo de cuestiones.

Las aplicaciones estudiadas en el apartado, son muy limitadas. La que utiliza la ETSINF no permite crear nuevas titulaciones ni especialidades asociadas a estas y tampoco guarda los datos de forma persistente en una base de datos. La de la UM, por el contrario, sí que tiene cierta flexibilidad a la hora de crear tribunales, pero todos ellos se crean de uno en uno y con una persona que añada cada profesor manualmente asignándole el cargo que ejercerá.

A modo de resumen sobre el estudio realizado acerca de varias aplicaciones similares, se puede concluir que hacer una gestión manual de los datos es complicado y poco práctico. Automatizar el proceso de creación y gestión conllevaría muchos beneficios tanto de ahorro de tiempo, como de tratamiento de datos sensibles de personal docente.

3. Análisis del Problema

3.1 Solución Actual

Los principales inconvenientes de la forma de trabajar actual para crear y gestionar la conformación de los tribunales, son:

- La gestión e introducción de los profesores en el sistema. Actualmente el Jefe de Estudios trabaja con un fichero Excel donde introduce a mano los datos de cada profesor, para luego convertir ese fichero en un CSV y subirlo al sistema actual, pero el fichero debe actualizarse con nuevos datos constantemente, cada curso. Esto supone una serie de desventajas, como, por ejemplo, que el fichero se puede extraviar o corromper perdiendo así todos los datos de los profesores. Otra desventaja es el tiempo perdido cada vez que se tiene que subir el fichero CSV al sistema, que a su vez acarrea otro problema, no hay persistencia de ciertos datos de interés sobre los profesores que participan en los tribunales.
- Por otra parte, el sistema solo trabaja con los datos de la titulación Ingeniería Informática, por lo que actualmente los tribunales de las demás titulaciones y másteres de la ETSINF se están creando a manualmente. Esto supone un esfuerzo extra para el usuario, puesto que, si en un futuro se crean más titulaciones y/o másteres, o el número de estudiantes crece, el tiempo para crear todos los tribunales seria inasumible, por la cantidad de parámetros que se tienen que tener en cuenta para su conformación.

Con el sistema actual lo que se pretende es automatizar todo esto. Los datos de todos los profesores estarán subidos a una base de datos, por lo que la información de estos será persistente. Si el Jefe de Estudios quisiera dar de alta a un profesor nuevo en el sistema, tan solo tendría que rellenar los datos de un formulario y automáticamente el profesor nuevo pasara a formar parte del sistema. La persistencia de los datos de los profesores por otro lado permite que con esos datos de interés que se han comentado anteriormente, se puedan almacenar y trabajar con ellos para satisfacer ciertos requisitos que se citaran más adelante.

La mejora más sustancial del actual sistema es la opción de poder crear tribunales de todas las titulaciones de la ETSINF, siguiendo los criterios definidos por la comisión académica, dando soporte automatizado a la paridad, siempre que se pueda conseguir. Con esto se pretende suplir esa carencia, que por otro lado hacía que la conformación de los tribunales de las demás titulaciones se tuviera que realizar a mano. Esta mejora también permitirá que si en un futuro a corto, mediano o a largo plazo se crea una titulación o master nuevo en la ETSINF, tan solo habría que introducirlo en el sistema a través de un sencillo formulario.

3.2 Estudio de viabilidad

En este apartado evaluaremos si con los recursos actuales con los que se cuentan y el tiempo, será factible o no el desarrollo del proyecto desde los siguientes puntos de vista: Temporales, operativos y técnicos.

3.2.1 Viabilidad Temporal

En este apartado se hará un estudio sobre la viabilidad temporal del proyecto, empezando como se puede observar en la *Figura 5*, una estimación de horas de todas las fases que tendrá el desarrollo del sistema.

Fases del desarrollo	Duración (Dias)	Duración (Horas)
Analisis y especificación de requisitos	15	120
Diseño	10	80
Desarrollo	20	160
Pruebas	10	80
Validación	5	40
Total Horas		480

Figura 5. Estimación horas inicial

Como se puede apreciar en la tabla, la estimación de horas para el desarrollo del proyecto es de 480 horas-hombre calculadas para una dedicación de 8 horas al día. Suponiendo que al desarrollador no le surge ningún imprevisto, como, por ejemplo, ponerse enfermo en mitad de una fase del desarrollo y en este caso también se contemplar el periodo de formación que el desarrollador tendrá que hacer para ponerse al día de las tecnologías que se usaran en el proyecto, por lo que se ha decidido que en base a esas 480 horas-hombre se le sumen un 10% extra para cubrir los imprevistos que puedan afectar al coste temporal del proyecto.

En base a lo dicho en el párrafo de arriba en la *Figura 6*, se puede ver el resultado de la estimación del total de horas que va a suponer el desarrollo del software.

Estimación desarrollo del proyecto software	Imprevistos (10% sobre el total de horas)
480	48
Total (Horas) proyecto	528

Figura 6. Estimación final horas

Por motivos académicos y laborales solo se le dedicaran a la semana 25 horas, lo que suponen 22 semanas de desarrollo, lo que supondrán unos 5 meses y medio de trabajo.

El proceso de desarrollo software que se ha seguido para desarrollar el proyecto ha sido una un proceso de desarrollo iterativo o también conocido como incremental. Este proceso de desarrollo software a grandes rasgos consiste en repetir todas las fases de forma

iterativa hasta llegar al producto final. Cada cierto tiempo se llegaba a la fase de validación y se repetía todo el proceso en el caso de ser necesario. En puntos posteriores se hará un análisis más en profundidad de este proceso.

3.2.2 Viabilidad Técnica

En este apartado hablaremos sobre las herramientas y tecnologías necesarias para llevar a cabo el desarrollo del sistema.

El proyecto que se va a desarrollar en el presente TFG es una aplicación web basada en una API REST, y para su desarrollo se necesitaran las siguientes herramientas:

- Visual Studio Code: Visual Studio Code es uno de los editores de texto más populares en la actualidad, desarrollado por Microsotf para entornos Windows, Linux y Mac. Incluye soporte para la depuración, está integrado con Git, finalización inteligente de código, etc... También es destacable que es Open Source y gratuito.
- Git: Es un software de control de versiones de software libre creado por el creador de Linux, Linus Torvals. Su propósito es llevar el registro de los cambios en archivos y coordinar que varias personas estén trabajando de forma cooperativa.
- GitLab: Es un servicio web basado en Git para el control de versiones. Es también un gestor de repositorios, un sistema de seguimiento de errores y también ofrece alojamiento de wikis, todo esto bajo una licencia Open Source.

Como se ha podido observar todas las herramientas que se van a utilizar son de código abierto, por lo que no supondrán un coste monetario adicional al desarrollo del proyecto.

3.2.3 Viabilidad Operativa

El desarrollador disponía de algunos conocimientos sobre la tecnología que se iba a usar para desarrollar el sistema, por lo que se tenían que se tuvo que invertir un periodo de formación para aprender las tecnologías que se iban usar. Estas tecnologías por sus particularidades, tenían un pico de aprendizaje suave, junto con los conocimientos previos del desarrollador hizo que el tiempo invertido en esta formación fuera mínimo.

3.3 Conclusiones

Como se ha podido estudiar en este apartado, el problema que se quiere abordar en este proyecto necesita una solución específica y a medida puesto que la solución actual no cubre todas las necesidades que el usuario requiere para llevar a cabo las tareas de conformar los tribunales y las tareas asociadas a estas.

También se ha podido verificar que el proyecto es viable tanto en costo temporal y monetario, aunque el desarrollador tiene que pasar un tiempo formándose en parte de las tecnologías usadas para el desarrollo del sistema, aun con este coste temporal adicional de

formación el proyecto se podrá terminar si no surge ningún imprevisto que quede fuera del alcance del desarrollador en las horas planteadas inicialmente. Además, se disponen de las herramientas para el comiendo del proyecto.

4. Proceso de desarrollo Software

Este apartado se centrará en describir el proceso de desarrollo software que se ha decidido usar en el desarrollo de este proyecto software. La *Figura 7* [5] muestra de forma visual el concepto de este proceso que se va a explicar en los siguientes apartados, dando una visión general y profunda sobre las diferentes fases en las que se divide el desarrollo de la solución.

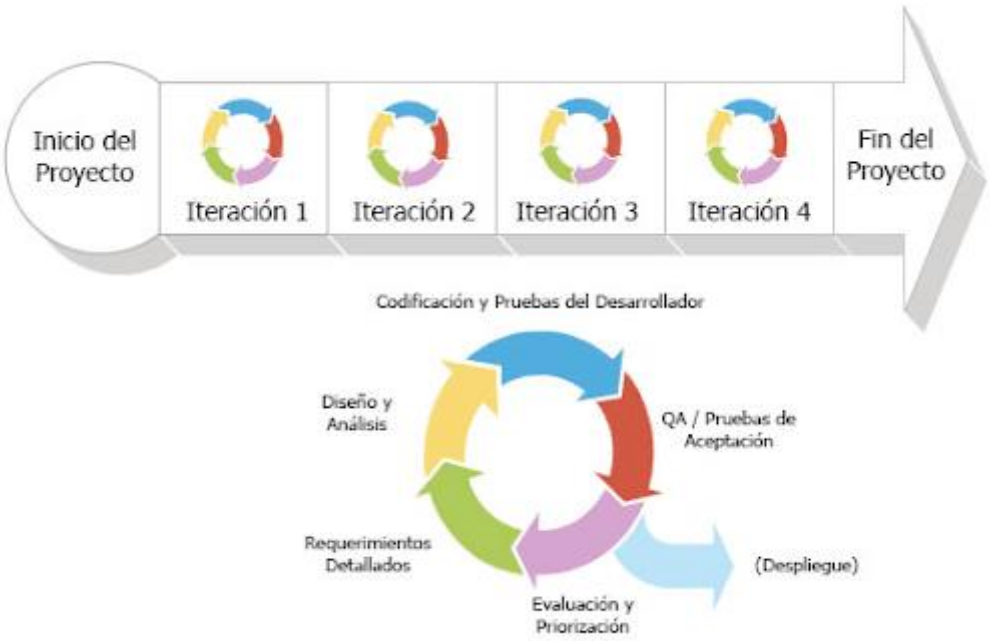


Figura 7. Metodología Iterativa o Incremental

4.1 Proceso software Iterativo o Incremental

Este proceso de desarrollo software [6] surge de la necesidad de poder suplir las debilidades del modelo tradicional en cascada. Este proceso consiste en iterar de forma progresiva las diferentes fases por la que pasa un desarrollo software. Cada iteración de estas fases da como resultado la entrega al usuario final o al cliente funcionalidad adicional, construyendo así un entorno adecuado para el cumplimiento de los requerimientos que en ocasiones pueden ir cambiando a lo largo del tiempo. El objetivo de estas iteraciones es un desarrollo ágil y rápido del producto software que se está construyendo.

4.2 Ventajas

A continuación, se enumerarán una serie de ventajas que tiene usar este proceso de desarrollo software frente a otros procesos:

- Se pueden gestionar las expectativas del cliente de manera regular, es decir, esto nos permite lidiar con clientes que no sabes exactamente qué es lo que necesitan. También puede permitir al cliente realizar cambio a corto plazo por cambios en el mercado o por cualquier cambio del entorno.
- El cliente o el usuario pueden obtener resultados usables e importantes a corto plazo.
- Se pueden gestionar de manera natural los cambios que van apareciendo durante el desarrollo del proyecto.
- Permite conocer el estado real del proyecto con precisión que a su vez no permite gestionar la complejidad del mismo.
- Se minimizan el número de errores que se pueden generar durante el desarrollo.
- El cliente en caso de no querer seguir con el proyecto, solo perdería los recursos de una o carias iteraciones, no del proyecto entero.

4.3 Desventajas

Como cualquier proceso de desarrollo no todo son cosas buenas y por ello, también enumeraremos los diferentes inconvenientes que presenta:

- La disponibilidad del cliente ha de ser alta, es decir, la comunicación con los desarrolladores ha de ser continua, puesto que la participación en el desarrollo se da de manera continuada.
- Al finalizar cada iteración del producto, este tiene que dar como resultado la finalización de los requisitos planteados al inicio de la iteración, puesto que debe de resultar un producto usable.
- La relación cliente/desarrollador se tiene que basar en una relación ganar/ganar.
- Se han de disponer de técnicas y herramientas que permitan realizar cambios fácilmente en el producto.

4.4 Fases del proceso de desarrollo iterativo y/o incremental

En este apartado hablaremos de las diferentes etapas y tareas que compondrán el proceso de desarrollo software iterativo en el ámbito de la ejecución de nuestro proyecto. Algunas de estas etapas y/o tareas son comunes a otros procesos.

4.4.1 Fase de preparación

Esta fase consiste en la aceptación del proyecto, la asignación de una fecha de inicio y de una fecha final. Se definirán los recursos destinados al proyecto. También se definirán los primeros requisitos del proyecto y estos serán los de primer nivel. En un desarrollo empresarial de alto nivel, esta fase sería mucho más compleja, pero para el proyecto que nos

atañe, con las premisas mencionadas anteriormente se tendrá suficiente para finalizar esta fase del proceso.

4.4.2 Fase de definición

En esta fase se definen las tareas orientadas a reconocer las funcionalidades necesarias en el nuevo sistema. Aquí profundizamos un poco más en los requerimientos de alto nivel que se definieron en la fase previa. También se validarán y se formalizaran los cambios se van a implementar.

4.4.3 Fase de prototipado iterativo

En esta fase se llevarán a cabo el diseño de los prototipos y/o versiones que se lo irán entregando al cliente en cada iteración de las fases.

En una primera instancia se instalarán y configurarán los entornos y herramientas para el desarrollo de los prototipos y/o versiones que se le irán entregando al cliente.

Se llevará a cabo la implementación del código que dará funcionalidad a los requisitos previamente definidos y se realizará el diseño del producto en nuestra una aplicación web. Se realizarán pruebas de para la verificación de los requisitos y se validarán con el cliente final y en el caso de no ser satisfactorias se volverán a repetir hasta su validación.

4.4.4 Fase Go live & Support

En esta última fase del desarrollo ya se ha llegado a la última iteración del producto por lo que solo faltaría desplegarlo, es decir, la entrada en producción del sistema. En esta fase también se detectarán las incidencias que no se detectaron en las fases previas y se llevara a cabo su corrección. También se empezará a llevar a cabo el mantenimiento del sistema.

4.5 Estructura del proyecto

Como se ha podido ver en los puntos previamente explicados, este proyecto se ira dividiendo en las consecutivas iteraciones que se vayan hablando con el Jefe de Estudios, el usuario final del sistema.

En la primera reunión con el Jefe de Estudios, a grandes rasgos se definieron la mayoría de los requisitos del sistema, pero se concretaron de manera más profunda y concisa los primeros requisitos que debería de tener el sistema, y se decidió que la primera funcionalidad que tendría que estar presente en la aplicación al terminar la primera iteración tendría que ser un sistema de autenticación y la vista de profesores. En esta vista se podría realizar acciones de crear, listar, modificar y borrar, de los profesores presentes en el sistema.

A partir de esta primera iteración iríamos dando forma al sistema y añadiendo la funcionalidad necesaria para llegar al producto final.

También a grandes rasgos se habló sobre las iteraciones posteriores y de la funcionalidad que deberían abarcar estas.

5. Especificación de requisitos

5.1 Introducción

Cuando se habla del análisis de requisitos de un producto software, podemos definirlo como la creación de un documento de especificación de requisitos que describa el sistema a hacer, pero no como hacerlo. Los requisitos son una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado. La definición de estos objetivos debe ser un trabajo conjunto entre los analistas software y el cliente.

La fase de requisitos según el estándar IEEE 1074 (IEEE, 1991) [7] se dividen en tres grandes fases:

- Definir los requisitos software.
- Definir los requisitos de las interfaces software con el resto del sistema.
- Integrar los requisitos en un documento de especificación.

En el proyecto que nos atañe solo nos centraremos en las dos primeras fases del análisis de requisitos definiendo los Requisitos Funcionales y los No Funcionales.

5.2 Requisitos Funcionales

En este punto se van a especificar los requerimientos del software. En este apartado se tiene como objetivo formalizar las funcionalidades de la aplicación y hacer un esquema inicial de su funcionamiento. A continuación, se van a detallar los **Requisitos Funcionales (RF)**:

- CRUD titulaciones: El administrador podrá crear, leer, actualizar y eliminar las titulaciones que se encuentren asociadas a la ETSINF.
- CRUD profesores: El administrador podrá crear, leer, actualizar y eliminar los profesores que impartan clase en las diferentes titulaciones.
- CRUD convocatorias: El administrador podrá crear, leer, actualizar y eliminar las convocatorias en las que se lleven a cabo las defensas de los TFGs/TFMs.
- CRUD especialidades: EL administrador podrá crear, leer, actualizar y eliminar las especialidades o ramas que están asociadas a las titulaciones.
- CRUD tribunales: El administrador podrá crear, leer, actualizar y eliminar los tribunales de una titulación designados para una o varias convocatorias.
- Generar tribunales: El administrador podrá generar varios tribunales a la vez de una titulación en su correspondiente convocatoria de forma automática.
- CRUD de usuario: El administrador podrá dar de alta a nuevos usuarios, visualizarlos, modificarlos y darlos de baja, en cualquier momento.
- Login: El administrador tendrá que acceder al sistema usando un usuario (correo) y una contraseña.

- Logout: El administrador podrá desconectarse de la sesión y salir del sistema.
- Generar informes: El administrador podrá visualizar y descargarse un informe en PDF y .t de los tribunales generados.

5.3 Requisitos No Funcionales

En este apartado se van a especificar las propiedades que el sistema debe tener. El objetivo es saber que requisitos específicos va a poder cumplir el software. Para esto se detallarán los **Requisitos No Funcionales (RNF):**

- Usable: La aplicación debe ser fácil de usar e intuitiva. El usuario debe de saber utilizar el sistema solo con la interfaz de usuario.
- Mensajes de información orientativos: El sistema cada vez que el usuario realice una tarea deberá informar con un mensaje de éxito o error. También dichos mensajes deberán avisar al usuario que se está realizando una tarea critica (Eliminar datos).
- Disponibilidad: El sistema debe de estar disponible los 365 días durante las 24 horas.
- Compatibilidad de navegadores: La aplicación deberá ser compatible con cualquier navegador que utilice el usuario.
- Seguridad de peticiones HTTP: Todas las comunicaciones que se realicen a la API REST dentro de la aplicación deberán de validarse mediante un token asociado a la sesión de usuario.
- Tiempo de peticiones HTTP: Todas las peticiones que se realicen a la API REST no deberán superar 1 segundo de tiempo.

5.4 Diagrama de casos de uso

Dentro del **Lenguaje de Modelado Unificado (UML)** se define una notación grafica para representar casos de uso. Esta notación grafica pretende dar una vista general y simple del comportamiento del sistema con los diferentes actores que interactuaran con él.

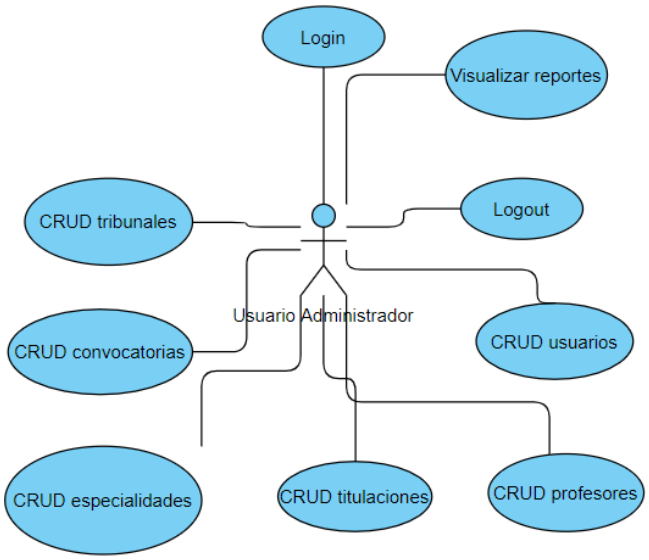


Figura 8. Diagrama de caso de uso.

5.5 Descripción de casos de uso

A continuación, se van a describir con más detalle los casos de uso vistos en la *Figura 2*. También se mostrarán algunos diagramas de flujo de la aplicación.

Caso de uso	Login.
Actores	Administrador.
Propósito	Autenticarse para poder acceder a la aplicación.
Resumen	El administrador con su email y contraseña, puede identificarse para acceder al sistema.
Precondiciones	-
Postcondiciones	EL usuario queda autenticado.

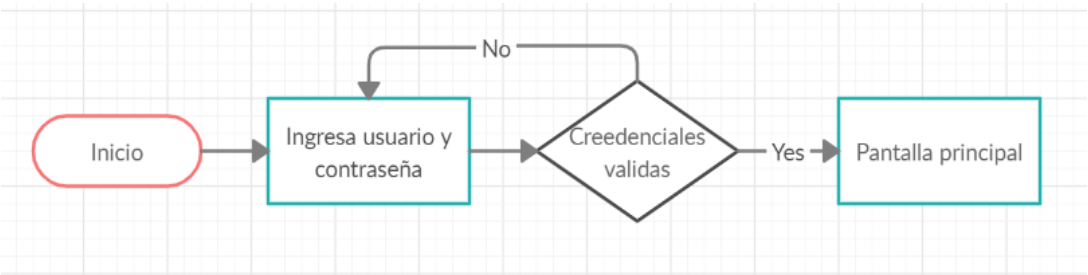


Figura 9. Diagrama de flujo Login.

En las siguientes 4 tablas se explicarán los casos de uso asociados al **CRUD (Create, Read, Update and Delete)** y se mostrara el diagrama de flujo de **"CRUD Usuarios".**

Caso de uso	Crear usuario.
Actores	Administrador.
Propósito	Registrar usuarios administradores.
Resumen	EL administrador del sistema puede dar de alta nuevos usuarios.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Usuario creado.

Caso de uso	Listar usuarios.
Actores	Administrador.
Propósito	Ver el listado de usuarios.
Resumen	El administrador podrá visualizar una lista con todos los usuarios que existen en el sistema.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Usuarios listados.

Caso de uso	Modificar usuario.
Actores	Administrador.
Propósito	Modificar un profesor.
Resumen	El administrador podrá modificar los datos de un usuario.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Datos del usuario actualizados.

Caso de uso	Borrar usuario.
Actores	Administrador.
Propósito	Borrar un profesor.
Resumen	El administrador podrá borrar del sistema a un usuario.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Usuario borrado.

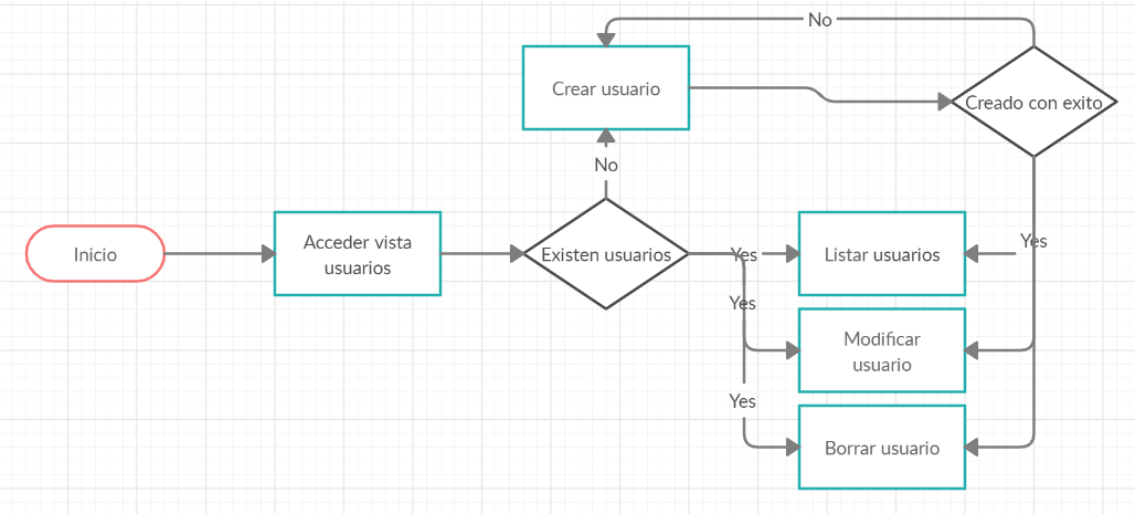


Figura 10. Diagrama de flujo CRUD Usuarios.

El siguiente caso de uso que se va a explicar es **"Visualizar informes"**, así como su diagrama de flujo.

Caso de uso	Visualizar informes.
Actores	Administrador.
Propósito	Generar documentos pdf y txt.
Resumen	El administrador del sistema podrá generar informes de los tribunales creados.
Precondiciones	El administrador ha iniciado sesión. El tribunal o tribunales existen.
Postcondiciones	Informe descargado.

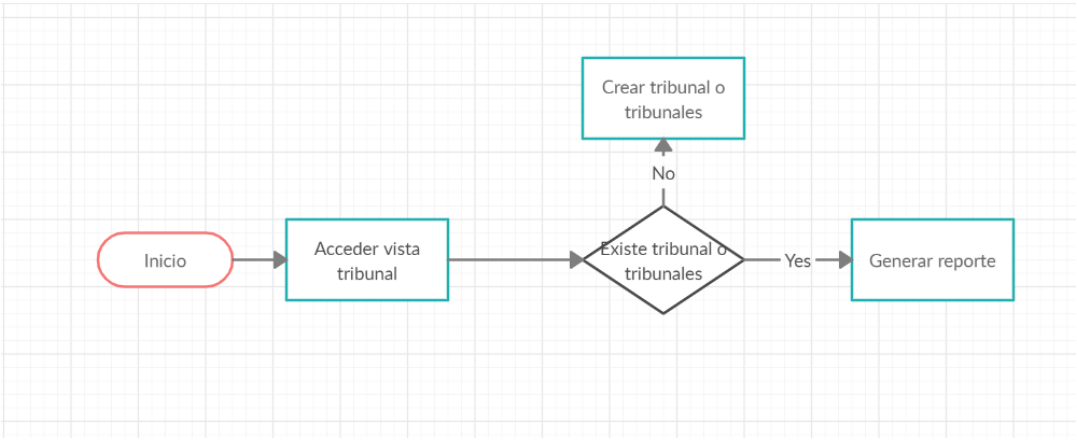


Figura 11. Diagrama de flujo Generar Reporte.

En las siguientes 4 tablas se explicarán los casos de uso asociados al **CRUD (Create, Read, Update and Delete)** y se mostrara el diagrama de flujo de **"CRUD Profesores".**

Caso de uso	Crear profesor.
Actores	Administrador.
Propósito	Crear un profesor.
Resumen	El administrador podrá crear mediante un formulario.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Profesor creado.

Caso de uso	Listar profesores.
Actores	Administrador.
Propósito	Ver el listado de profesores.
Resumen	El administrador podrá visualizar una lista con todos los profesores que existen en el sistema.
Precondiciones	El administrador ha iniciado sesión. Los profesores deben de existir.
Postcondiciones	Profesores listados.

Caso de uso	Actualizar profesor.
Actores	Administrador.
Propósito	Modificar un profesor.
Resumen	El administrador podrá modificar los datos de un profesor.
Precondiciones	El administrador ha iniciado sesión. Los profesores deben de existir.
Postcondiciones	Datos del profesor actualizados.

Caso de uso	Borrar profesor.
Actores	Administrador.
Propósito	Borrar un profesor.
Resumen	El administrador podrá borrar del sistema a un profesor.
Precondiciones	El administrador ha iniciado sesión. Los profesores deben de existir
Postcondiciones	Usuario borrado.

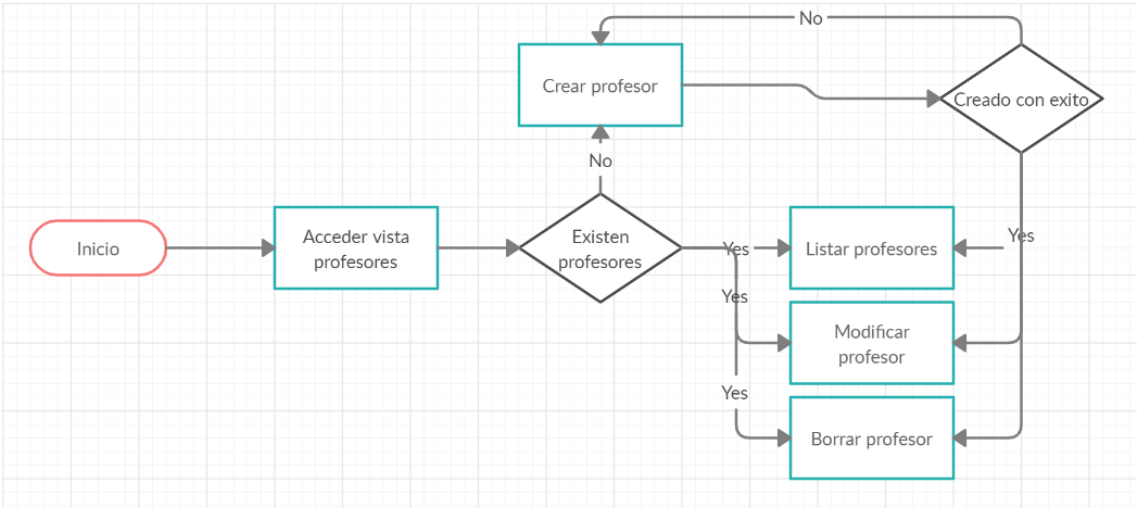


Figura 12. Diagrama de flujo CRUD profesores.

En las siguientes 4 tablas se explicarán los casos de uso asociados al **CRUD (Create, Read, Update and Delete)** y se mostrara el diagrama de flujo de **"CRUD titulaciones".**

Caso de uso	Crear titulación.
Actores	Administrador.
Propósito	Crear una titulación.
Resumen	El administrador podrá crear mediante un formulario.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Titulación borrada.

Caso de uso	Listar titulaciones.
Actores	Administrador.
Propósito	Ver el listado de titulaciones.
Resumen	El administrador podrá visualizar una lista con todas las titulaciones que existen en el sistema y sus correspondientes especialidades.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	Titulaciones listadas.

Caso de uso	Actualizar titulación.
Actores	Administrador.
Propósito	Modificar una titulación.
Resumen	El administrador podrá modificar los datos de una titulación.
Precondiciones	El administrador ha iniciado sesión. Las titulaciones deben de existir.
Postcondiciones	Datos de la titulación actualizados

Caso de uso	Borrar titulación.
Actores	Administrador.
Propósito	Borrar una titulación.
Resumen	El administrador podrá borrar del sistema a una titulación.
Precondiciones	El administrador ha iniciado sesión. Las titulaciones deben de existir.
Postcondiciones	Titulación borrada junto con todas las especialidades asociadas.

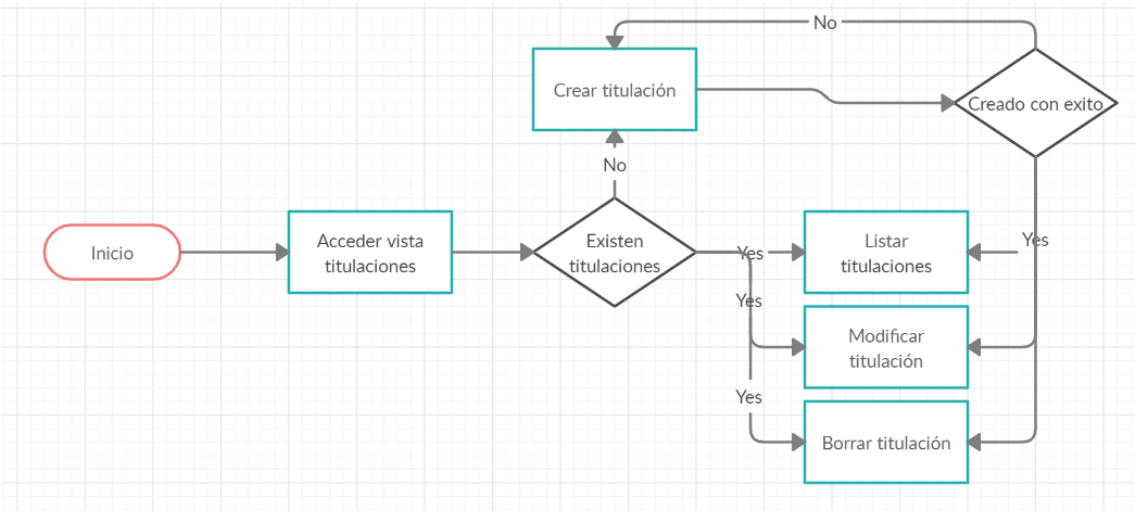


Figura 13. Diagrama de flujo CRUD titulaciones.

En las siguientes 4 tablas se explicarán los casos de uso asociados al **CRUD (Create, Read, Update and Delete)** y se mostrara el diagrama de flujo de **"CRUD especialidades".** Para el caso de uso "CRUD convocatorias" es el exactamente igual, por lo que no hará falta incluirlo.

Caso de uso	Crear especialidad.
Actores	Administrador.
Propósito	Crear una especialidad.
Resumen	El administrador podrá crear mediante un formulario especialidades.
Precondiciones	El administrador ha iniciado sesión. Hay una titulación, con la que asociar la especialidad.
Postcondiciones	Especialidad creada y asociada a una titulación.

Caso de uso	Listar especialidades.
Actores	Administrador.
Propósito	Ver el listado de especialidades.
Resumen	El administrador podrá visualizar una lista con todas las especialidades que existen en el sistema.
Precondiciones	El administrador ha iniciado sesión. Las titulaciones deben de existir.
Postcondiciones	Especialidades listadas.

Caso de uso	Actualizar especialidades.
Actores	Administrador.
Propósito	Modificar una especialidad.
Resumen	El administrador podrá modificar los datos de una especialidad.
Precondiciones	El administrador ha iniciado sesión. Las titulaciones deben de existir.
Postcondiciones	Datos de la especialidad actualizados.

Caso de uso	Borrar especialidades.
Actores	Administrador.
Propósito	Borrar una especialidad.
Resumen	El administrador podrá borrar del sistema a una especialidad.
Precondiciones	El administrador ha iniciado sesión. Las titulaciones deben de existir.
Postcondiciones	Especialidades borradas.

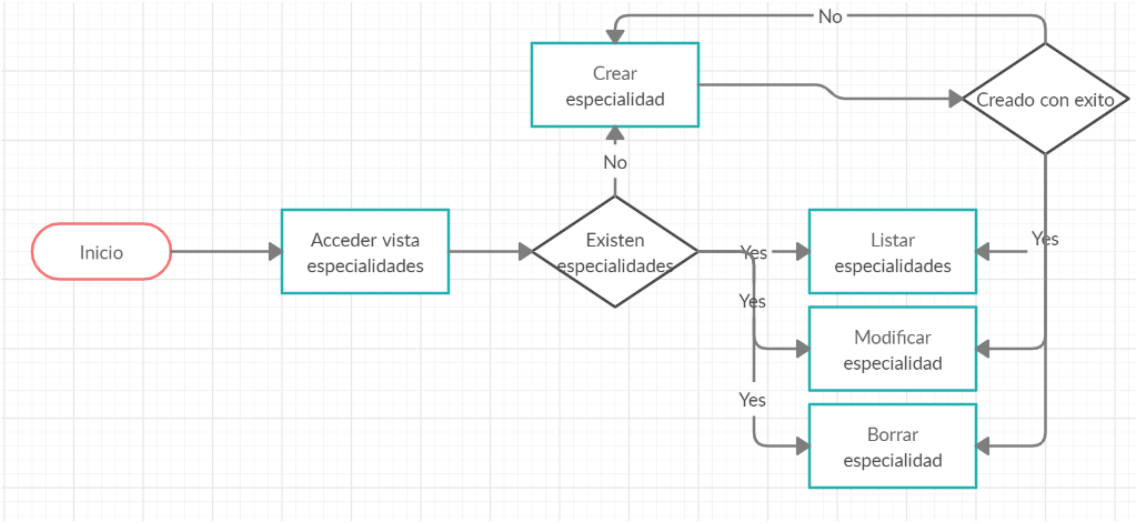


Figura 14. Diagrama de flujo CRUD especialidades.

En las siguientes 4 tablas se explicarán los casos de uso asociados al **CRUD (Create, Read, Update and Delete)** y se mostrara el diagrama de flujo de **"CRUD tribunales".**

Caso de uso	Crear tribunal.
Actores	Administrador.
Propósito	Generar tribunales.
Resumen	El administrador podrá generar un tribunal o varios de una titulación y especialidad de una vez.
Precondiciones	El administrador ha iniciado sesión. La titulación o especialidad deben existir. La convocatoria debe existir.
Postcondiciones	Tribunal creado.

Caso de uso	Listar tribunales.
Actores	Administrador.
Propósito	Ver un listado de los tribunales.
Resumen	El administrador podrá visualizar una lista de todos los tribunales y los correspondientes profesores asociados al mismo.
Precondiciones	El administrador ha iniciado sesión. Los tribunales deben de existir.
Postcondiciones	Tribunales listados.

Caso de uso	Actualizar tribunales.
Actores	Administrador.
Propósito	Modificar un tribunal.
Resumen	El administrador podrá modificar los datos de un tribunal.
Precondiciones	El administrador ha iniciado sesión. Los tribunales deben de existir.
Postcondiciones	Datos de los tribunales actualizados

Caso de uso	Actualizar tribunales.
Actores	Administrador.
Propósito	Modificar un tribunal.
Resumen	El administrador podrá modificar los datos de un tribunal.
Precondiciones	El administrador ha iniciado sesión. Los tribunales deben de existir.
Postcondiciones	Tribunal borrado.

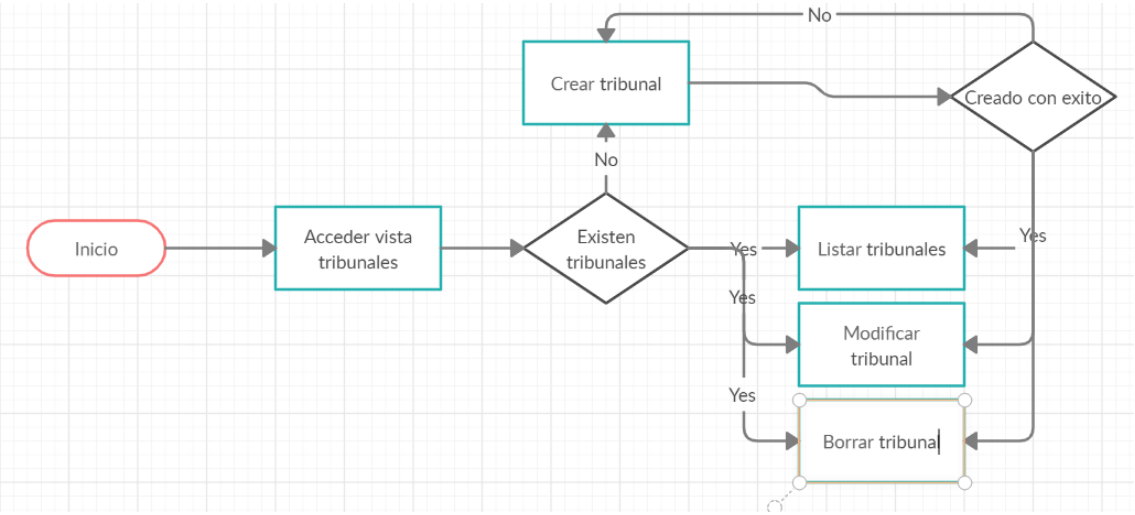


Figura 15. Diagrama de flujo CRUD tribunales.

6. Diseño

6.1 Introducción

En el desarrollo de software una de las fases por la que tiene que pasar el desarrollo es por el diseño. Esta etapa es fundamental y depende del software podríamos decir que es fundamental. En esta etapa se tiene que llegar a una solución que cumpla con los requerimientos funcionales y no funcionales que se han establecido en la fase de análisis de requisitos.

El diseño del software incide directamente sobre la capacidad del sistema para el total cumplimiento de los requerimientos establecidos. Un error en esta fase puede acarrear problemas en todo el proyecto y provocar que se caiga en una vorágine de cambios constantes y por ende tener que rehacer constantemente el trabajo. Por eso esta fase es de vital importante porque puede decantar un software al éxito o al fracaso sin olvidarnos que un diseño bien realizado es sinónimo de software de calidad.

Normalmente esta fase suele constar de varias subetapas:

- Diseño de los datos. Se define y modela la base de datos y la relación entre las diferentes tablas que la conforman. En nuestro proyecto la base de datos se ha modelado con la herramienta MySQL Workbench.
- Diseño Arquitectónico. En esta etapa se describe como el software se tiene que comunicar con los diferentes componentes que lo componen, con los sistemas que operan junto a él y con los usuarios que lo emplean. En el proyecto se usará la arquitectura Modelo Vista controlador (MVC) que viene integrada dentro del Framework que se ha usado para el desarrollo de la aplicación.
- Diseño de la interfaz. En esta etapa se suelen diseñar los prototipos que darán como resultado la interfaz visual de la aplicación, con esto lo que logra es dar un pequeño vistazo a lo que más adelante será la aplicación final. En nuestro proyecto la herramienta para definir los prototipos de las vistas que tendrá nuestra aplicación se ha creado que la herramienta Pencil.

6.2 Diseño y modelado de la base de datos

Nuestro sistema para poder persistir los datos, dispondrá de una base de datos SQL que está formada por las diferentes tablas que se muestran en la *Figura 16.*

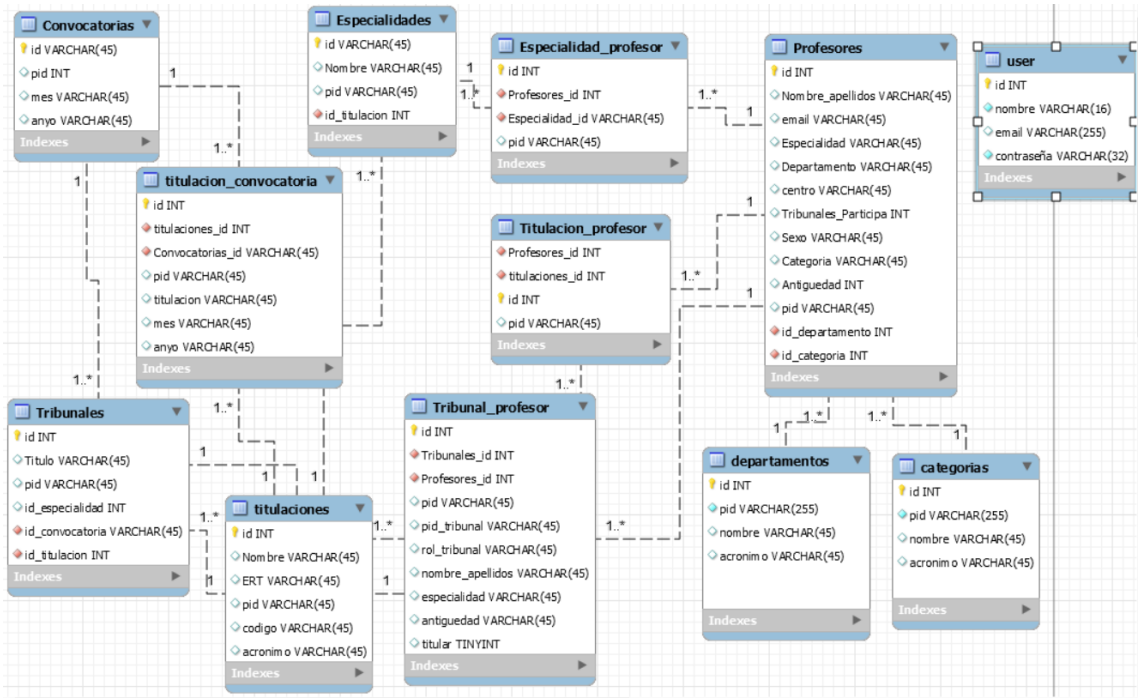


Figura 16. Modelo de base de datos.

Todas las tablas, para dotar al sistema de seguridad adicional y asegurarnos que ninguna entidad, ni persona externa pueda acceder a los datos de manera fácil, se ha añadido un campo Pid, este campo es el identificador público, que servirá para realizar acciones sobre los datos que típicamente necesitaran de un identificador.

- **Profesores**: En esta tabla se almacenan todos los profesores que pueden pasar a formar parte de un tribunal. Está compuesta por los siguientes campos:
  - Id: Identificador del profesor.
  - Pid: Identificador público.
  - Nombre_apellidos: Nombre y apellidos del profesor.
  - Email: Correo electrónico del profesor.
  - Centro: Centro del profesor.
  - Especialidad: Especialidad o especialidades del profesor.
  - Departamento: Departamento del profeso.
  - Tribunal_Participa: Numero de tribunales en los que el profesor participa.
  - Sexo: sexo del profesor.
  - Categoría: Categoría del profesor.
  - Antigüedad: Número de años que el profesor lleva impartiendo clase.
  - Categorias_id: Clave foránea de categorías.
  - Departamentos_id: Clave foránea de departamentos.

- **Titulaciones**: Esta tabla alberga las titulaciones de las ETSINF. Está compuesta por los siguientes campos:
  - Id: Identificador de la titulación.
  - Pid: identificador público.

  - Nombre: Nombre de la titulación.
  - ERT: Estructura responsable del título.
  - Código: Código de la titulación.
  - Acrónimo: Siglas de la titulación.

- **Especialidades**: Esta tabla contiene las especialidades de cada título. Está formada por los siguientes campos:
  - Id: Identificador de la titulación.
  - Pid: identificador público.
  - Nombre: Nombre de la especialidad.
  - Id_titulación: clave foránea de la titulación.

- **Convocatorias**: Tabla donde de almacenan las convocatorias de los tribunales. Contiene los siguientes campos:
  - Id: Identificador de la convocatoria.
  - Pid: Identificador público.
  - Mes: Mes de la convocatoria.
  - Anyo: Año de la convocatoria.

- **Tribunales**: Tabla donde se almacenan los tribunales. Está formada por los siguientes campos:
  - Id: Identificador del tribunal.
  - Pid: Identificador público.
  - Id_especialidad: Clave foránea de la especialidad.
  - Id_convocatoria: Clave foránea de la convocatoria.
  - Id_titulacion: Clave foránea de la titulación.

- **Departamentos**: Tabla donde se guardan los departamentos donde pertenecen los profesores. Está compuesta por los siguientes campos:
  - Id: Identificador del departamento.
  - Pid: Identificador público.
  - Nombre: Nombre del departamento.
  - Acrónimo: Siglas del departamento.

- **Categorías**: Tabla donde se almacenan las categorías que tienen los profesores. Está compuesta por los siguientes campos:
  - Id: Identificador de la categoría.
  - Pid: Identificador público.
  - Nombre: Nombre de la categoría.
  - Acrónimo: Siglas de la categoría.

- **Users**: Tabla donde se guardan los usuarios que pueden acceder al sistema para su gestión. Está compuesta por los siguientes campos:
  - Id: Identificador del usuario.
  - Nombre: Nombre del usuario.
  - Email: Correo electrónico del usuario.

    - Contraseña: Contraseña del usuario.

- **Titulacion_profesor:** Tabla intermedia que relaciona a profesores y a las titulaciones a los que pertenecen. Sus campos son:
    - Id: Identificador de Titulacion_profesor.
    - Pid: Identificador público.
    - Id_profesor: Clave foránea de profesor.
    - Id_titulacion: Clave foránea de titulación.

- **Tribunal_profesor:** Tabla intermedia que relaciona a los profesores que forman un tribunal. Está formada por:
    - Id: Identificador de Tribunal_profesor.
    - Pid: Identificador público.
    - Pid_tribunal: Identificador público de tribunal.
    - Rol_tribunal: Rol que asumirá el profesor en el tribunal.
    - Nombre_apellidos: Nombre y apellidos del profesor.
    - Especialidad: Especialidad del profesor.
    - Antigüedad: Antigüedad del profesor en la ETSINF.
    - Titular: Valor booleano para saber si un profesor es titular o no.
    - Id_tribunal: Clave foránea tribunal.
    - Id_profesor: Clave foránea profesor.

- **Titulacion_convocatoria:** Tabla intermedia que relaciona a una titulación con una convocatoria. Sus campos son:
    - Id: Identificador de Titulacion_convocatoria.
    - Pid: Identificador público.
    - Titulación: Nombre de la titulación.
    - Mes: Mes de la convocatoria.
    - Anyo: Año de la convocatoria.
    - Id_titulación: Clave foránea de titulación.
    - Id_convocatoria: Clave foránea de convocatoria.

- **Especialidad_profesor:** Tabla intermedia que relaciona que especialidades tienen los profesores. Sus campos son:
    - Id: Identificador de Especialidad_profesor.
    - Pid: Identificador público.
    - Id_especialidad: Clave foránea especialidad.
    - Id_profesor: Clave foránea profesor.

Una vez se han explicado las tablas que conforman la base de datos y los campos que tienen, vamos a dar una breve explicación de cómo están relacionadas las tablas y de la cardinalidad de cada una de estas relaciones.

- La tabla profesores esta relaciona con la tabla departamentos y categorías con una cardinalidad de 1 a M. Esto es que un profesor solo puede pertenecer a un departamento y a una categoría. También está relacionada con las tablas

especialidades, tribunales y titulaciones con una cardinalidad de N a M, que esto da como lugar a tres tablas intermedias que son la de Especialidad_profesor, Tribunal_profesor y Titulacion_profesor.

- La tabla tribunales está relacionada con las tablas convocatorias y titulaciones con una cardinalidad de 1 a M. Por lo que una titulación solo puede pertenecer a una convocatoria y solo puede tener una titulación.
- La tabla especialidades está relacionada con la tabla titulaciones con una cardinalidad de 1 a M. Por lo que una especialidad solo puede pertenecer a una titulación.
- La tabla convocatoria está relacionada con la tabla de titulaciones con una cardinalidad de N a M. Por lo que da lugar a una tabla intermedia que es la de Titulacion_convocatoria.

6.3 Arquitectura Software

La arquitectura usada en el sistema que estamos desarrollando es la del Modelo Vista Controlador (MVC) [8]. El MVC es un patrón d arquitectura, que separa la capa de persistencia o de datos y lo que es la capa de lógica o de lógica de negocio de una aplicación de su capa de presentación.

Para ello esta arquitectura propone la construcción de tres componentes básicos que son el modelo, la vista y el controlador. Este patrón arquitectónico se fundamenta en las ideas de utilización de código y la separación de conceptos, cosas que buscan para que los desarrolladores puedan crear software y posteriormente el mantenimiento de ese software sea muy sencillo.

6.3.1 Descripción del patrón Arquitectónico

Como se ha comentado, ese patrón se compone de tres componentes fundamentales que son:

- **Modelo**: Es la representación de la información, por lo que este componente gestiona todos los accesos a dicha información. Envía a la vista aquellos datos que el usuario u otra aplicación le solicita en cada momento.
- **Controlador**: Responde a eventos, usualmente acciones del usuario e invoca las peticiones al modelo cuando se hace alguna solicitud sobre la información. Por lo que se podría decir que el controlador hace de intermediario entre la vista y el modelo.
- **Vista**: Presenta el modelo en un formato adecuado para que el usuario pueda interactuar con los datos a través de acciones.

En la *Figura* 17, se puede apreciar de forma gráfica como interactúan el modelo, la vista y el controlador.

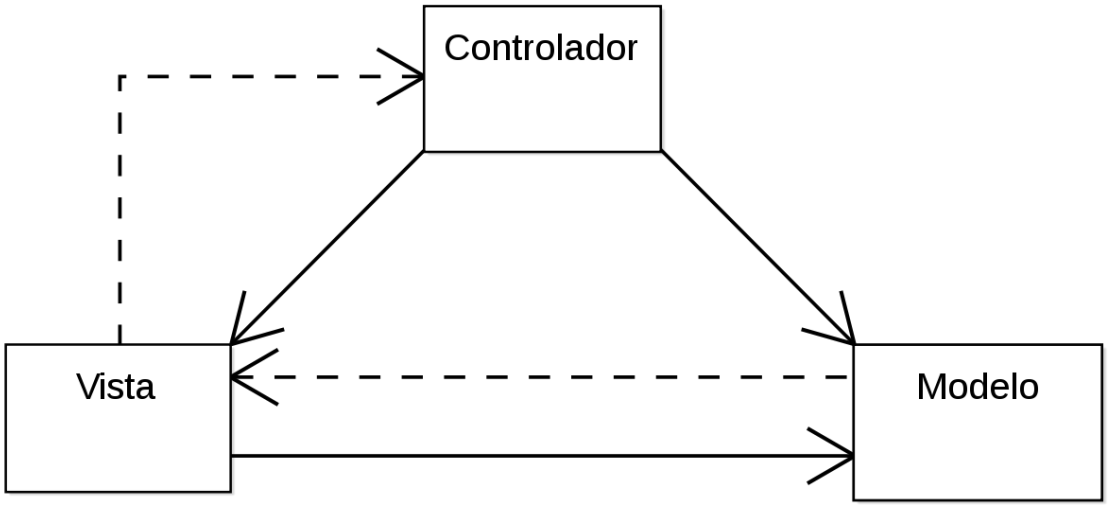


Figura 17. Modelo Vista Controlador

En nuestro sistema la arquitectura viene definida por el Framework de PHP que se ha usado para el desarrollo de la aplicación. En el punto 7.3 se explicará con detalle como Laravel implementa este patrón arquitectónico y como se estructuran los diferentes componentes, anteriormente mencionados.

6.4 Diseño de la interfaz. Prototipos

En este apartado presentaremos diferentes prototipos realizados con la herramienta Pencil [17]. Por simplicidad no mostraremos todos los prototipos realizados porque muchos de ellos son redundantes, por lo que solo se mostraran y explicaran los más relevantes para la aplicación, así como algunos de los formularios modales.

6.4.1 Login

El login a la aplicación es fundamental para que el usuario se pueda conectar al sistema y poder hacer uso de él. En la *Figura 18*, se puede apreciar, que el formulario de logueo del usuario es intuitivo y fácil de usar, tiene dos Input de texto y dos iconos situados a la izquierda. También se tiene un Check para que el usuario si así lo desea pueda almacenar los datos y no tener que poner sus credenciales cada vez que tenga que entrar al sistema. Y por último un botón para poder acceder al sistema si todo está correcto. En el punto 9 se mostrará la misma vista, pero ya en un entorno real.

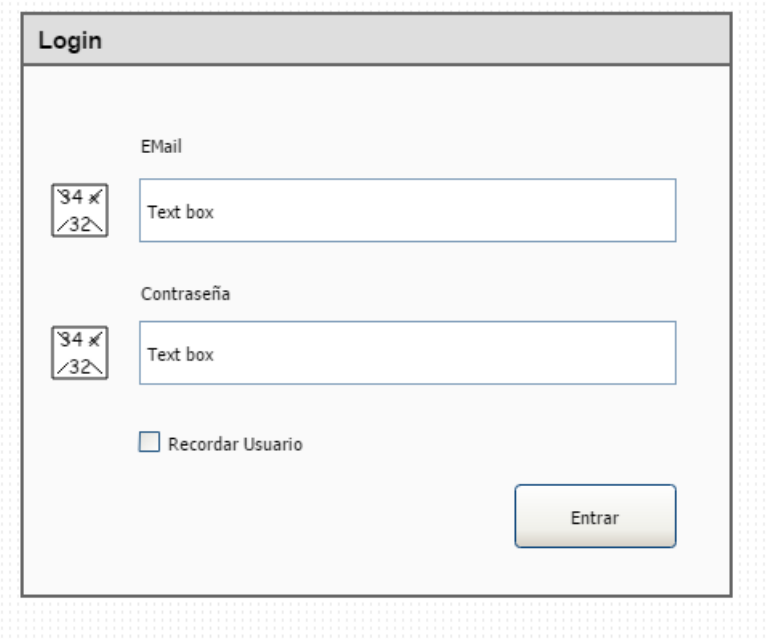


Figura 18. Vista Login

6.4.2 Vistas

Las diferentes vistas de nuestro sistema están divididas en 6 pantallas, pero de ellas 6, cinco de ellas son idénticas lo único que cambia es la funcionalidad que ofrecen, por lo que aquí solo se mostraran tres de esas vistas y se explicara su diseño.

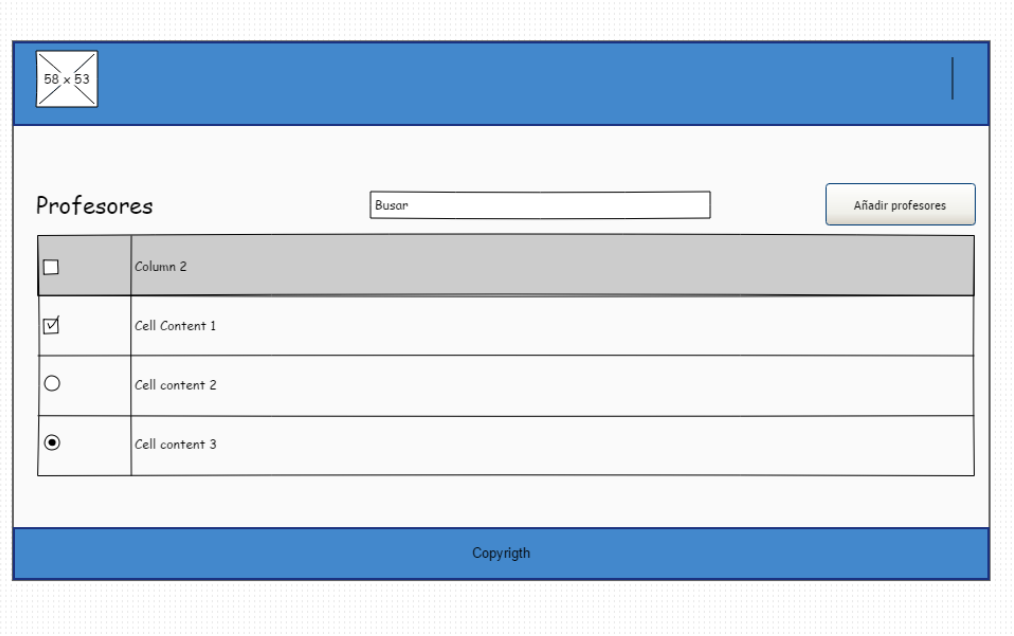


Figura 19. Vista Profesores

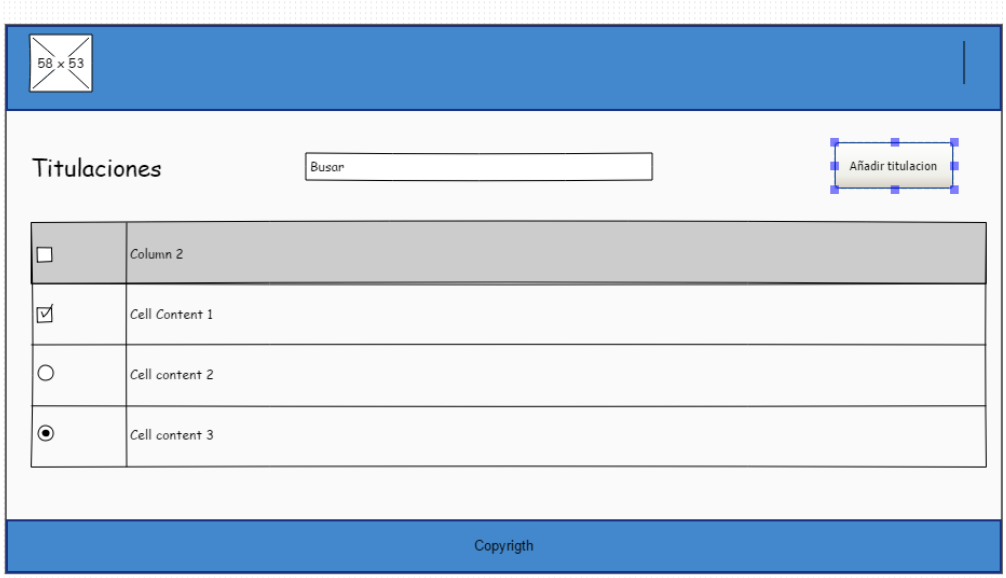


Figura 20. Vista Titulaciones

En la *Figura 19* y *20*, se pueden observar la vista de profesores y titulaciones, estas vistas están constituidas por el nombre de la vista, un Input de búsqueda, con el que podremos buscar en la tabla adyacente y un botón de Añadir, arriba a la derecha. También cuenta con una tabla en la que se mostraran los datos, en estos dos casos de los profesores y las titulaciones. Contaran también con un Header y un Footer. En el Header habrá una barra de menú arria a la izquierda y un icono a la derecha para poder hacer Logout. En el Footer solo habrá información del CopyRight.

La *Figura 21*, muestra la vista tribunales que en esencia es idéntica a las otras dos vistas anteriormente, pero tiene algunas particularidades propias

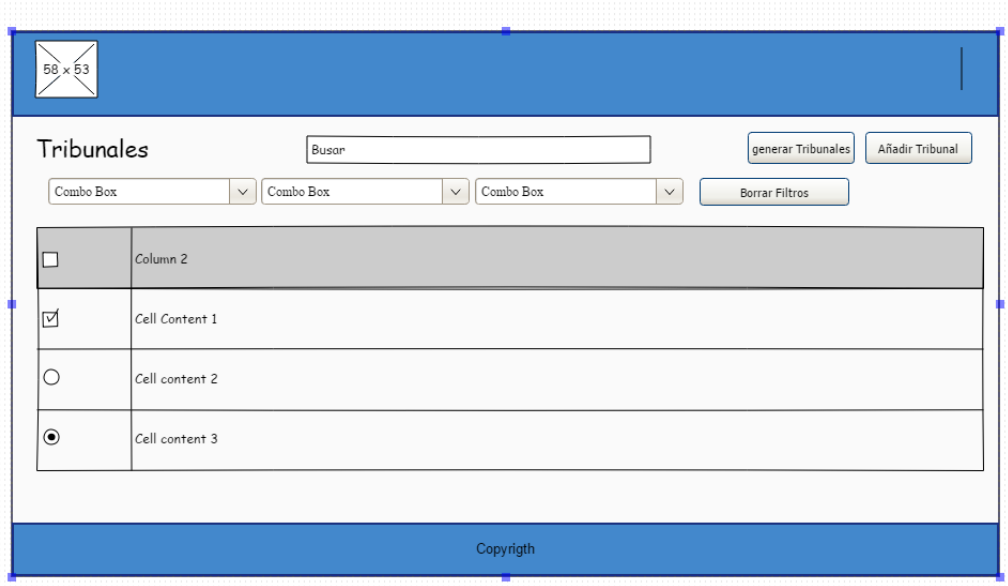


Figura 21. Vista tribunales

Las diferencias con las demás vistas radican en que, esta vista cuenta con dos botones que son Añadir tribunal y Generar tribunal, y unos filtros para buscar de forma personalidad, puesto que los datos de esta vista se irán actualizando conforme el Jefe de Estudios vaya creando los tribunales en las diferentes convocatorias.

6.4.3 Menú

En la *Figura 22*, está el prototipo del menú. Este menú cuenta con los enlaces a las diferentes vistas y unos iconos representativos para cada opción.

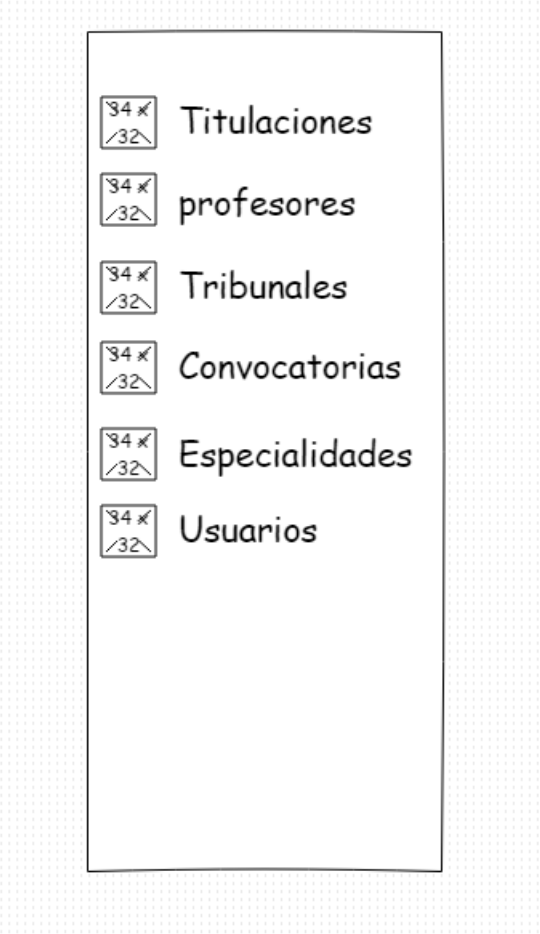


Figura 22. Menú

6.4.4 Modales

Los Modales con los que cuenta la aplicación son abundantes, por cada botón de acción hay uno diferente. Por lo que hemos comentado al principio del apartado solo se mostraran varios de ellos para simplificar. Cuando se hace click a un botón dentro del sistema

se desplegará un formulario en los que realizar las diferentes acciones que nos ofrece la funcionalidad del sistema.

En la *Figura 23 y 24*, podemos ver que los dos modales se conforman de Inputs para insertar texto o de ComboBox para elegir una o varias opciones de datos. También se tienen dos botones para cancelar o aceptar, que sirven para confirmar los datos del formulario o cancelarlos.

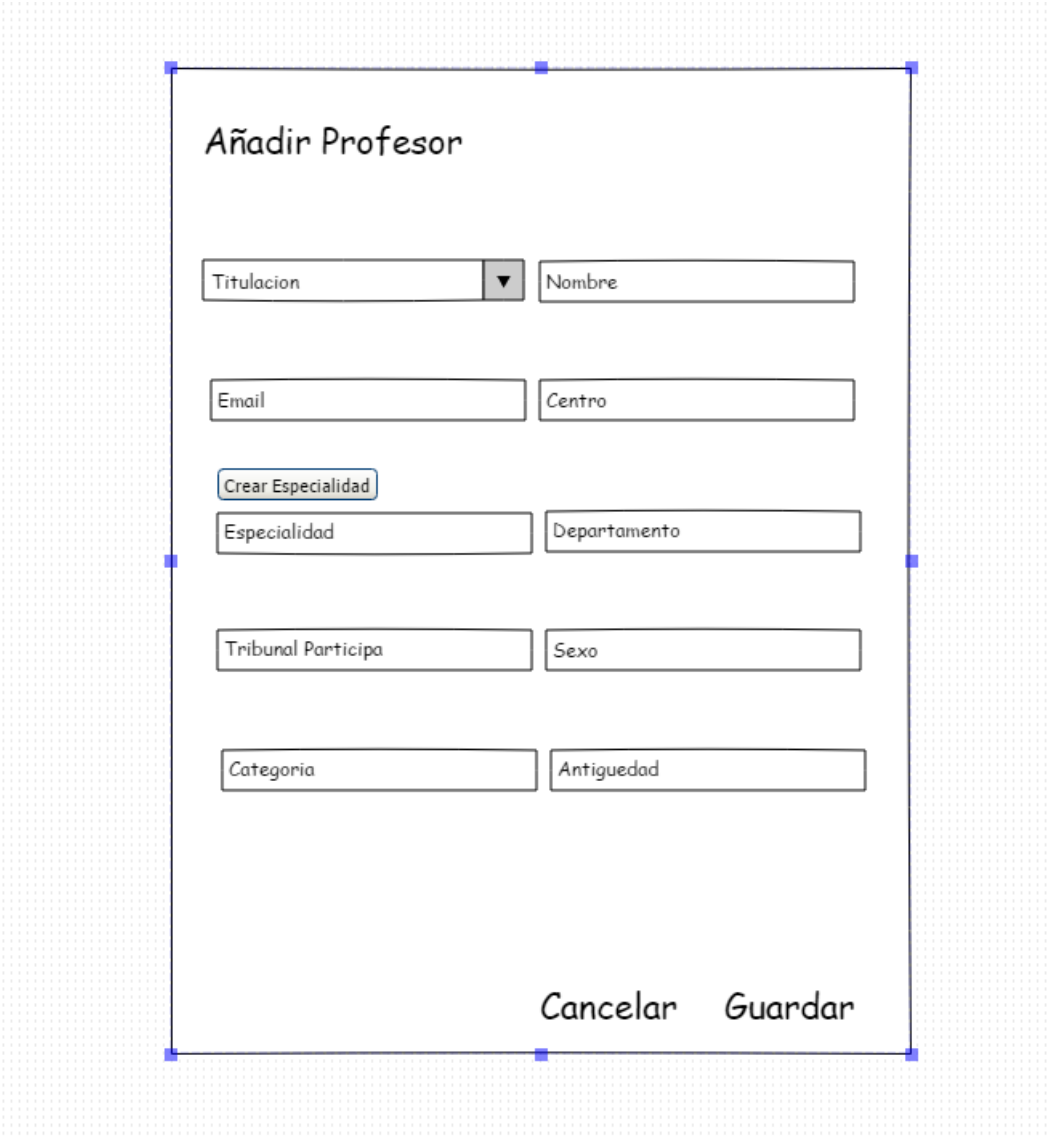


Figura 23. Modal Añadir profesor

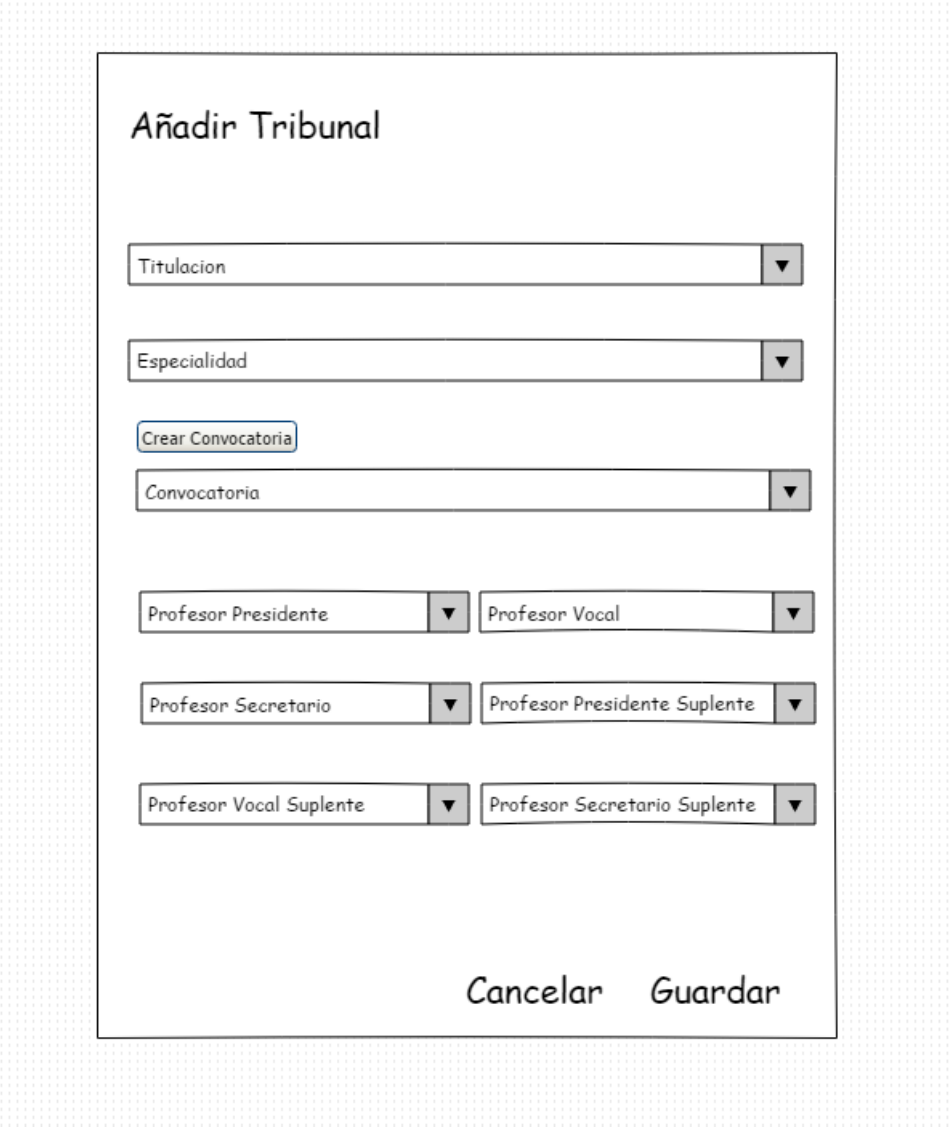


Figura 24. Modal Añadir tribunal

6.4.5 Informes

Los informes con los que contara la aplicación son de dos tipos, uno en formato PDF y el otro en formato TXT. El diseño del informe en formato PDF, seguirá las directrices aconsejadas por el Jefe de Estudios, es decir, en el encabezado tiene que estar el logo de la Universidad y un separador, en el Footer el logo de la Escuela y los datos de contacto. El diseño relevante será el del PDF, por eso solo se incluirá este prototipo.

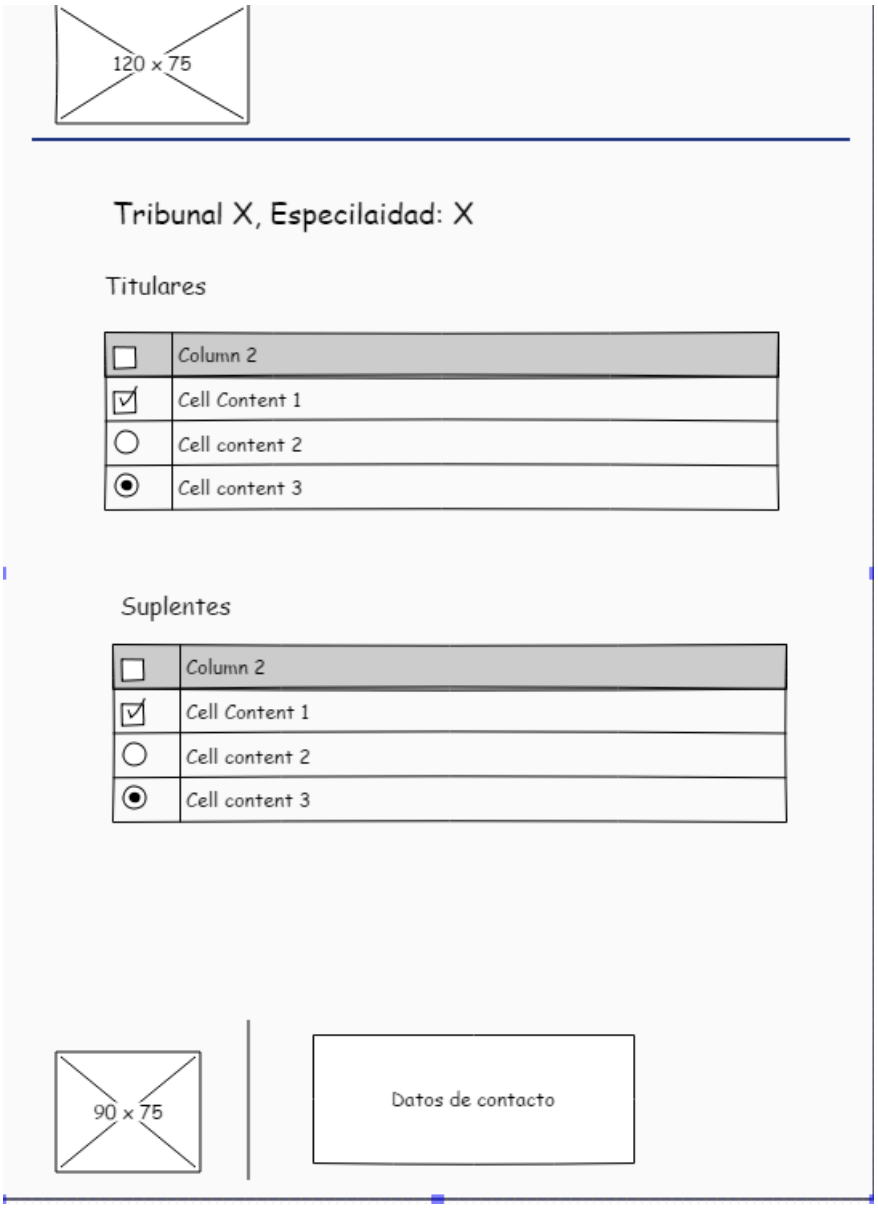


Figura 25. Diseño Informe

6.4.6 Conclusiones

Como hemos podido ver el desarrollo de prototipos es una buena forma para diseñar una interfaz de usuario de forma rápida y poder hacernos una pequeña idea de cómo quedara el resultado final. Obviamente un prototipo no lo podemos considerar una versión definitiva del producto porque conforme se vayan sucediendo las diferentes iteraciones, estos prototipos están sujetos a cambios, aun así, si el diseño es robusto, podríamos decir que se acercaría bastante al diseño final.

El jefe de Estudios, el usuario final de la aplicación, en una de las reuniones que se tuvieron, dio el visto bueno a estos prototipos de diseño y una vez validados, pudimos seguir a la siguiente fase.

7. Tecnologías y Herramientas

7.1 Introducción

En este punto, describiremos las tecnologías que se han usado para llevar a cabo el proyecto y las herramientas que se han necesitado durante todo el proceso de desarrollo software. Estas tecnologías son las más usuales típicas y usuales que nos podemos encontrar en un desarrollo de una aplicación web. Por otro lado, las herramientas usadas, han sido elegidas, por ser de código abierto y gratuitas y por preferencia personal del desarrollador.

7.2 Contexto tecnológico

En este apartado se van a mostrar y explicar todas las tecnologías y herramientas usadas para llevar a cabo el desarrollo de la aplicación. Al ser una aplicación web, existen dos partes bien diferenciadas, el frontend, la parte cliente y el backend la parte del servidor. Pero antes definiremos que es Laravel, puesto que este Framework es la base de nuestra aplicación.

Para la persistencia de los datos se ha hecho uso de sistema de gestión de bases de datos (SGDB), por lo que se ha facilitado mucho la tarea de trabajar con los datos.

Las demás herramientas que se van a describir en el siguiente punto, han sido de vital importancia para el desarrollo de este proyecto y que el proceso de prototipado e implementación del código, como también tener acceso a simular las peticiones Http por parte de Postman, han facilitado el proceso de implementación en gran medida.

Laravel

Laravel [9] es un Framework de código abierto para el desarrollo de aplicaciones y servicios web. La filosofía de este Framework es la desarrollar código en PHP, de forma elegante, simple que permita el uso de una sintaxis expresiva para crear código de forma sencilla permitiendo multitud de funcionalidades. Gran parte de este Framework está formado por dependencias, especialmente de Symfony, esto a su vez implica que el desarrollo de Laravel está basado en sus dependencias.

Laravel está basado en el patrón arquitectónico MVC que ya hemos visto anteriormente. Laravel incluye en el modelo un sistema de mapeo de datos relacional llamado ELOQUENT ORM que facilita la creación de estos modelos. Laravel incluye de paquete un sistema de procesamiento de plantillas llamado Blade, aunque en este proyecto se ha decantado por no usarlo, y usar otro tipo de tecnología para la vista, como veremos más adelante.

Y por último Laravel ofrece soporte para los controladores siendo estos los que dotan al sistema de funcionalidad y permiten organizar el código en clases sin tener que escribirlo todas en las rutas.

Eloquent

Eloquent [10] es lo que se conoce por el nombre de ORM (Object Relational Mapping), un sistema que nos permite llevar la capa de datos y/o persistencia por medio de objetos, es decir, los datos de las tablas se mapean a objetos, permitiendo así evitarnos realizar todas las consultas sobre los datos. El acceso a sus relaciones también se realiza por medio de propiedades de objetos, lo que nos facilita el acceso a la información, siempre que haya una relación de dependencia entre las distintas entidades que conforman nuestro sistema.

Este ORM implementa el patrón "Active Record", un patrón que permite almacenar en una base de datos relacional objetos que se encuentran en memoria. Esto lo hace por métodos ya definimos save(), update() o delete(), permitiendo así una abstracción de la escritura en la base de datos. En "Active Record" una tabla está relacionada con una clase. La clase es lo que se conoce en Laravel como el Modelo.

7.2.1 FrontEnd

VueJs

VueJS [11] es un Framework progresivo para la construcción de interfaces de usuario. A diferencia de otros, VueJs está diseñado desde sus inicios para ser adoptado incrementalmente. La biblioteca principal de este Framework se enfoca en la capa de vista, por lo que es muy simple de utilizar y de integrar con otros proyectos y bibliotecas existentes.

Una de las características de este Framework es su reactividad, es decir, el nuestro sistema podremos escalar los datos como nosotros queramos sin tener que preocuparnos de cuándo y cómo actualizarlos. También cuenta con un sistema de modularización en los que en estos componentes se albergara el código HTML, CSS y JavaScript necesario para dotar al módulo de funcionalidad y funcionar como una pieza independiente. Estas piezas se van componiendo en un árbol jerárquico de componentes hasta formar nuestro sistema.

Vuetify

Vuetify [12] es un Framework que combina la potencia de VueJs con la estética de Materal Desing. Permite acelerar el desarrollo de aplicaciones web, permitiendo incorporar gran cantidad de componentes listo para usar.

Vuetify se basa en el sistema tipo Grid para la ordenación del layout de la página. Dispone de una enorme librería de componentes que incluyen todo tipo de elementos de interfaces graficas. Todo ellos se pueden configurar de diversas opciones para cambiar por completo su estética y comportamiento.

Vuetify es un Framework de código abierto y tiene el apoyo de una gran comunidad que trabajan de forma activa para incorporar nuevos elementos.

7.2.2 BackEnd

PHP

PHP [13] es un lenguaje de programación de uso general que se adapta al desarrollo web. Fue creado originalmente por el programador y matemático Rasmus Lerdorf en 1994. Fue evolucionando sin una especificación formal estándar hasta el año 2014, a partir de este año se ha ido poco a poco trabajando para crear una especificación formar escrita.

Fue creado para la creación de páginas web dinámicas. Para llegar a esto el servido cuenta con un intérprete que suele estar instalado en forma de modulo, que se encarga de interpretar el código que le llega de una petición HTTP. Cuando el servidor ha procesado la petición este devuelve una respuesta ya convertida en texto HTML.

Una de las ventajas de usar PHP es su gran simplicidad para ser usada y aprendida por las personas principiantes que se están adentrando en el mundo del desarrollo de aplicaciones o servicios web. A su vez, también ofrece un amplio abanico de características para aquellos desarrolladores avanzados, para proyectos más complejos.

Actualmente PHP es un lenguaje muy popular para el desarrollo de páginas web, contando con varios Frameworks, como, por ejemplo, Laravel.

REST

REST [14] es un estilo de arquitectura software para sistemas de hipermedia distribuidos como la WWW. En un principio REST se refería a un conjunto de principios de arquitectura, pero con los años ha ido derivando a un sentido más amplio, para describir cualquier interfaz entre sistemas que utilice el protocolo HTTP, para obtener datos o realizar acciones sobre esos datos, en cualquier formato.

Las ventajas de utilizar REST son:

- Protocolo Cliente/Servidor sin estado, es decir, cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Esto permite que, tanto el cliente como el servidor no tengan que recordar ningún estado sobre las comunicaciones.
- Conjunto de operaciones bien definidas, que se aplican a todos los recursos. Estas operaciones son POST, GET, PUT, DELETE, estas operaciones se equiparan a las operaciones CRUD.
- Sintaxis universal para identificar recursos.
- Las representaciones habituales de los estados en un sistema REST son típicamente en HTML Y XML.

MySQL

MySQL [15] es un sistema de gestión de bases de datos de código abierto creado por MySQLAB. Actualmente goza de gran popularidad por su fiabilidad, rendimiento y su sencillez a la hora de integrarlo en cualquier proyecto software.

MySQL se utiliza en la mayoría de las aplicaciones web de tipo CMS, junto a PHP. Esto es a causa de su baja concurrencia en la modificación de los datos, pero muy alta en su lectura, esto hace que destaque en la rapidez que es capaz de consultar la información. Gracias a estos factores actualmente es uno de los SGBD más populares del mercado.

Axios

Axios [16] es una librería de JavaScript que puede ejecutarse en el navegador y que nos permite realizar operación como cliente HTTP, por lo que podemos realizar solicitudes a un servidor y recibir respuestas fáciles de procesar.

Para utilizar AXIOS utilizaremos el API de las denominadas promesas, es decir, cuando recibamos una respuesta del servidor, se llamar a un callback configurado en el *then* y en el momento que arroje un error, se correrá la misma función, definida por un *catch*.

Algunas de las ventajas que tiene trabajar con esta librería son:

- API unificada.
- Esta altamente pensado para el consumo de servicios web, API REST que devuelvan un JSON.
- Es una librería muy sencilla de usar y puede ser un complemento ideal en sistemas en los que no se esté usando JQuery (librería de JavaScript) y otras librerías como REACT.
- Es una librería muy liviana.
- Es compatible con todos los navegadores actuales.

Axios es una alternativa idea para aplicaciones donde se requiera el acceso a recursos REST, con un peso muy ligero agrega mucho valor gracias a una cantidad de opciones de configuración sobre las solicitudes.

7.2.3 Herramientas

Git

Software de gestión de versiones creado por Linus Torvalds, pensado para que cuando un proyecto este gestionado por un alto número de desarrolladores y se tengan gran cantidad de ficheros, estos se traten con confiabilidad y eficiencia.

Visual Studio Code

Software de editor de código fuente, de código libre, creado por Microsoft, pensado para Windows, Linux y Mac. Cuenta con soporte para la depuración, control integrado de versiones, resaltado de sintaxis, etc… Es compatible con multitud de lenguajes de programación. También podemos conectar de forma sencilla a un repositorio, para guardar nuestro código. Es compatible con plugins que dotan al editor de diversas funcionalidades, como, por ejemplo, resaltado de código, indentación automática del código, etc… Cuenta con un debug, para poder depurar nuestro codigo.

MySQL Workbench

Software de diseño de bases de datos que integra desarrollo software, diseño de bases de datos. También integra el sistema gestor de bases de datos de MySQL.

Homestead

Es un paquete de oficial de Vagrant que proporciona un entorno de desarrollo sin la necesidad de tener instalado PHP, un servidor web y ningún otro software adicional en nuestro equipo. Homestead es una máquina virtual que se integra con Laravel para comenzar nuestro proyecto de forma inmediata.

Pencil

Es una herramienta diseñada para con el objetivo de proporcionar una interfaz para el desarrollo de prototipos, modelos. Es una aplicación multiplataforma, gratuita y de Open Source. Pencil como herramienta para el modelado de prototipos nos pone a disposición varias colecciones de formas para poder crear diferentes tipos de interfaces, desde aplicaciones webs a sistemas móviles. La mayoría de estas colecciones vienen integradas, pero también contamos con soporte online para descargarnos otras e instalarlas fácilmente.

Postman

Postman es una herramienta software que nace como una extensión para el popular navegador Google Chrome. Esta herramienta nos permite crear peticiones HTTP sobre una API de una forma sencilla y poder probar, de manera rápida y eficiente simulando las peticiones de un navegador, con esto podremos comprobar cuales son los resultados de estas peticiones sobre los recursos solicitados. Nos ofrece multitud de utilidades adiciones para poder gestionar nuestras APIs de una forma sencilla. Nos proporciona herramientas para documentar, realizar una monitorización de las APIs. También nos permite la posibilidad de crear teams para trabajar de forma colaborativa.

El uso de Postman es gratuito, aun así, nos ofrece planes de pago que nos ofrecen características adicionales a las anteriormente citadas. Para el desarrollo de nuestro sistema, con la versión gratuita ha sido suficiente.

8. Implementación

El uso de un Framework como Laravel ha propiciado que el desarrollo de la aplicación web, haya sido algo ágil y dinámico, pero a la vez seguro. Ha facilitado en gran medida, la creación de los controladores, los modelos y las vistas, necesarias. Una vez que se han definido todos los requisitos, se ha tenido el visto bueno a los prototipos y se ha instalado y configurado todas las herramientas, se ha podido crear un proyecto Laravel. Este proyecto, viene ya preconfigurado y con un esqueleto definido. Una vez se ha mirado y comprendido la estructura que nos proporciona el Framework se ha procedido con la implementación de la solución en los lenguajes anteriormente explicados.

8.1 Estructura del proyecto

En este apartado explicare como se ha organizado el proyecto de forma interna, la estructura de carpetas, el esqueleto, todo esto lo genera el propio Framework una vez se ha creado el proyecto. En nuestro caso particular no se ha usado el paquete Blade para las vistas que viene por defecto en Laravel, si no que las vistas se han organizado utilizando componentes, usando VueJs.

Como podemos observar en la *Figura 25*, Laravel organiza el proyecto en diferentes directorios y archivos de configuración, de las cuales resaltaremos los directorios app, database, resources y routes. De los archivos de configuración el más relevante y el único que se comentara en este apartado será el .env.

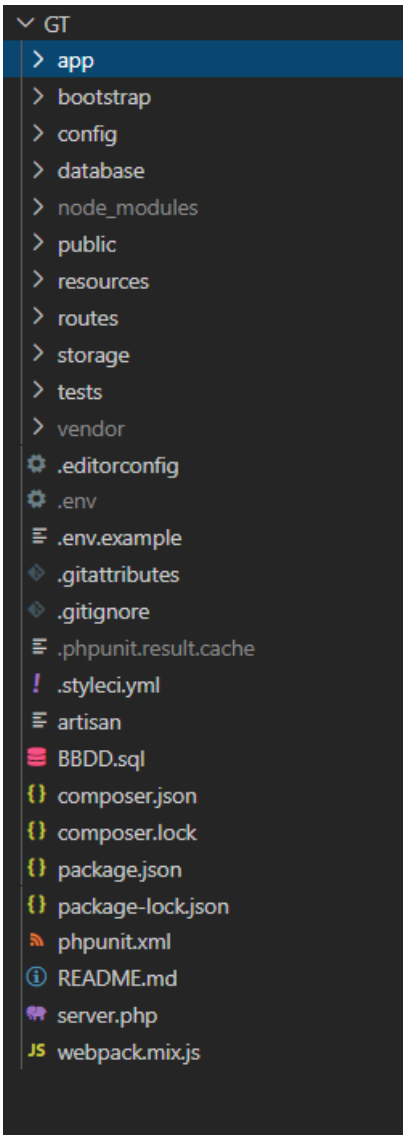


Figura 26. Estructura proyecto

8.1.1 Directorio App

En este directorio podemos encontrar diversas carpetas, y los modelos correspondientes encargados del mapeo a la base de datos. En las diferentes carpetas del directorio, las más importantes son el manejador de excepciones, la carpeta Http donde se encuentran nuestros controladores, que son los encargados de dotar al sistema de funcionalidad.

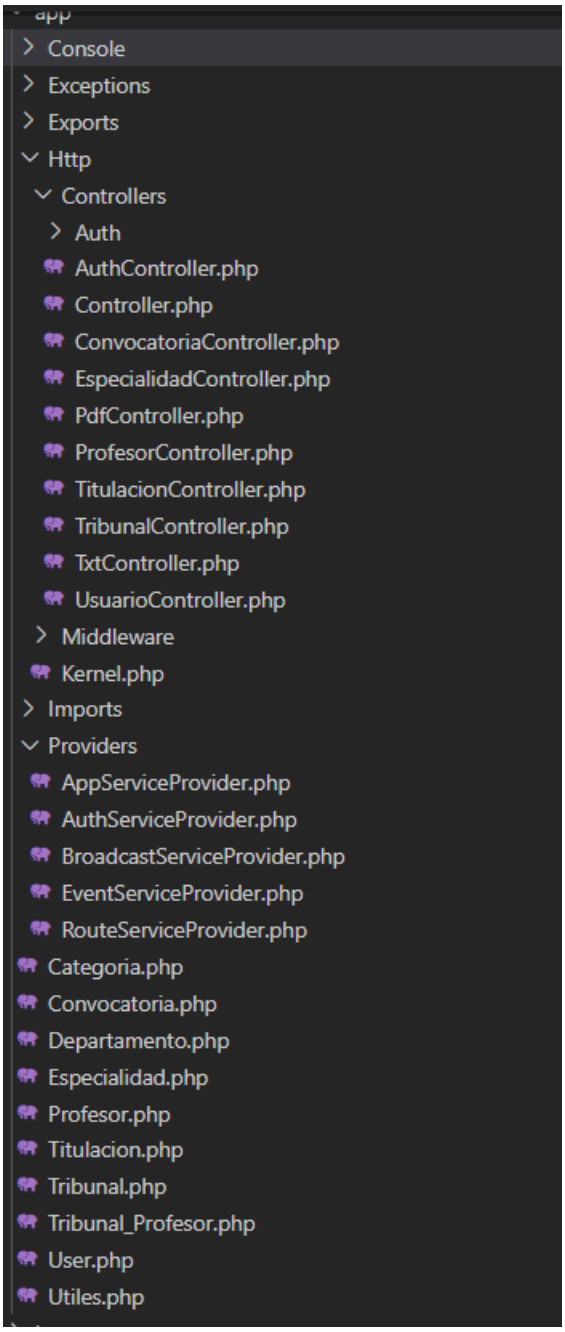


Figura 27. App

En puntos posteriores se darán algunos del código escrito en los modelos y en algunos controladores.

Y por último los providers, que son los archivos que se encargan de crear las instancias de otros objetos para hacer que la aplicación funcione. Estas instancias son las encargadas también de crear las instancias que conformar el núcleo de Laravel. Los providers ya vienen con todo lo necesario para que la aplicación funcione, pero también podemos crearlos en caso de necesitarlos.

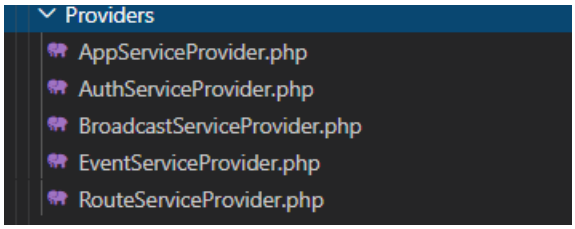


Figura 28. Providers

8.1.2 Directorio Database

En este directorio nos encontramos con las migraciones. Esta carpeta es las más relevante dentro de este apartado, puesto que contiene los ficheros encargados de crear las tablas y los campos de nuestra base de datos.

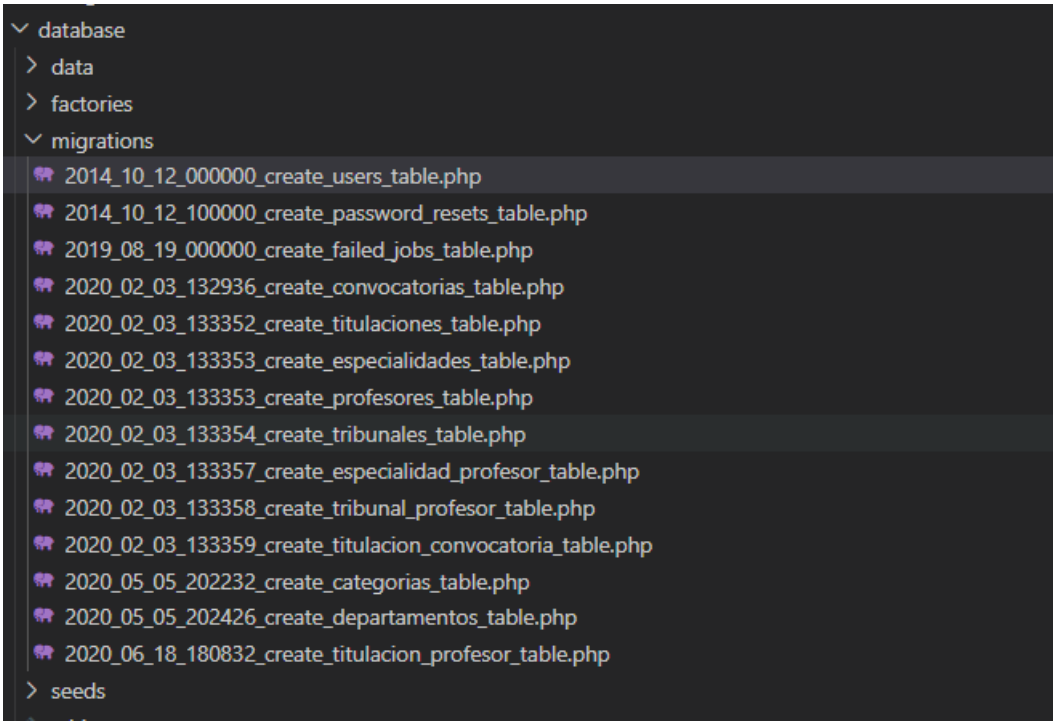


Figura 29. Database

Estos archivos son los encargados de crear nuestra base de datos, es decir, nosotros como desarrolladores tan solo tenemos que definir los campos y las relaciones entre las diferentes tablas y automáticamente, el Framework creara el código SQL necesario para crear la base de datos, obviamente necesitamos un sistema gestor de bases de datos y una base de datos con su usuario para poder acceder al código generado por estas clases.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('pid')->nullable();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Figura 30. Migration

También podríamos hablar de los seeders pero la única función que tiene es la de insertar datos de prueba, si estamos en un entorno de desarrollo o si así lo decidimos, crear los datos del entorno de producción y que se regeneren una vez queramos subir la aplicación a un servidor.

8.1.3 Directorios Resources y Routes

El directorio resources contiene varias carpetas, pero la que atañe, donde se encuentran las vistas de este proyecto es la de js. También está la carpeta views es la que viene por defecto en el Framework y como se ha comentado anteriormente, se ha decidido no hacer uso del paquete Blade, que son simplemente archivos HTML con directivas PHP.

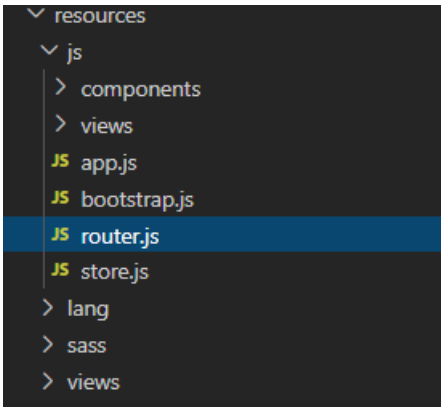


Figura 31. Resources

En la raíz de la carpeta js, se encuentras los archivos JavaScript de configuración. En estos archivos se encuentran varias configuraciones internas del proyecto y la configuración, realizada de forma personalizada, de la navegación entre los diferentes componentes, es decir, las rutas que después se verán en un navegador web. Como se puede ver en la *Figura 31*, las rutas login y home están a la misma altura, pero con la diferencia que, para acceder a home, primero se tiene que estar autenticado. A partir de la ruta home, todas las demás son rutas anidadas. Las rutas están asociadas a un componente. Estos componentes son las vistas de nuestro sistema.

```
// Routes
const router = new VueRouter({
    mode: 'history',
    linkActiveClass: 'is-active',
    routes: [
        {
            path: '/',
            name: 'login',
            component: Login,
        },
        {
            path: '/home',
            name: 'home',
            component: Home,
            meta: {
                requiresAuth: true,
            },
            children: [
                {
                    path: '/profesores',
                    name: 'profesores',
                    component: Profesores,
                    meta: {
                        requiresAuth: true,
                    }
                },
                {
                    path: '/convocatorias',
                    name: 'convocatorias',
                    component: Convocatorias,
                    meta: {
                        requiresAuth: true,
                    }
                },
                {
                    path: '/titulaciones',
```

Figura 32. navegación entre componentes

En la carpeta components se encuentran todas las vistas que vera el usuario, en la aplicación, utilizando la tecnología VueJs y Vuetify anteriormente explicada.

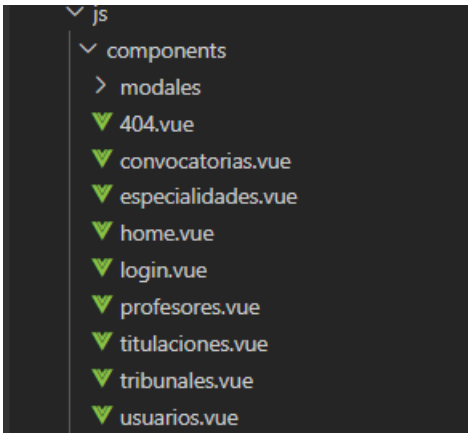


Figura 33. Componentes.

Para finalizar el directorio routes, que posiblemente sea uno de los directorios más importantes del Framework, contiene los archivos del sistema de rutas de Laravel. Como se puede ver en la *Figura 33.*

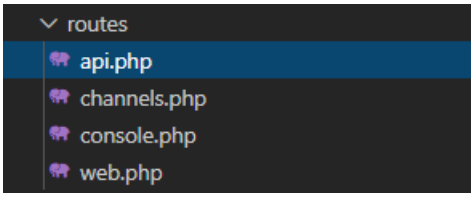


Figura 34. Routes

Este directorio tiene 4 archivos, de los cuales los más importantes son, api.php donde definiremos todas las rutas para crear APIs en la aplicación. Estas rutas las podremos definir de muchas maneras, dependiendo si la aplicación está dotada por un proceso de autenticación o no, que en el caso de nuestro software si cuenta con este proceso, lo que hace que todas las rutas que queramos ocultar a usuarios no registrados entren dentro del middleware “auth”. También podremos agregar parámetros, para las operaciones que precisen de ellos. También podremos crear prefijos y así poder tener organizadas en grupos.

```php
    return $request->user();
});

Route::post('login', 'AuthController@login')->name('login');

Route::group(['prefix' => 'pdf'], function() {
    Route::post('/crearPdf/{pid}', 'PdfController@crearPdf');
    Route::post('/crearMultiplePdf/{pids}', 'PdfController@crearMultiplePdf');
});
Route::group(['prefix' => 'txt'], function() {
    Route::post('/crearTxt/{pid}', 'TxtController@crearTxt');
});
Route::group(['middleware' => 'auth:api'], function() {
    Route::get('logout', 'AuthController@logout');
    Route::get('Authuser', 'AuthController@user');
    Route::post('crearUsuario', 'AuthController@createUsuario');
});

Route::group(['middleware' => 'auth:api'], function() {
    Route::group(['prefix' => 'admin'], function() {
        Route::group(['prefix' => 'profesores'], function() {
            Route::get('/getProfesores', 'ProfesorController@verProfesores');
            Route::post('/crearProfesor', 'ProfesorController@createProfesor');
            Route::post('/editarProfesor/{pid}', 'ProfesorController@updateProfesor');
            Route::get('/eliminarProfesor/{pid}', 'ProfesorController@deleteProfesor');
            Route::get('/eliminarProfesores/{pids}', 'ProfesorController@deleteProfesores');
            Route::post('/crearTodos', 'ProfesorController@importCsv');
        });
        Route::group(['prefix' => 'departamentos'], function() {
            Route::get('/getDepartamentos', 'ProfesorController@verDepartamentos');
            Route::post('/crearDepartamento', 'ProfesorController@createDepartamento');
            Route::post('/editarDepartamento/{pid}', 'ProfesorController@updateDepartamento');
            Route::get('/eliminarDepartamento/{pid}', 'ProfesorController@deleteDepartamento');
        });
        Route::group(['prefix' => 'categorias'], function() {
            Route::get('/getCategorias', 'ProfesorController@verCategorias');
            Route::post('/crearCategoria', 'ProfesorController@createCategoria');
            Route::post('/editarCategoria/{pid}', 'ProfesorController@updateCategoria');
            Route::get('/eliminarCategoria/{pid}', 'ProfesorController@deleteCategoria');
        });
        Route::group(['prefix' => 'titulaciones'], function() {
            Route::get('/getTitulaciones', 'TitulacionController@verTitulaciones');
            Route::post('/crearTitulacion', 'TitulacionController@createTitulacion');
            Route::put('/editarTitulacion/{pid}', 'TitulacionController@updateTitulacion');
            Route::delete('/eliminarTitulacion/{pid}', 'TitulacionController@deleteTitulacion');
            Route::get('/eliminarTitulaciones/{pids}', 'TitulacionController@deleteTitulaciones');
        });
        Route::group(['prefix' => 'convocatorias'], function() {
            Route::get('/getConvocatorias', 'ConvocatoriaController@verConvocatorias');
            Route::post('/crearConvocatoria/{pid_titulacion}', 'ConvocatoriaController@createConvocatoria');
            Route::get('/getConvocatoriasTitulacion/{pid}', 'ConvocatoriaController@getConvocatoriasTitulacion');
            Route::put('/editarConvocatoria/{pid}', 'ConvocatoriaController@updateConvocatoria');
            Route::delete('/eliminarConvocatoria/{pid}', 'ConvocatoriaController@deleteConvocatoria');
            Route::get('/eliminarConvocatorias/{pids}', 'ConvocatoriaController@deleteConvocatorias');
        });
        Route::group(['prefix' => 'especialidad'], function() {
            Route::get('/getEspecialidades', 'EspecialidadController@verEspecialidades');
            Route::post('/crearEspecialidad/{pid_titulacion}', 'EspecialidadController@createEspecialidad');
            Route::put('/editarEspecialidad/{pid}', 'EspecialidadController@updateEspecialidad');
            Route::delete('/eliminarEspecialidad/{pid}', 'EspecialidadController@deleteEspecialidad');
            Route::get('getEspecialidades_titulacion/{pid}', 'EspecialidadController@getEspecialidades_titulacion');
            Route::get('/eliminarEspecialidades/{pids}', 'EspecialidadController@deleteEspecialidades');
```

Figura 35. Codigo api.php

Y para finalizar el archivo web.php, donde se definen las rutas de la web, que, en nuestro caso particular, solo tendrá referenciada la ruta del archivo index.html.

```php
Route::get('/{any?}', function (){
    return view('index');
    })->where('any', '^(?!api\/)[\/\w\.-]*');
Auth::routes();
```

Figura 36. Codigo web.php

8.2 API REST

Como se ha explicado en puntos anteriores, nuestro sistema va a usar una API REST, que consumiremos desde nuestra aplicación. Para realizar peticiones a la API, simplemente se deben hacer peticiones Http con verbos adecuados (POST, GET, PUT…) a los diferentes recursos del servicio. A continuación, se mostrarán los servicios disponibles.

Login e Informes

Tarea	Método	Ruta
Obtener el token de autenticación.	POST	api/login
Destruye el token de autenticación y cierra sesión.	GET	api/logout
Crea un informe PDF de un tribunal.	POST	api/pdf/crearPdf /{pid}
Crea un informe en PDF con múltiples tribunales	POST	api/pdf/crearMultiplePdf/{pids}
Crea un informe en TXT de un tribunal.	POST	api/txt/crearTxt/{pids}

Profesores

Tarea	Método	Ruta
Obtener una lista de profesores.	GET	api/admin/profesores/getProfesores
Crear un profesor nuevo.	POST	api/admin/profesores/crearProfesor
Editar un profesor existente.	PUT	api/admin/profesores/editarProfesor/{pid}
Eliminar un profesor existente.	PUT	api/admin/profesores/eliminarProfesor/{pid}
Eliminar múltiples profesores existentes.	DELETE	api/admin/profesores/eliminarProfesores/{pids}

Titulaciones

Tarea	Método	Ruta
Obtener una lista de titulaciones.	GET	api/admin/titulaciones/getTitulaciones
Crear una titulación nueva.	POST	api/admin/ titulaciones /crearTitulaciones
Editar una titulación existente.	PUT	api/admin/ titulaciones /EditarTitulación/{pid}
Eliminar una titulación existente.	PUT	api/admin/ titulaciones /eliminarTitulación/{pid}
Eliminar múltiples titulaciones existentes.	DELETE	api/admin/titulaciones/eliminarTitulaciones/{pids}

Usuarios

Tarea	Método	Ruta
Obtener una lista de usuarios.	GET	api/admin/usuarios/getUsuarios
Crear un usuario nuevo.	POST	api/admin/usuarios/crearUsuario
Editar un usuario existente.	PUT	api/admin/usuarios/EditarUsuario/{pid}
Eliminar un usuario existente.	PUT	api/admin/usuarios/eliminarUsuario/{pid}
Eliminar múltiples usuarios existentes.	DELETE	api/admin/ usuarios /eliminarTitulaciones/{pids}

Tribunales

Tarea	Método	Ruta
Obtener una lista de tribunales.	GET	api/admin/tribunales/getTribunales
Crear un tribunal nuevo.	POST	api/admin/ tribunales /crearTribunal
Editar un tribunal existente.	PUT	api/admin/ tribunales /EditarTribunal/{pid}
Eliminar un tribunal existente.	DELETE	api/admin/ tribunales /eliminarTribunal/{pid}
Generar tribunales múltiples.	POST	api/admin/ tribunales /genenerarTribunal
Cambia un profesor del tribunal.	POST	api/admin/tribunales/cambiarProfesor
Obtiene todos los profesores de un tribunal.	GET	api/admin/tribunales/getProfesoresTribunales

Especialidades

Tarea	Método	Ruta
Obtener una lista de especialidades.	GET	api/admin/especialidades/getEspecialidades
Crear una especialidad nueva.	POST	api/admin/ especialidades /crearEspecialidad
Editar una especialidad existente.	PUT	api/admin/ especialidades /EditarEspecialidad/{pid}
Eliminar una especialidad existente.	DELETE	api/admin/especialidades/eliminarEspecialidad/{pid}
Eliminar múltiples especialidades existentes.	DELETE	api/admin/especialidades/eliminarEspecialidades/{pids}

Convocatorias

Tarea	Método	Ruta
Obtener una lista de convocatorias.	GET	api/admin/convocatorias/getConvocatorias
Crear una convocatoria nueva.	POST	api/admin/convocatorias/crearConvocatoria
Editar una convocatoria existente.	PUT	api/admin/ convocatorias /EditarConvocatoria/{pid}
Eliminar una convocatoria existente.	DELETE	api/admin/convocatorias/eliminarConvocatoria/{pid}
Eliminar múltiples convocatorias existentes.	DELETE	api/admin/convocatorias/eliminarConvocatorias/{pids}

8.3 Modelo Vista Controlador

El patrón arquitectónico MVC que se implementa en la aplicación consta de tres partes bien diferenciadas que como se ha explicado anteriormente consta del Modelo, de la Vista y del Controlador.

Los controladores están situados en la carpeta app/http/controllers, como se ha visto en el punto anterior. Para crear un controlador en Laravel, tan solo tendremos que ejecutar este comando.

```
vagrant@homestead:~/code/gt$ php artisan make:controller profesoresController
```

Figura 37. Make:Controller.

Con este sencillo comando podremos crear un controlador, que veremos un ejemplo de su implementación en el siguiente apartado.

Estos controladores, tienen la lógica de negocio de nuestro sistema. Los métodos de estos controladores, suelen retornar arrays, objetos o vistas, en el caso de nuestro sistema solo se devolverán arrays con datos o mensajes de texto.

Los controladores, actúan como interfaz entre los modelos y las vistas, aplicando las transformaciones y lógicas necesarias.

Los modelos se crean al igual que los controladores con un comando.

```
vagrant@homestead:~/code/gt$ php artisan make:model Profresor
```

Figura 38. Make Model

Los modelos referencian las estructuras de los datos de la aplicación. Los datos pueden ser transferidos desde una base de datos, una clase u desde otra entidad. Los datos que referencian los modelos, en nuestra aplicación serán transformados desde el controlador para ser retornados a la base de datos o a las vistas. Se ofrecerá también, un ejemplo real del sistema.

Para finalizar, las vistas son la representación de la información en forma de interfaz de usuario. En nuestra aplicación, como se ha hecho uso de componentes de VueJs y Vuetify, se crearan a mano, simplemente dándoles una extensión .vue. Los datos pueden venir directamente del modelo o del controlador, que previamente habrá transformado los datos según los requisitos de negocio. Al igual que con el controlador y el modelo, daremos un ejemplo de implementación de las vistas.

En la *Figura 39*, se puede apreciar cómo funciona esta arquitectura de una forma más práctica.

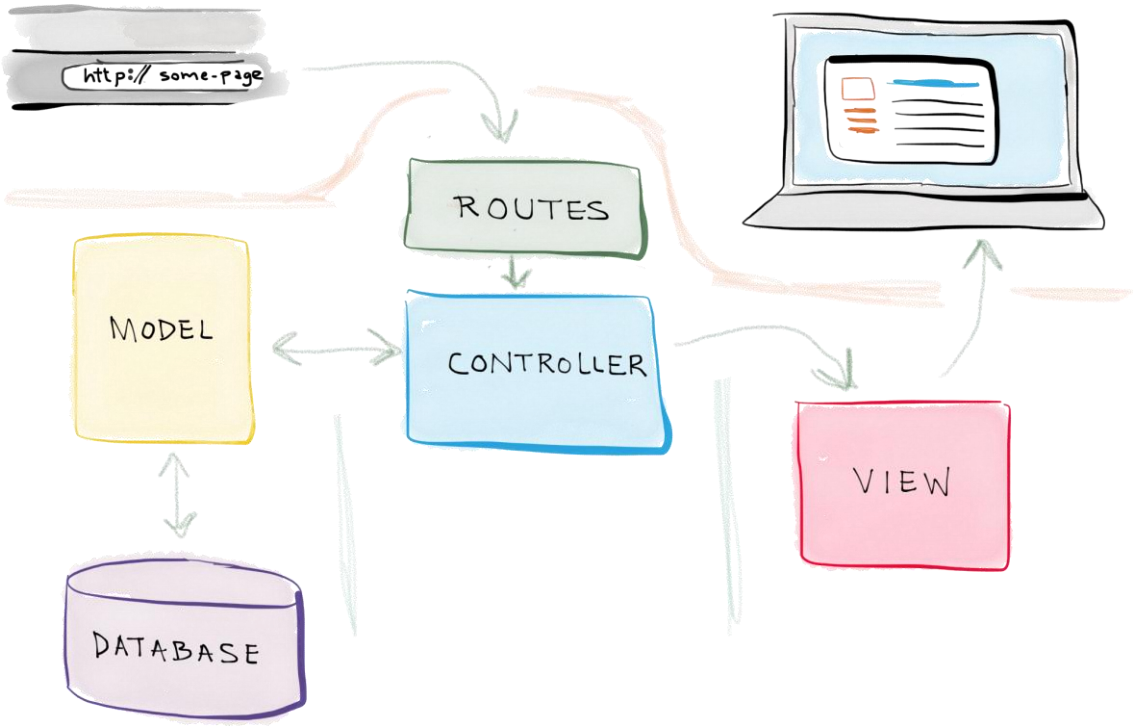


Figura 39. Representación MVC

En la imagen podemos ver como desde el navegador llamamos a una, esta ruta es una petición Http solicitando un recurso a nuestra API REST. Estas rutas son gestionadas por el control de rutas de Laravel, que sabe cómo tiene que redireccionar las peticiones. Una vez hecho esto, si el recurso solicitado es una petición DELETE, el controlador, hará uso del

modelo correspondiente y este realizará la acción de borrar el dato de la base de datos. Una vez realizada esta acción, el controlador devolverá lo que tenga que devolver para que el usuario desde la vista sea consciente de que la acción ha sido llevada a cabo con éxito.

8.3.1 Controladores

A continuación, se va a mostrar pequeños fragmentos de código de los controladores donde esta se ha programado la lógica de la aplicación. Se muestran varios ejemplos del controlador de ProfesorController y de TribunalController.

```
{
    public function verProfesores()
    {
        $profesores = Profesor::with('titulaciones')->get();
        foreach ($profesores as $profesor) {
            foreach($profesor->titulaciones as $titulacion){
                $profesor->titulacion .= $titulacion->nombre . ', ';
            }
            $profesor->titulacion = substr($profesor->titulacion, 0, -2);
        }
        return json_decode($profesores);
    }

    public function createProfesor(Request $request)
    {
        $profesor = Profesor::create(array(
            'pid' => Profesor::generate(),
            'nombre_apellidos' => $request->nombre_apellidos,
            'email' => $request->email,
            'centro' => $request->centro,
            'especialidad' => $request->especialidad,
            'departamento' => $request->departamento,
            'tribunal_participa' => $request->tribunal_participa,
            'sexo' => $request->sexo,
            'categoria' => $request->categoria,
            'antiguedad' => $request->antiguedad
        ));
        if($profesor){
            $pids = explode(',', $request->pid_titulacion);
            foreach($pids as $pid){
                $id_titulacion = Titulacion::where('pid', $pid)->first()->id;
                $profesor->titulaciones()->attach($id_titulacion,
                    array(
                    'pid' => Profesor::generate(),
                    'id_profesor' => Profesor::where('pid', $profesor->pid)->first()->id
                    )
                );
            }
            return $profesor;
        }
        else
            return "A ocurrido un error al crear el profesor";
    }
```

Figura 40. ProfesorController

En la *Figura 40*, se puede apreciar el código perteneciente al controlador ProfesorController, donde se ven dos métodos, en concreto el método de verProfesores() donde se accede al modelo Profesores y nos trae una lista con todos los profesores de la base de datos, para posteriormente devolverlos. Y también está el método createProfesor(), que a través de un formulario que se encuentra en la parte de la vista, insertamos un profesor en la base de datos.

```php
public function componerTribunal(Request $request){
    $tribunales = [];
    $profesores = [];
    $titulacion = Titulacion::where('pid', $request->pid_titulacion)->first();
    $roles = ['Presidente', 'Vocal', 'Secretario', 'Presidente Suplente', 'Vocal Suplente', 'Secretario Suplente'];
    for($i = 0; $i < $request->numero; $i++){
        if($request->pid_especialidad){
            $especialidad = Especialidad::where('pid', $request->pid_especialidad)->first();
            $profesoresEspecialidad = $titulacion->profesores()->where('tribunal_participa', '<', 3)
                                        ->where(function ($query) use ($especialidad){
                                            if($especialidad){
                                                $query->where('especialidad', 'like', '%' . $especialidad->nombre . '%');
                                            }
                                        })->get()->toArray();

            $otrosProfesores = $titulacion->profesores()->where('tribunal_participa', '<', 3)
                                        ->where(function ($query) use ($especialidad){
                                            if($especialidad){
                                                $query->where('especialidad', 'not like', '%' . $especialidad->nombre . '%');
                                            }
                                        })->get()->toArray();
            $profesores = $this->asignarProfesoresTitulares($profesoresEspecialidad, $otrosProfesores);
            if($profesores){
                $tribunal = Tribunal::create(array(
                    'pid' => Tribunal::generate(),
                    'id_especialidad' => $especialidad->id,
                    'id_titulacion' => $titulacion->id,
                    'id_convocatoria' => Convocatoria::where('pid', $request->pid_convocatoria)->first()->id
                ));
            }else{
                return 'Ha ocurrido un error al asignar los profesores de la especialidad: ' . $especialidad->nombre;
            }
        }else{
            $profesoresTitulacion = $titulacion->profesores()->where('tribunal_participa', '<', 3)
            ->get()->toArray();
            $profesores = $this->asignarProfesoresTitulares($profesoresTitulacion);
            if($profesores){
                $tribunal = Tribunal::create(array(
                    'pid' => Tribunal::generate(),
                    'id_especialidad' => null,
                    'id_titulacion' => $titulacion->id,
                    'id_convocatoria' => Convocatoria::where('pid', $request->pid_convocatoria)->first()->id
                ));
            }else{
                return 'Ha ocurrido un error al asignar los profesores de la titulacion: ' . $titulacion->nombre;
            }
        }
        if($tribunal){
            for($j = 0; $j < 6; $j++){
                if($j < 3){
                    $tribunal_profesor = Tribunal_Profesor::create(array(
                        'pid' => Tribunal_Profesor::generate(),
                        'pid_tribunal' => $tribunal->pid,
                        'id_profesor' => Profesor::where('pid', $profesores[$j]['pid'])->first()->id,
                        'id_tribunal' => $tribunal->id,
                        'rol_tribunal' => $roles[$j],
                        'nombre_apellidos' =>  $profesores[$j]['nombre_apellidos'],
                        'especialidad' => $profesores[$j]['especialidad'],
                        'antiguedad' => $profesores[$j]['antiguedad'],
                        'titular' => true
                    ));
                    $profesor = Profesor::where('pid', $profesores[$j]['pid'])->first();
                    $profesores[$j]['tribunal_participa']++;
                    if($profesor->update($profesores[$j]));
                }else{
                    $tribunal_profesor = Tribunal_Profesor::create(array(
                        'pid' => Tribunal_Profesor::generate(),
```

Figura 41. TribunalController

El fragmento de la *Figura 41*, es un trozo de código del método componerTribunal() este método se encarga de preparar y crear las estructuras de datos y llamar a los métodos que contienen el algoritmo de la conformación de los tribunales, que retornaran los tribunales conformados con los profesores titulares y los suplentes.

8.3.2 Modelos

Como se ha explicado en puntos anteriores la arquitectura MVC trabaja con modelos que son los encargados de representar la información y los accesos a esta. Estos accesos a la base de datos pueden ser de escritura o lectura. Como sabemos Laravel trabaja con el ORM Eloquent, y en última instancia es este quien nos facilita el trabajo de realizar las consultas a través de los métodos que tiene predefinidos y que nos permitirán realizar, las tareas más comunes y que más se suelen repetir cuando trabajamos con una base de datos. En la *Figura 37*, se mostrará el modelo de Profesor, y se explicará brevemente las peculiaridades del ORM que se está utilizando.

```php
<?php

namespace Gestion_tribunales;

use Illuminate\Database\Eloquent\Model;
use Gestion_tribunales\Utiles;

class Profesor extends Utiles
{
    protected $table = 'profesores';
    protected $fillable = ['pid', 'nombre_apellidos', 'email', 'centro', 'especialidad', 'departamento',
                           'max_tribunales', 'tribunal_participa', 'convocatoria', 'sexo', 'categoria', 'antiguedad'];
    protected $hidden = ['id'];

    public function especialidades()
    {
        return $this->belongsToMany('Gestion_tribunales\Especialidad', 'especialidad_profesor', 'id_profesor', 'id_especialidad');
    }
    public function titulaciones()
    {
        return $this->belongsToMany('Gestion_tribunales\Titulacion', 'titulacion_profesor', 'id_profesor', 'id_titulacion');
    }
    public function tribunales()
    {
        return $this->belongsToMany('Gestion_tribunales\Tribunal', 'tribunal_profesor', 'id_profesor', 'id_tribunal');
    }
}
```

Figura 42. Modelo Profesor

En este modelo se está usando el ORM Eloquent, en las funciones especialidades(), titulaciones() y tribunales(). En estas funciones le indicamos al modelo las relaciones N a M que tenemos en la base de datos. Posteriormente en los controladores, cuando queramos recoger los datos de estas tablas intermedias, tan solo tendremos que llamar a las funciones aquí indicadas.

Como se puede observar el lenguaje es muy sencillo de utilizar y claro de visualizar. En el vector $fillable, le estamos indicando al modelo que esos campos son en los que tiene que insertar los datos cuando hagamos una petición de inserción en la base de datos. También podemos crear algunas funciones, que servirán para relacionar nuestros modelos de datos y poder, por ejemplo, crear tablas intermedias de las relaciones N a M, sin la necesidad de crear modelos adicionales.

Estas tablas son denominadas tablas pivote, y con esta tabla pivote, podemos unir dos entidades y solo necesitaremos los identificadores de las dos entidades. Opcionalmente también podremos agregar campos adicionales.

Por otro lado, se ha creado un modelo adicional, donde se ha creado un algoritmo para crear el Pid (Public Identifier), y así dotar a las llamadas Http a la API, que se realizaran desde el cliente. Cuando se necesite realizar una acción, por ejemplo, de borrar algún dato, en vez de solicitarse un Id, se solicitará un Pid.

```php
class Utiles extends Model
{
    public static function generate($length = 10) {
        return substr(str_shuffle("0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ@"), 0, $length);
    }
}
```

Figura 43. Generación Pid

8.3.3 Vistas

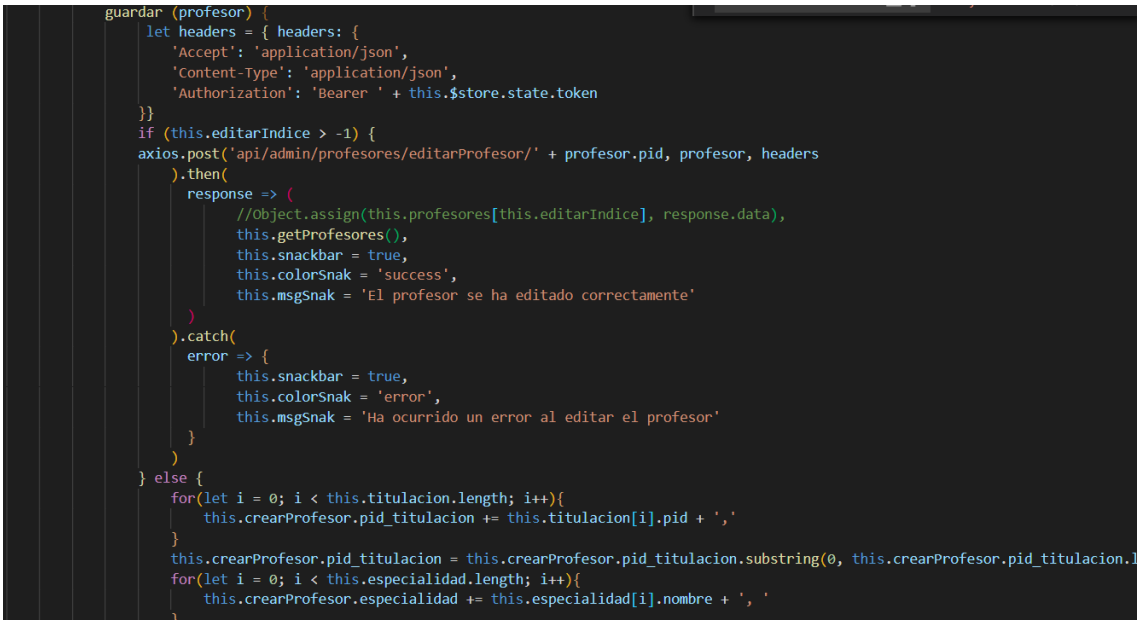
Las vistas en la arquitectura que usa nuestro sistema, son las encargadas de mostrar los datos y ejecutar código JavaScript en el cliente. Este código, sirve para realizar algunas de las acciones básicas como, por ejemplo, realizar las llamadas Http, con Axios, y devolver la respuesta enviada por los controladores. Estas respuestas en ultima se mostrarán junto con las vistas en el navegador en formato HTML.

A continuación, se van a mostrar varias capturas referentes a las vistas, que contendrán llamadas Axios para realizar peticiones Http a nuestra API REST, código de VueJS y Vuetify que conforman nuestras vistas.

```
<template>
    <v-layout align-start>
        <v-flex>
            <v-toolbar flat color="white">
                <v-toolbar-title>
                    Profesores
                </v-toolbar-title>
                <v-spacer/>
                <v-icon>mdi-file-pdf</v-icon>
            </v-toolbar>
            <v-data-table ···
            </v-data-table>
            <v-snackbar ···
            </v-snackbar>
            <v-dialog v-model="dialogborrar" :return-value.sync="profeBorrar" persistent max-width="500">
                <v-card>
                    <v-card-title>¿Esta seguro que desea borrar este profesor?</v-card-title>
                    <v-card-actions>
                    <v-spacer></v-spacer>
                            <v-btn
                                color="blue darken-1"
                                text
                                @click="borrarProfesor(profeBorrar)">Aceptar
                            </v-btn>
                            <v-btn
                                color="blue darken-1"
                                text
                                @click="dialogborrar = false">Cancelar
                            </v-btn>
                    </v-card-actions>
                </v-card>
```

Figura 44. Vista Profesores, Vuetify

En el fragmento de código que se observa en la *Figura 44*, se puede ver cómo esta implementado Vuetify en nuestro proyecto, mostrando un modal y varios botones que servirán de eventos para darle funcionalidad a la aplicación.



```
guardar (profesor) {
    let headers = { headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + this.$store.state.token
    }}
    if (this.editarIndice > -1) {
    axios.post('api/admin/profesores/editarProfesor/' + profesor.pid, profesor, headers
        ).then(
          response => (
                //Object.assign(this.profesores[this.editarIndice], response.data),
                this.getProfesores(),
                this.snackbar = true,
                this.colorSnak = 'success',
                this.msgSnak = 'El profesor se ha editado correctamente'
          )
        ).catch(
          error => {
                this.snackbar = true,
                this.colorSnak = 'error',
                this.msgSnak = 'Ha ocurrido un error al editar el profesor'
          }
        )
    } else {
        for(let i = 0; i < this.titulacion.length; i++){
            this.crearProfesor.pid_titulacion += this.titulacion[i].pid + ','
        }
        this.crearProfesor.pid_titulacion = this.crearProfesor.pid_titulacion.substring(0, this.crearProfesor.pid_titulacion.l
        for(let i = 0; i < this.especialidad.length; i++){
            this.crearProfesor.especialidad += this.especialidad[i].nombre + ', '
```

Figura 45, Vista Profesores, VueJS

En la *Figura 45*, se observa la función guardar(), que es simplemente la función que se llama cuando le damos al botón enviar, del formulario añadir profesor. También podemos ver que cuando lanzamos la llamada de AXIOS, funciona con promesas como se ha explicado anteriormente y simplemente ponemos la ruta donde está el método al que queremos llamar y le pasamos los parámetros que corresponda y si la llamada al método es satisfactoria nos devolverá el profesor que se ha insertado en la base de datos. Y si ha ocurrido un error se nos informara a través de snackbar.

9 Aplicación Final

El sistema este diseñado para que lo use una única persona, puesto que va a ser una aplicación interna de la ETSINF, mayormente la usara el Jefe de Estudios de la misma. Por lo que los datos asociados a los profesores, titulaciones y usuarios administradores se meterán a la base de dato nada más cargar el sistema en producción.

En este punto se enseñará el resultado final de la aplicación.

9.1 Login

El primer paso para poder acceder a la aplicación es mediando el logueo del usuario. El login se lleva a cabo rellenando un campo Email y otro campo Contraseña. Una vez el usuario ha puesto sus credenciales en el formulario, puede marcar el check para que se guarde su usuario y contraseña o no, es totalmente opcional.

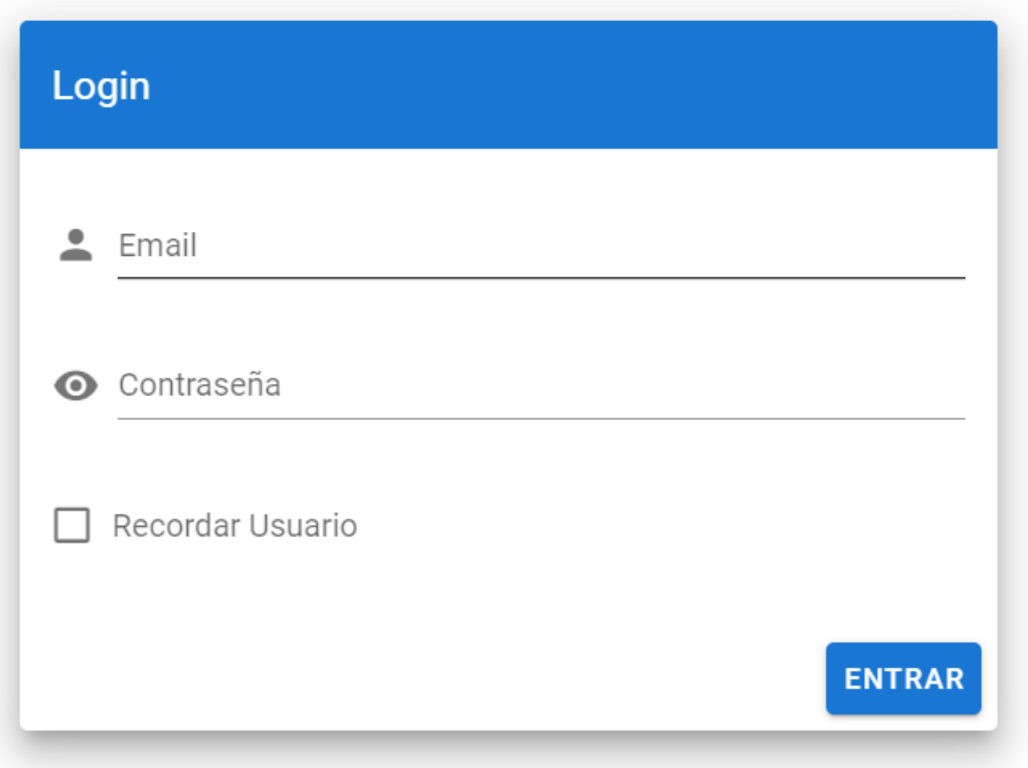


Figura 46. Login

Cuando el usuario le da al botón Entrar, si las credenciales son válidas, al pulsar el botón entrar, al usuario se le redireccionara a la siguiente vista, ya dentro del sistema. Por el contrario, si las credenciales no se encuentran en el sistema se le avisara al usuario que las credenciales, no son válidas, como se muestra en la *Figura 57*.

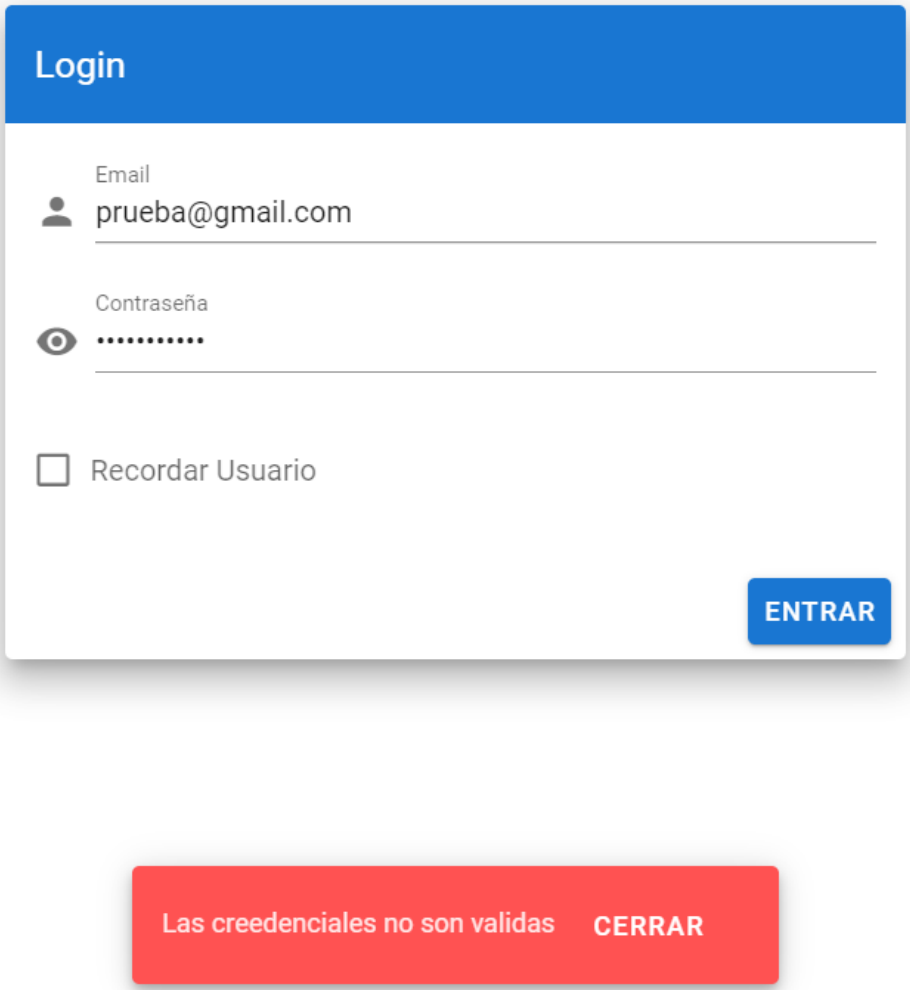


Figura 47. Creedenciales incorrectas

Si el usuario rellena los campos con datos no validos o los deja en blanco, el propio formulario avisara de que los campos deben de estar rellenos y además de forma correcta.

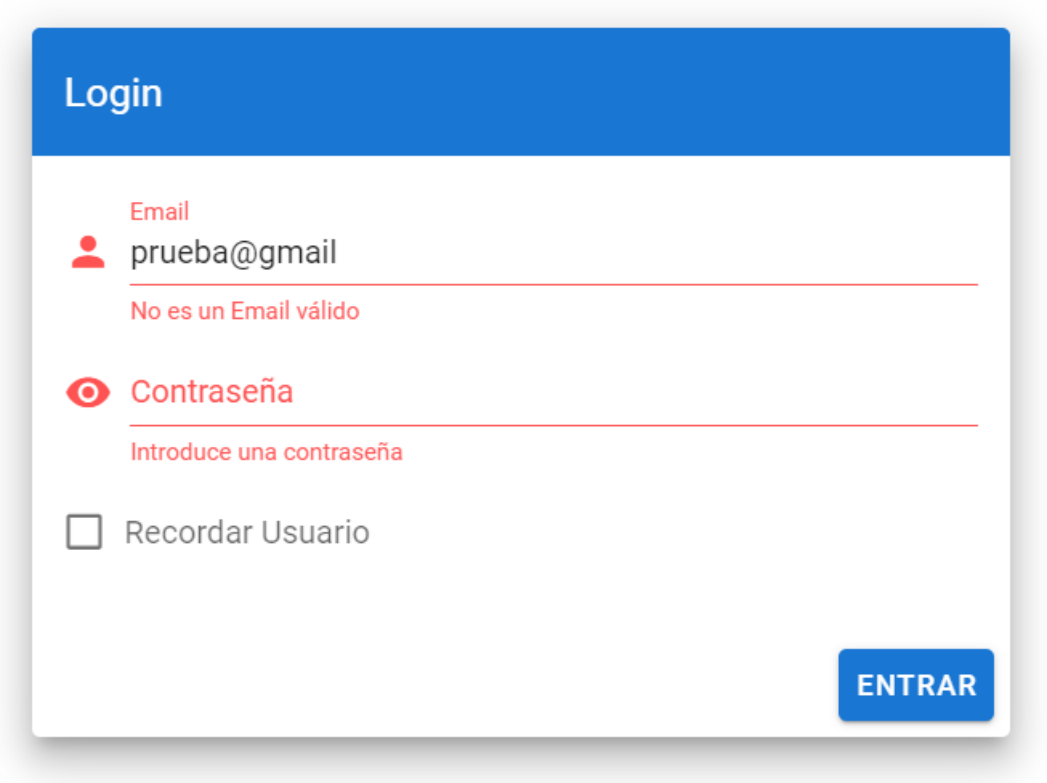


Figura 48. Campos no válidos

Una vez que se han puesto las credenciales correctas, como se ha dicho antes el usuario ingresara en la aplicación, en la vista de Titulaciones.

9.2Vistas

En este apartado se mostrarán y se explicaran las vistas que tiene la aplicación final en un entorno real. Las vistas son muy parecidas entre sí por eso y por simplificar que no sea redundante solo se mostrará la vista de titulaciones y la de tribunales, puesto que esta última es una vista un poco diferente a las demás.

9.2.1 Titulaciones

En la vista Titulaciones se muestran los datos de las titulaciones que la ETSINF tiene ofertadas actualmente. Los datos que se muestran en la tabla son el nombre, acrónimo, código y el ERT de las titulaciones. Si la titulación tiene especialidades asociadas, se mostrarán en el desplegable que tiene cada una de ellas. Al final de cada titulación, en la parte derecha hay un icono por cada titulación en forma de dots verticales, que será un pequeño desplegable con diferentes acciones, como se puede observar en la *Figura 49.*

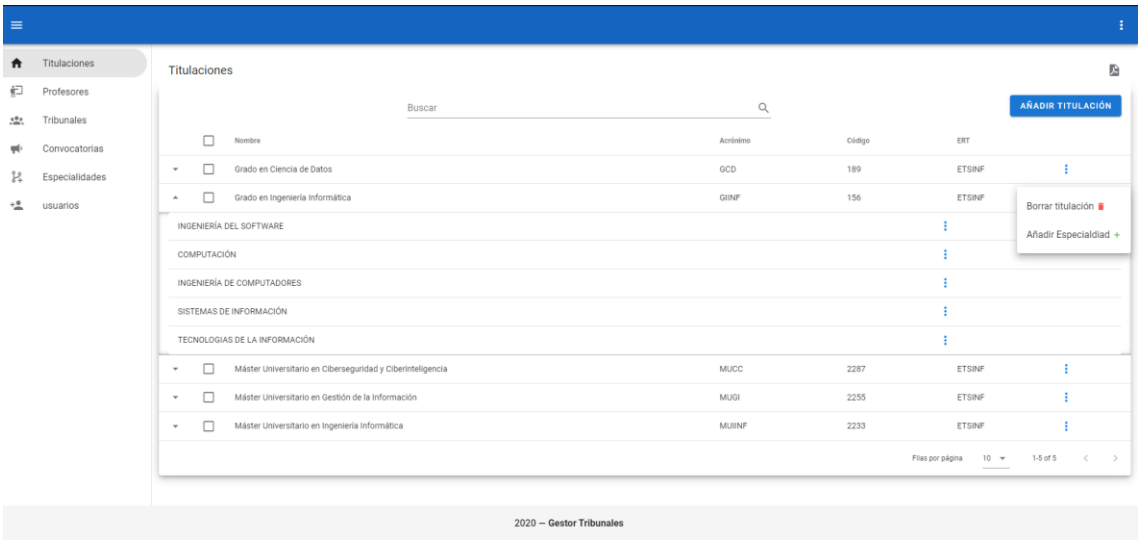


Figura 49. Vista Titulaciones

Al pulsar el botón Añadir titulación, se mostrará un formulario modal para poder crear una nueva titulación. Esta acción es común a todas las vistas, por lo que solo se mostrara una vez para simplificar, aunque se muestre el modal de añadir titulación.

Figura 50. Formulario Titulaciones

Si todo ha ido bien y se ha creado la titulación en la base de datos, saltara un pequeño snackbar para informar de que la inserción ha sido un éxito. Estas notificaciones nos informaran

de si una acción seleccionada por el usuario, ha tenido éxito o si, por el contrario, ha ocurrido un error.

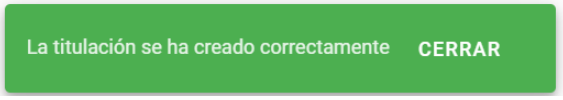


Figura 51. Notificación titulación creada con éxito

De igual manera cuando hacemos click en el dot vertical de puntos de una titulación se nos despliegan una serie de acciones que podemos realizar, en este caso borrar titulación y añadir especialidad. Se nos despliega un modal para confirmar que deseamos borrar esta titulación o un modal para crear una nueva especialidad que se asociara a esa titulación.

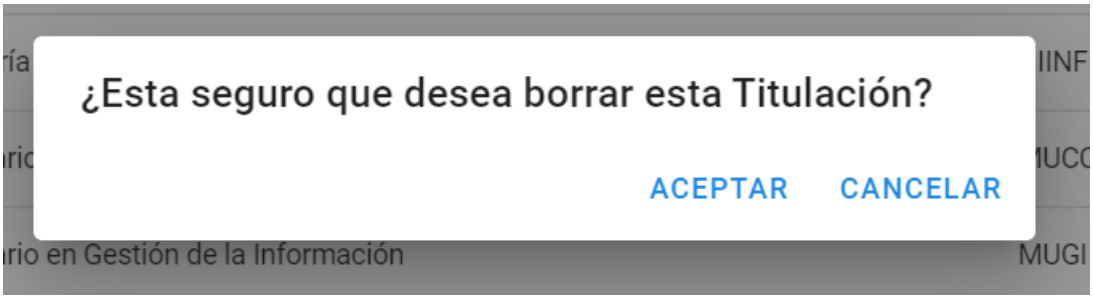


Figura 52. Confirmar borrado

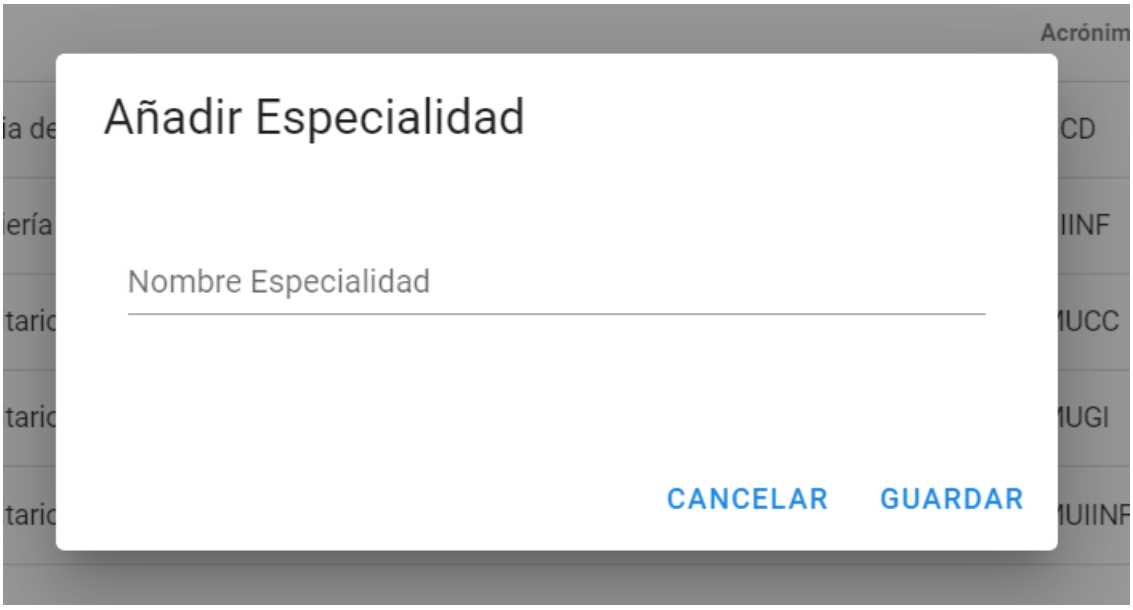


Figura 53. Modal especialidad

También podremos seleccionar todas las filas de la lista, y borrar todas las titulaciones de golpe o las que seleccionemos. Como es lógico, en el caso de las titulaciones, si las borramos, también se borrarán en cascada las especialidades que esta tenga asociada. Cuando

se seleccionan múltiples filas, el botón de Añadir se quita de la pantalla y nos aparece otro para poder borrar todas las filas seleccionadas.

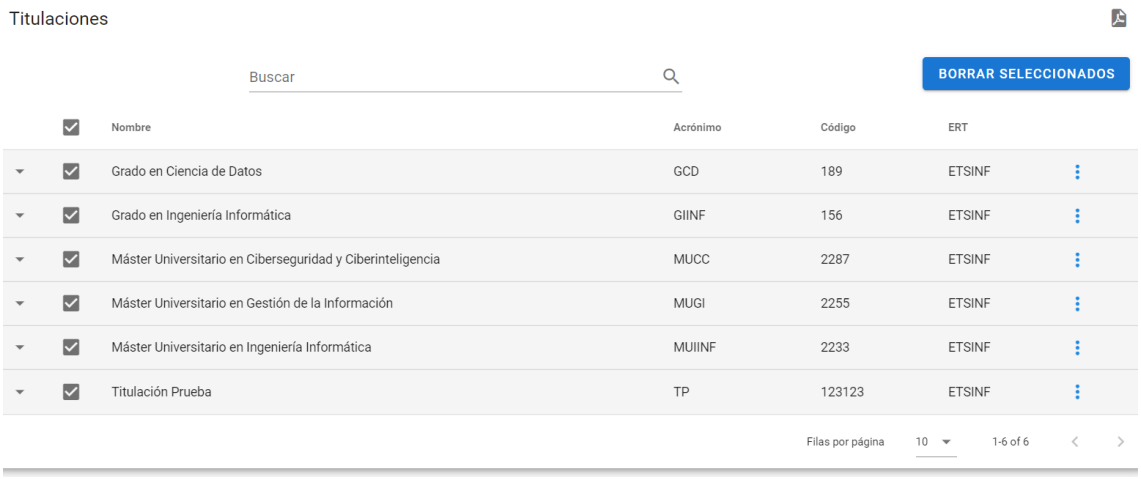


Figura 54. Selección todas las filas

También podemos modificar las titulaciones o cualquier dato, solo tenemos que hacer click encima del nombre de la titulación y podremos modificar el dato.

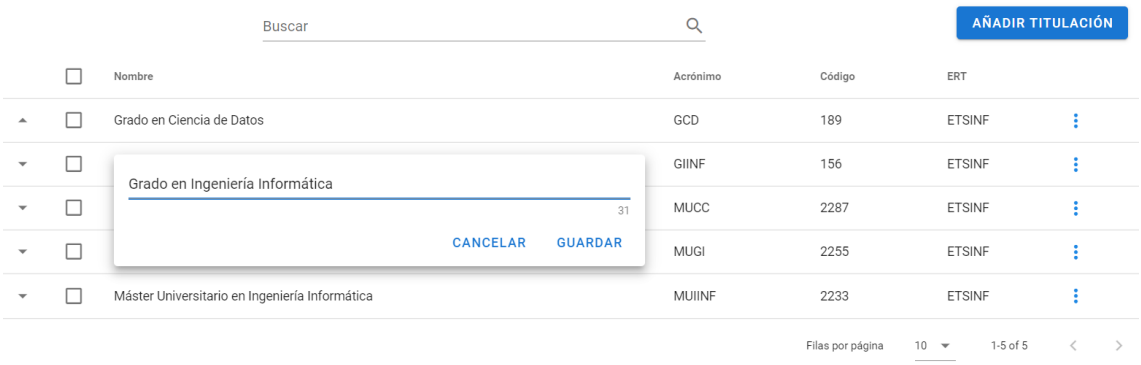


Figura 55. Modificar titulación

Las demás vistas que conforman el sistema como se ha dicho al inicio del apartado, son prácticamente idénticas visualmente, con tan solo pequeños detalles que son adatados a la vista correspondiente.

Como pequeño inciso, comentar que, en la vista de convocatorias, hay un formulario con un campo relativamente especial, es decir, este campo nos muestra un calendario para poder seleccionar el mes de la convocatoria, al seleccionar el mes, de manera automática, en el campo siguiente se nos inserta el año que le corresponde.

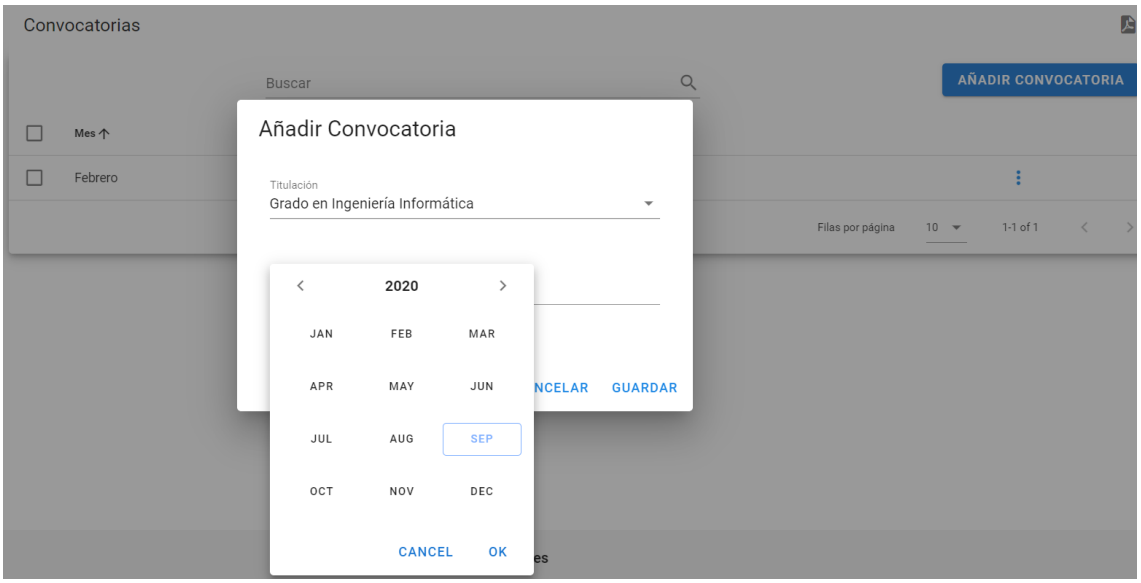


Figura 56. Selección mes.

9.2.2 Tribunales

Esta vista es la más peculiar de todas y por supuesto la más importante y por eso, la tenemos que separar de las demás, visualmente es muy similar, pero tiene ciertas peculiaridades que la hacen diferente.

Para empezar esta vista cuenta con varios botones, uno de ellos para generar los tribunales de forma automática, otro para crearlos de forma manual, seleccionando a los profesores uno a uno y por último otro para resetear los filtros. También cuenta con unos ComboBox encima de la tabla para poder filtrar datos según la titulación, especialidad y el mes de la convocatoria, para poder visualizar los tribunales de una convocatoria, titulación y especialidad concretos.

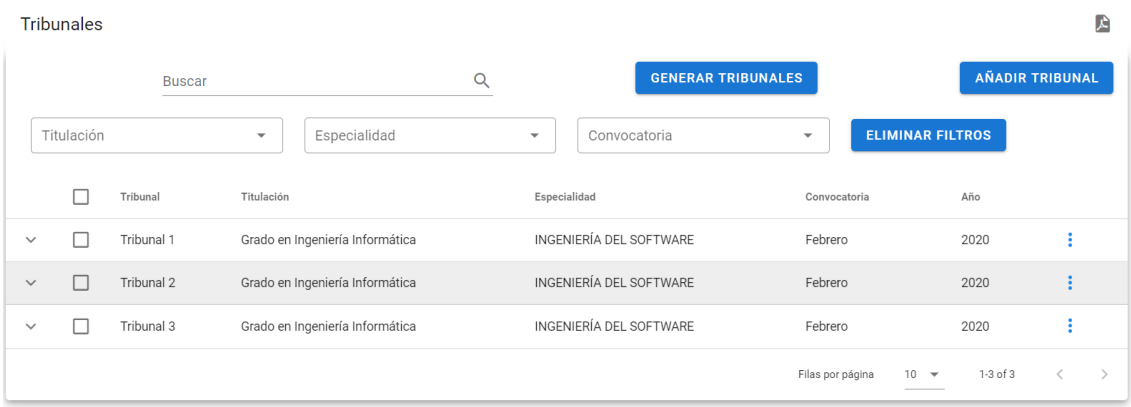


Figura 57. Vista Tribunales.

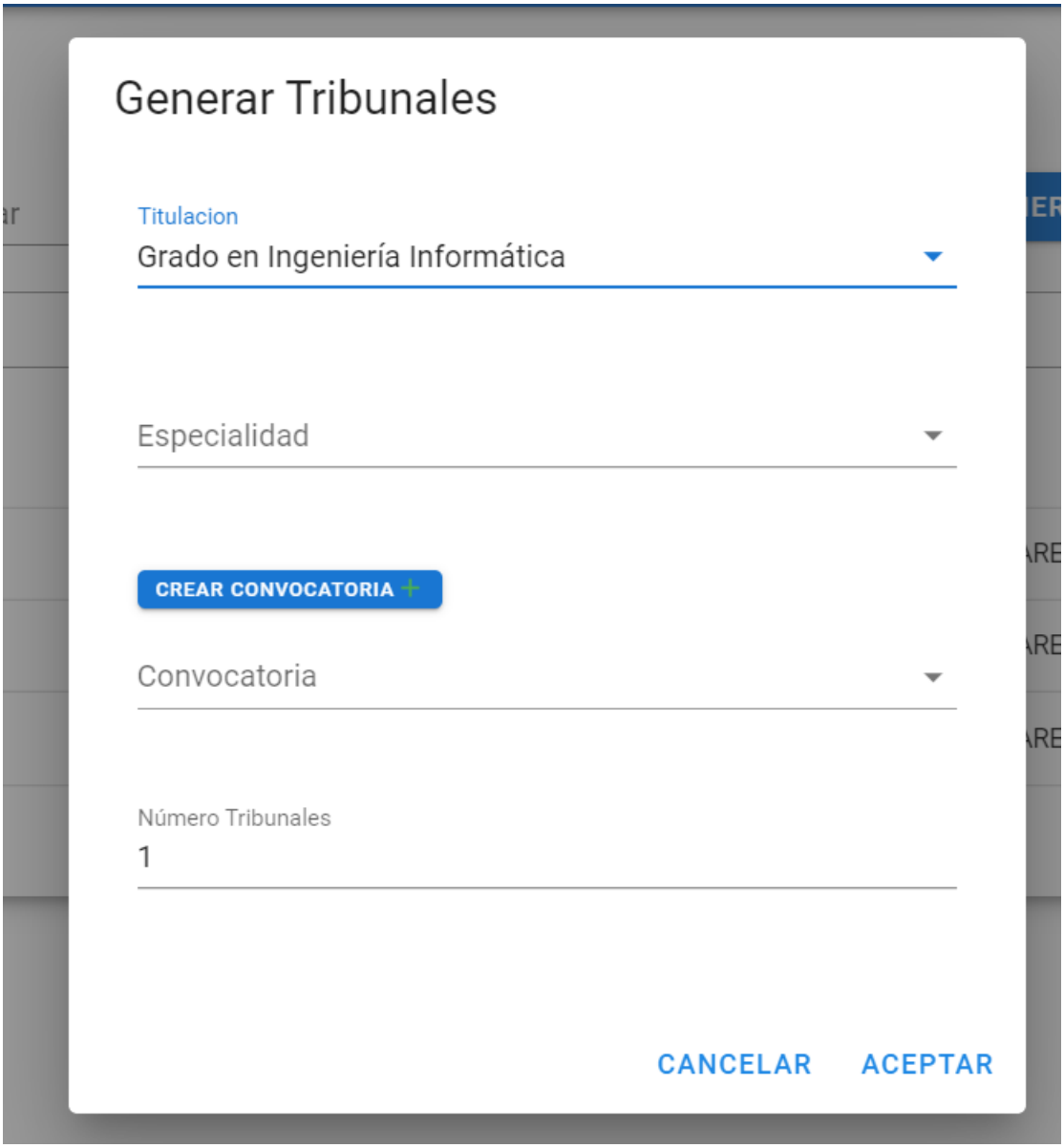


Figura 58. Generar tribunales

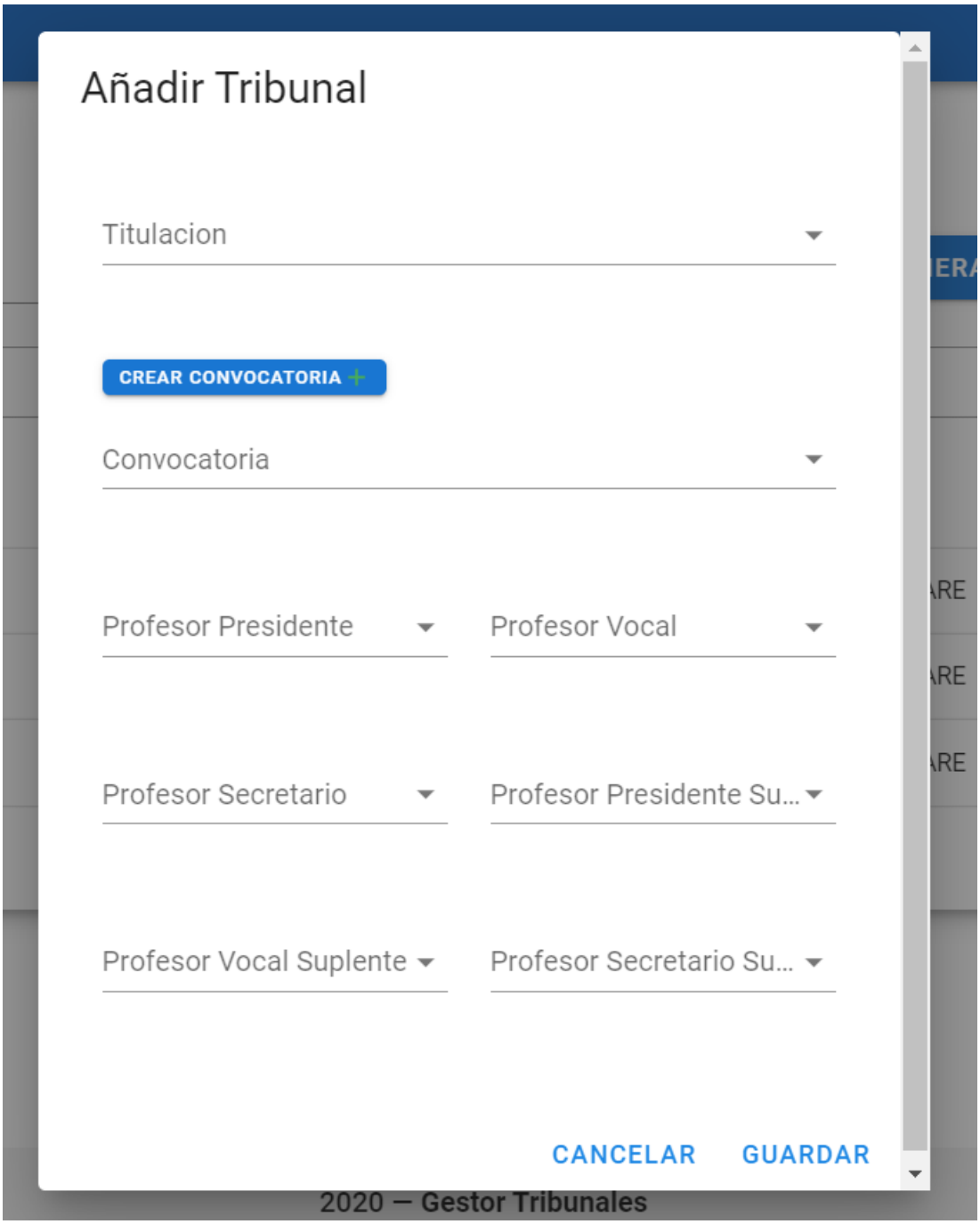


Figura 59. Añadir tribunal

La *Figura 58 y 59*, corresponden con los modales para la creación de los tribunales.

Esta vista también cuenta con acciones adicionales al pulsar los dots verticales de los tribunales, estas opciones aparte de la de borrar el tribunal, también tiene opciones de descargar un archivo en formato TXT y en formato PDF. Con estos documentos podemos visualizar los profesores y sus respectivos cargos en el tribunal.

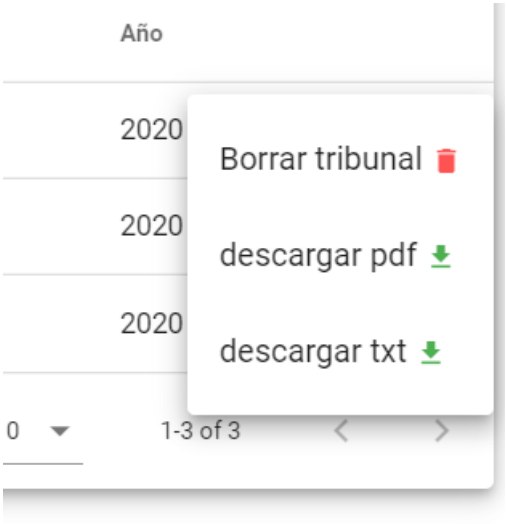


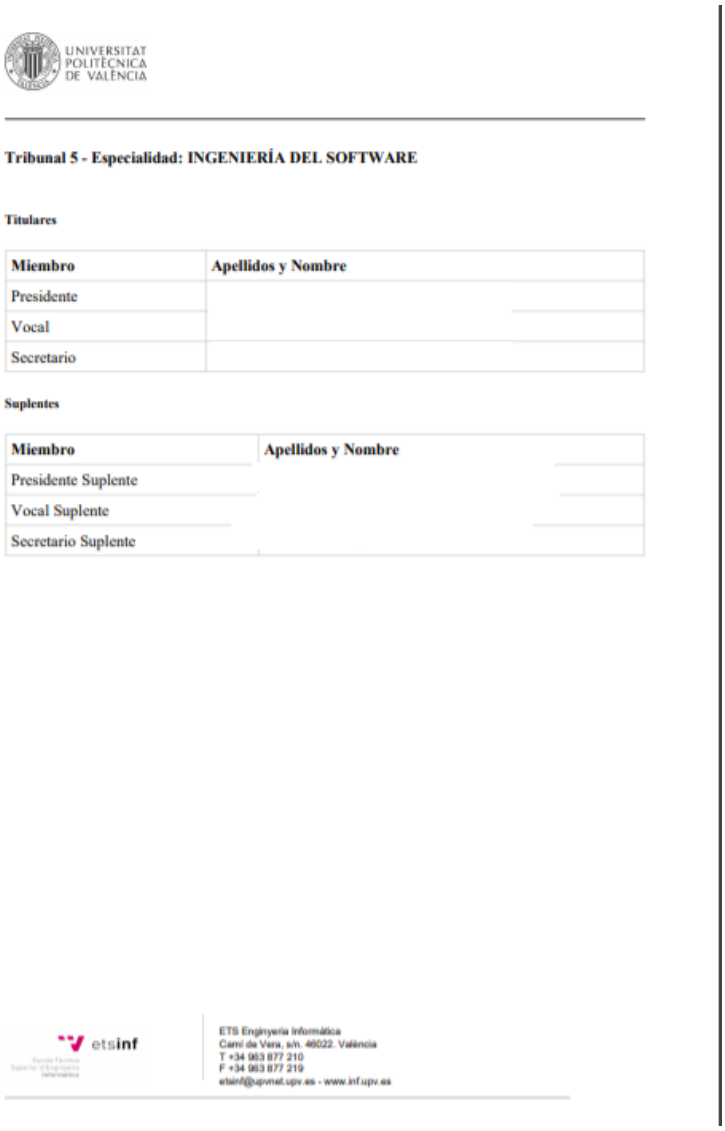
Figura 60. Acciones tribunal

Tenemos la opción como se ve en la *Figura 60*, de descargar un informe de un tribunal en concreto. Al pulsar el botón, se descargará un informe con la información sobre los profesores que componen ese tribunal en concreto. En la siguiente *Figura*, se puede apreciar como quedaría el informe final en este formato, los datos de los profesores se han omitido por privacidad.

```
TRIBUNAL #5 ESPECIALIDAD: INGENIERÍA DEL SOFTWARE:
        TITULARES:
                - PRESIDENTE:
                - VOCAL:
                - SECRETARIO:
        SUPLENTES:
                - PRESIDENTE:
                - VOCAL:
                - SECRETARIO:
```

Figura 61. Informe tribunal Txt

La *Figura 62*, se corresponde con el mismo informe, pero en este caso generado en formato PDF. Los datos de los profesores en este caso al igual que en el anterior se han omitido por privacidad.



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Tribunal 5 - Especialidad: INGENIERÍA DEL SOFTWARE

Titulares

Miembro	Apellidos y Nombre
Presidente	
Vocal	
Secretario	

Suplentes

Miembro	Apellidos y Nombre
Presidente Suplente	
Vocal Suplente	
Secretario Suplente	

etsinf

ETS Enginyeria Informàtica
Camí de Vera, s/n. 46022. València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es

Figura 62. Informe tribunal pdf

Para finalizar, el Jefe de Estudios, también puede seleccionar todos los tribunales de la lista que más le interesen y generar un informe de todos ellos a la vez.

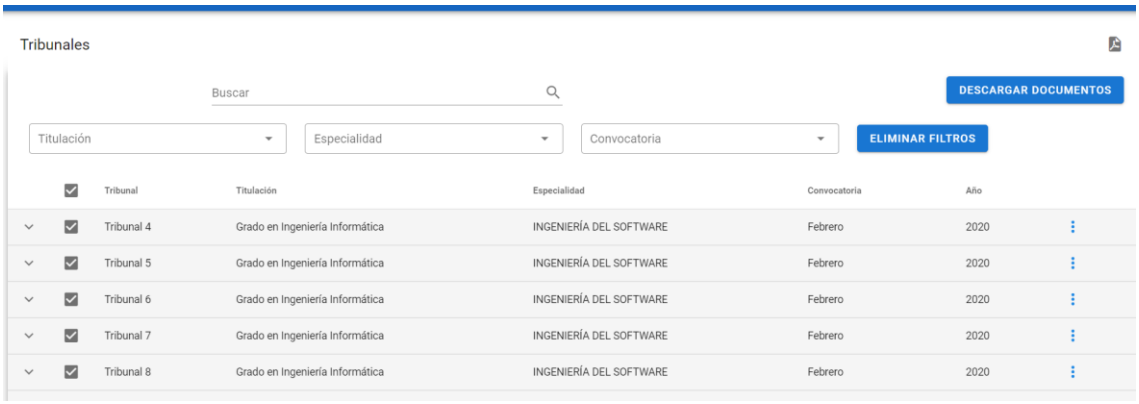


Figura 63. Todos los informes.

El formato de la presentación en estos informes múltiples, es exactamente el mismo que los anteriores.

10 Conclusiones

Durante el desarrollo de la aplicación software de este proyecto, me he podido dar cuenta de que a la hora de construir software de calidad una de las cosas más importantes es la organización. Desde un primer momento se ha de tener claro que se quiere hacer y sobre cómo se quiere hacer. Una aplicación web que va a ser usada en un entorno real, está condicionada a ciertos factores que pueden determinar su éxito o su fracaso, por eso un desarrollo siguiendo una metodología ágil es clave.

Al usar varios Frameworks de tecnologías bien conocidas a la hora de realizar una aplicación web simplifica mucho el trabajo de programación, puesto que cuentan con el soporte de una gran comunidad detrás. Estos Frameworks son herramientas fundamentales para llevar a cabo un proyecto exitoso.

La gestión y la creación de la base de datos usando un SGBD ha sido una tarea sencilla, aun así, lo más laborioso de este proceso es poder identificar de forma correcta el modelo de datos que se tiene que usar. Una vez se tuvo claro esto, su creación fue muy sencilla, porque tan solo hubo que crear los correspondientes modelos y el propio SGBD se encargó de crear el código necesario para crear la base de datos.

Lo más complicado bajo mi punto de vista fueron las labores de formación del Framework VueJS, aunque el uso de este Framework es sencillo, la curva de aprendizaje de sus particularidades es algo elevada. Una vez aprendido a usarlo, surgieron varios problemas, pero como he comentado antes, este tipo de tecnologías cuenta con una gran comunidad detrás que sirve de apoyo a los desarrolladores.

Por otra parte, la elección de Vuetify, también fue un gran acierto. La creación de una interfaz, para cualquier sistema puede ser un reto para cualquier desarrollador, pero con este tipo de herramientas, hacer un diseño minimalista, que es lo que se pretendía desde un primer momento, y atendiendo a las necesidades que el usuario final tenía en mente, ha podido llevarse a cabo sin la necesidad de tener demasiados conocimientos en diseño web.

En mi opinión el proyecto, tras pasar por todas las fases de desarrollo y diversas iteraciones y tras mucho feedback por mi parte y la del Jefe de Estudios, se podría concluir que el desarrollo del sistema que ha sido todo un éxito.

Pero como en todo el software que se crea desde cero, la finalización de este es solo el comienzo de una nueva etapa de mantenimiento, de detección y corrección de posibles bugs y de futuras mejoras que se podrían llevar a cabo.

10.1 Relación proyecto-estudios cursados

Los estudios que se han cursado en el Grado, me han servido para poder enfocar la creación de este software de una manera más profesional, es decir, gracias a los conocimientos adquiridos en asignaturas como Ingeniería del Software, y las reuniones con el Jefe de Estudios, se ha podido detallar los requisitos del sistema de forma clara y concisa.

Por otra parta la decisión de usar una metodología ágil, fue gracias a la asignatura de Proceso software y proyecto de ingeniería de software, donde pudimos comprobar de primera mano a través de un pequeño software que se creó, todo el proceso que rodea a la creación de un proyecto software.

También han sido determinantes los conocimientos a la hora de crear y modelar el sistema de datos de la base de datos, de la asignatura Bases de datos, sin estos conocimientos, el modelado de la base de datos, no hubiera sido tan preciso y quizás hubiera tenido errores que a la larga hubieran lastrado el desarrollo.

En general el Grado de Ingeniería Informática, me ha dotado de los conocimientos necesarios para poder aprender y desenvolverme por mi cuenta, proporcionándome herramientas útiles y la capacidad de abstracción para la resolución de los problemas que han ido apareciendo en este proyecto.

10.2 Trabajo a futuro y líneas de mejora

En esta versión final del proyecto se han cumplido todos los requisitos planteados en un inicio, pero como la mayoría de software está destinado a ir evolucionando con el paso del tiempo.

Las posibles mejoras que se pueden realizar de cara el futuro son varias y tanto a nivel funcional como a nivel interno de la aplicación.

El software se ha desarrollado partiendo de la base de que solo se iba a usar en un ordenador, tanto de escritorio como portátil, por eso una futura mejora seria adaptar el diseño a un teléfono móvil o a una Tablet.

También se ha construido el software modularizando las vistas y ciertos elementos, pero aun así no ha sido suficiente y por motivos de viabilidad temporal, no se ha modularizado todo lo que me hubiera gustado, por eso una futura mejora, podría ser modularizar todos los formularios modales para tener un único punto de entrada a esos modales.

También, se podrían mejorar de cara al futuro ciertos aspectos de la interfaz, haciéndola aún más sencilla e intuitiva.

Por otro lado, el Jefe de Estudios, ya que es él, el usuario final de la aplicación, sería el encargado de pedir de mejoras funcionales de cara al futuro.

11 Referencias

[1] Yair, "Que es composer y cómo usarlo" (2019). [En línea]. Disponible: https://styde.net/que-es-composer-y-como-usarlo/

[2] Concepción Palacios Bernal y Antonio Calvo-Flores Segura, "Sobre la herramienta de gestión de TFG: Fase de depósito y defensa de trabajos" [En línea]. Disponible: https://www.um.es/documents/115466/3202656/Para+tutores.+Validar+TFG.pdf/46ba861e-2c02-455f-9a5b-9f7d8aff40d1

[3] Comisión Académica, "Normativa de trabajo de fin de grado", 2013, 2017, 2018, 2020. [En línea]. Disponible: https://www.inf.upv.es/www/etsinf/wp-content/uploads/2020/04/Normativa-interna-modificada-CAT-29-Abril-2020.pdf

[4] Comisión Académica, "procedimiento para la asignación de tribunales e instrucciones para la defensa", 2020. [En línea]. Disponible: https://www.inf.upv.es/www/etsinf/wp-content/uploads/2020/06/ProcedimTribunal-y-Defensa-CAT_04_06_2020.pdf

[5] Michael James, "The Scrum Reference Card, and Other", [En línea]. Disponible: http://scrumreferencecard.com/reference-card-de-scrum/

[6] ProyectosAgiles.org, "Desarrollo iterativo o incremental", [En línea]. Disponible: https://proyectosagiles.org/desarrollo-iterativo-incremental/

[7] IEEE 1074-1991, "IEEE 1074-1991 - IEEE Standard for Developing Software Life Cycle Processes", [En línea]. Disponible: https://standards.ieee.org/standard/1074-1991.html

[8] Krasner, Glenn E.; Stephen T. Pope, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System", [En linea]. Disponible: https://web.archive.org/web/20100921030808/http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf .

[9] Laravel, [En línea]. Disponible: https://laravel.com/docs/7.x

[10] Miguel Angel Alvarez, "Manual de Laravel 5". [En línea]. Disponible: https://desarrolloweb.com/articulos/laravel-eloquent.html

[11] Evan you, "VueJs", 2014-2020, [En línea]. Disponible: https://vuejs.org/

[12] John Leider, "Vuetify", 2016-2020, [En línea]. Disponible: https://vuetifyjs.com/en/getting-started/quick-start/

[13] Rasmus Lerdorf, "PHP", 1994, [En línea]. Disponible: https://www.php.net/docs.php

[14] Roy Fielding, “CHAPTER 5, Representational State Transfer (REST)”. [En línea]. Disponible: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[15] MySQLAB, “MySQL”, [En línea]. Disponible: https://www.mysql.com/

[16] [En línea]. Disponible: https://github.com/axios/axios

[17] Evolus Website, “Pencil Proyect”. [En línea]. Disponible: https://pencil.evolus.vn/Features.html

Apéndice A

Procedimiento para la designación de tribunales e instrucciones para la defensa.

La aplicación que se ha desarrollado para este TFG, cuenta con diversas funciones y tareas, podemos destacar la de gestionar profesores, gestionar titulaciones etc.… Pero, la tarea inicial con la que surgió este proyecto, es la de poder automatizar la conformación de los tribunales de TFGs y TFMs, para que los alumnos puedan defender sus trabajos. El profesor encargado de conformar estos tribunales, tiene que hacerlo atendiendo a una normativa regulada y aprobada por la comisión académica [3] [4].

A continuación, se hará mención de la normativa regulada y aprobada por la comisión académica para la designación de los tribunales:

- Número de tribunales en los que un profesor puede participar. Cada profesor tendrá asociado un parámetro para saber en cuantos tribunales ha participado, que en este caso la normativa de la ETSINF determina que el número máximo en los que un profesor puede participar en una convocatoria es de 3 tribunales. Este parámetro solo será actualizado si el profesor ha participado en el tribunal como titular, en ningún caso, se incrementará si el profesor ha sido suplente.
- Se tiene que garantizar que dos profesores sean de la misma rama de la titulación, esto se intentará, siempre que sea posible que en un tribunal existan al menos dos profesores con experiencia o interés en la rama del tribunal y otro con interés o experiencia en otra diferente.
- En cada tribunal de calificación actuará como presidente el PDI de mayor categoría docente y antigüedad y como secretario el de menor categoría docente y antigüedad.
- La elección de los cargos que ostentaran los profesores y de si serán titulares o suplentes, se realizara totalmente por sorteo.
- La información necesaria para conformar los tribunales y los datos que serán los más relevantes a la hora de conformar los tribunales serán:
    - Nombre y apellidos del profesor.
    - Categoría
    - Antigüedad
    - Departamento
    - Especialidades
- Los tribunales constaran de 6 profesores, 3 titulares que serán los profesores que finalmente evalúen al alumno, y 3 profesores suplentes. Tantos los titulares como los suplentes atenderán a los mismos requisitos y criterios para su elección.

Como hemos podido ver, estos datos son determinantes para la conformación de los tribunales, por lo que se necesita que haya una persistencia de dichos datos para que el usuario atendiendo a los criterios de búsqueda que considere oportuno, podrá ver los profesores que cumplan esos criterios y/o modificar esos datos si lo viera necesario, puesto que actualmente

si se realizara una búsqueda con los mismos criterios de búsqueda, sería imposible saber que profesores cumplen esos criterios porque se trabaja con los ficheros de origen.

En base a esta normativa, se ha desarrollado un algoritmo para la elección de forma totalmente aleatoria de los profesores que conformarán un tribunal. Los datos que se han cogido para realizar el algoritmo son:

- El nombre y el apellido de los profesores.
- La categoría y la antigüedad, estos dos datos, junto con el sexo del profesor, son la base del algoritmo. La elección de los cargos según la normativa se realiza en base a estos parámetros.
- Las especialidades en la que son expertos los profesores, para poder garantizar así que al menos dos de los profesores de una misma rama, estén en el tribunal. Si esto no es posible, se garantizará que al menos uno si lo sea.
- El número de tribunales en los que un profesor puede participar. Este parámetro, indica que profesores a la hora de generar el tribunal en la aplicación, puede o no puede estar en la lista de profesores. Si es menor que 3, el profesor podrá ser elegido para participar en un tribunal y si es mayor o igual que 3 no.