UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Herramienta para la gestión académica de una escuela de música - Calificaciones

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Uriel Cárcel, Nicolás

Tutor/a: Villanueva García, Alicia

CURSO ACADÉMICO: 2023/2024

Resum

El desenvolupament d'este projecte té com a objectiu proporcionar a l'escola de música "La primitiva" un prototip de ferramenta per a la gestió de les notes dels seus alumnes, perquè posteriorment prenguen una decisió informada sobre la implementació definitiva. El desenvolupament de l'aplicació s'ha dut a terme seguint una metodologia SCRUM que ha permés portar una gestió dels avanços i dels canvis satisfactòriament. L'aplicació permetrà als professors i els tutors introduir les notes dels estudiants i consultar les seues dades personals d'una forma eficaç. Per a donar-li un sentit complet al projecte, és important estructurar i tractar les dades dels alumnes correctament, per la qual cosa s'ha desenvolupat un *backend* en Python i Flask amb el que mantindre la comunicació entre les accions dels usuaris en el *frontend* i la base de dades. D'esta manera, s'ha aconseguit desenvolupar una aplicació que complix amb les expectatives i requisits dels directors de "La primitiva", proporcionant-los l'oportunitat de prendre la decisió amb coneixement sobre la implementació del software a l'escola

Paraules clau: Desenvolupament web, React, Python, Flask

Resumen

El desarrollo de este proyecto tiene como objetivo proporcionar a la escuela de música "La primitiva" un prototipo de herramienta para la gestión de las calificaciones de sus alumnos, para que posteriormente tomen una decisión informada sobre la implementación definitiva. El desarrollo de la aplicación se ha llevado a cabo siguiendo una metodología SCRUM que ha permitido llevar una gestión de los avances y de los cambios satisfactoriamente. La aplicación permitirá a los profesores y los tutores introducir las notas de los estudiantes y consultar sus datos personales de una forma eficaz. Para darle un sentido completo al proyecto, es importante estructurar y tratar los datos de los alumnos correctamente, por lo que se ha desarrollado un *backend* en Python y Flask con el que mantener la comunicación entre las acciones de los usuarios en el *frontend* y la base de datos. De esta forma, se ha conseguido desarrollar una aplicación que cumple con las expectativas y requisitos de los directores de "La primitiva", proporcionándoles la oportunidad de tomar la decisión con conocimiento sobre la implementación del software en la escuela.

Palabras clave: Desarrollo web, React, Python, Flask

Abstract

The development of this project aims to provide the music school "La primitiva" with a prototype tool for the management of their students' grades, so that they can later make an informed decision about the final implementation. The development of the application has been carried out following a SCRUM methodology that has allowed a satisfactory progress and change management. The application will allow teachers and tutors to enter students' grades and consult their personal data in an efficient way. To give a complete sense to the project, it's important to structure and treat the students' data correctly, so a backend has been developed in Python and Flask to maintain the communication between the users' actions in the frontend and the database. In this way, it has been possible to develop an application that meets the expectations and requirements of the directors of "La primitiva", providing them with the opportunity to make an informed decision about the implementation of the software in the school.

Key words: Web development, React, Python, Flask

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

En Rafelbunyol la música forma parte de la vida cotidiana, y la creación de una web acorde a las necesidades de la escuela de música "La Primitiva" y una correcta gestión de los datos, es un desafío personal que afrontar para que esta cultura musical valenciana siga prosperando y salga fortalecida en la era digital. Actualmente, la escuela realiza todas sus gestiones de forma manual, sin contar con una web. Hacen uso de Excel para gestionar las matrículas, además de otros métodos tradicionales para calificar a los alumnos, que aunque efectivos, limitan la eficiencia y accesibilidad de la información. Por ello, más allá de la meta de proporcionar a la escuela un prototipo, mi objetivo es ofrecer a la escuela el conocimiento que necesitan para determinar la viabilidad del software, que se centrará en la parte donde los profesores y tutores introducirán las calificaciones y gestionarán información de los alumnos. A partir de esto, se determinará si realmente necesitan implementar la aplicación en su escuela, basándose en toda la información que les proporcionaré a lo largo del desarrollo del proyecto.

1.1 Motivación

La motivación para llevar a cabo este proyecto surge cuando me plantean el problema actual de una escuela de música en Rafelbunyol y su actual labor de gestión de los datos de los alumnos. La Comunidad Valenciana tiene una gran tradición musical y, a pesar de no haber tenido experiencia directa en escuelas de música, quería aportar lo que pudiera para preservar esa cultura musical valenciana. En este caso, mis conocimientos informáticos. El proyecto se me propone una vez termino mis prácticas de empresa como desarrollador *frontend*, por lo que me servía para demostrar todo lo aprendido e incluso seguir mejorando mis habilidades en desarrollo web. Por otra parte, me planteaba el desafío de aprender una nueva tecnología que nunca había utilizado, como es Python junto con el framework Flask, para llevar a cabo la comunicación entre el *backend* y el *frontend*.

1.2 Objetivos

El objetivo principal de este trabajo es realizar un prototipo de aplicación web que cubra las necesidades particulares de una escuela de música. En este caso se quiere enfocar el desarrollo en la parte de la gestión de las notas de los alumnos, con todo lo que incluye: introducción, visualización y modificación de calificaciones, roles de los usuarios, permisos y restricciones. Tras conseguir esto, la intención de la escuela es tener el conocimiento necesario de qué les puede ofrecer este software y decidir si quieren continuar con él, haciendo que el desarrollo de la aplicación se convierta en un desarrollo profesional.

A nivel personal, aparece el objetivo de demostrar los conocimientos adquiridos durante toda la carrera, y en especial, durante las prácticas de empresa, para llevar a cabo un proyecto de forma individual, teniendo que aprender nuevas tecnologías y con la presión de cumplir con las expectativas de unos clientes.

CAPÍTULO 2
Metodología

SCRUM [SB01] se caracteriza por su enfoque iterativo e incremental, lo que permite adaptar el proyecto a medida que evolucionan las necesidades del cliente. Se tiene una comunicación fluida entre el desarrollador y el *Product owner*, lo que favorece la anticipación ante adversidades y permite tomar las decisiones correctas a tiempo, en caso de que las cosas no vayan según lo esperado.

Desde el inicio del proyecto se adoptó la metodología ágil SCRUM. Tanto mi tutora del TFG, Alicia Villanueva, como los directores de la escuela de música **Escola de música La Primitiva de Rafelbunyol** hicieron de *Product owners* o clientes del proyecto. Mi experiencia con esta metodología fue adquirida tras usarla durante los dos cuatrimestres de la rama de Ingeniería del Software en las asignaturas de Proceso de software y Proyecto de ingeniería. Además de la experiencia durante el periodo académico, también he tenido la oportunidad de observar cómo funciona la metodología dentro de un ambiente profesional, por lo que considero que tengo bastante experiencia con ella y soy capaz de llevar un proyecto a cabo cumpliendo con los pasos que plantea esta metodología. No obstante, para la realización del proyecto, no hemos llevado a cabo dos de las fases que plantea [Del24]. Estas han sido la reunión diaria o *Daily Scrum* y el *Refinement* ya que no hemos considerado óptimo llevarlas a cabo. Por otra parte, al inicio y final de cada sprint, se realizaba el *Sprint Planning*, informando a los clientes sobre el trabajo a realizar en el próximo sprint. Además, al final de cada sprint, se llevaban a cabo el *Sprint Review* y el *Sprint Retrospective* con los clientes. Estas reuniones, aproximadamente cada dos semanas, permitían mostrar avances, consultar dudas y validar las tareas futuras, de forma que siempre estábamos las dos partes (cliente y desarrollador) en contacto, pudiendo anticiparnos a posibles cambios o errores y solventarlos a tiempo.

En el capítulo 4, Diseño de la solución, explicaré en detalle, no solo el diseño de la solución, sino también cómo, gracias a la metodología ágil SCRUM tuve la capacidad de llevar a cabo un buen diseño de la aplicación y de su funcionamiento, antes de comenzar con el desarrollo.

Seguir manteniendo contacto con SCRUM y en general con las metodologías ágiles me parecía importante de cara al mi futuro laboral ya que actualmente las empresas las han adoptado adoptando como metodologías principales a la hora de plantear proyectos [Per24].

CAPÍTULO 3
Diseño de la solución

Este apartado del trabajo se centrará en explicar cómo ha sido el proceso de diseñar la solución que más adelante se desarrollará en forma de aplicación web. Se verá como se empieza con un breve planteamiento de la necesidad, que incluye una descripción del funcionamiento actual de la escuela y una serie de requerimientos por parte de los clientes y acaba con una validación por parte de los directores de la escuela sobre las interfaces gráficas y funcionalidades planteadas. Este apartado ofrece un claro ejemplo de la metodología SCRUM empleada durante todo el proyecto y cómo la constante validación por parte de los clientes guía el proceso de diseño en la dirección correcta.

3.1 Recolección de información

Al inicio del proyecto se partía de una serie de documentos elaborados por parte de los clientes que son clave para entender las necesidades y tener una visión global de lo que sería el resultado final de la aplicación. El primer documento relevante, proporcionaba una visión general sobre el funcionamiento actual de la escuela. En este se encuentra información muy importante como los diferentes roles y una breve descripción de su papel en la escuela. Entre ellos se identifican como relevantes para llevar a cabo la solución, los **roles** de: Administrador, Dirección, Tutor y Profesor.

También aparece información actual sobre cómo abordan el hecho de calificar a los alumnos al final de cada evaluación, los diferentes permisos de los diferentes roles a la hora de evaluar y hacer uso de las funciones de la aplicación. Aparece una breve descripción sobre cómo se dividen los cursos académicos, los nombres de las asignaturas que hay y su tipo (colectivas o individuales). Por último en este documento, se proporcionan todas las tareas que realizan durante la evaluación la dirección de la escuela, los tutores y los profesores.

Por parte de la dirección, encontramos la configuración de las evaluaciones, los informes (estadísticas de los alumnos y sus calificaciones), las asignaturas colectivas, las asignaturas individuales y configurar las clases para ambos tipos de asignaturas (número de horas y asignar a un profesor).

Los tutores pueden, para cada alumno tutorizado: firmar el boletín de notas, firmar el informe del alumno e introducir comentarios en el boletín de notas. Dentro de los profesores, se diferencian entre profesores de asignatura colectiva y los de asignatura individual. Tanto los de asignatura colectiva como individual pueden acceder al listado de sus alumnos y calificar y poner comentarios asociados a su asignatura en el boletín de notas. La única diferencia se encuentra en que los profesores de asignaturas individuales, son tutores de los alumnos de su asignatura de instrumento. Además estos pueden

asignar cada alumno a una franja horaria. Se aclara también, que cualquier rol superior es capaz de realizar una función de un rol inferior siguiendo la jerarquía (Dirección, Tutor y Profesor).

Es importante señalar que esta diferenciación de roles dentro de la jerarquía de la escuela tendrá su importancia a la hora del diseño de la aplicación, ya que estos usuarios realizarán diferentes funcionalidades dentro de la web.

El segundo documento es un Excel con la información que recoge la escuela cuando un alumno realiza la matrícula para el siguiente año académico. Este documento incluye una variedad de información, desde datos personales del alumno y sus padres o tutores legales hasta datos relativos al año académico que va a cursar, como por ejemplo, los instrumentos que toca.

Por último, hay un tercer documento donde se recoge la información obtenida tras la primera reunión de mi tutora con los directores de la escuela de música. Este documento recoge la lista de requisitos funcionales y no funcionales que debe tener la aplicación final, los cuales se detallan a continuación.

Encontramos en primer lugar los siguientes requisitos funcionales:

- El rol de profesor será único sin distinguir entre tutor o no, pero las acciones dependerán de si actúa como tutor o no.
- En la interfaz donde los profesores ponen las calificaciones, para cada alumno se debe poder ver y editar tanto las notas como las observaciones.
- El rol de dirección debe poder abrir y cerrar períodos de evaluación. Una vez cerrado, ningún profesor podrá modificar las notas, únicamente dirección.
- Cualquier consulta de calificaciones será de lectura. También habrá una opción de escritura, para evitar modificar calificaciones por error.
- Dirección deberá poder asignar los alumnos y los profesores a los grupos.
- Tiene que haber una funcionalidad para cambiar entre los idiomas disponibles.
- Exportar datos Excel a la base de datos.
- Generación de PDF de las calificaciones de los alumnos.
- Se necesita un sistema de avisos o notificaciones en la aplicación.
- Listado con las estadísticas de aprobados/suspendidos por curso de cada asignatura.
- De cada alumno se deberá calcular la nota media del curso, que se calculará con las notas de la tercera evaluación.

Como requisitos no funcionales, se encuentran los básicos para el correcto funcionamiento de una aplicación web:

- El tiempo de respuesta: la aplicación debe responder a las solicitudes de los usuarios en un tiempo inferior a 25ms.

- La aplicación debe ser capaz de soportar múltiples usuarios simultáneamente sin perder rendimiento.
- Debe ser escalable para soportar un número creciente de alumnos.
- La interfaz de usuario debe ser intuitiva y fácil de usar, ya que los usuarios no tienen por qué ser expertos.
- Debe ser compatible con cualquier navegador.
- La aplicación tiene que estar disponible en castellano y en valenciano.

3.2 Diseño inicial

En esta fase del proyecto, una vez asimilada toda la información anteriormente nombrada, lo que se propone es una serie de interfaces y unas decisiones a nivel de funcionalidades y requisitos que deben aparecer en la aplicación, estableciendo así el alcance del proyecto. La decisión más importante a nivel de requisitos y funcionalidad es condensar los roles, es decir, agrupar las funcionalidades de todos los diferentes roles en dos: tutores y profesores. Para este proyecto el cual su objetivo era proporcionar a la escuela un prototipo de aplicación web, no se necesita la aparición de más roles ni tener una jerarquía tan marcada. En este caso, en el que lo importante es la evaluación de los estudiantes y la obtención de sus datos personales, los tutores serán los que tengan todos los permisos a nivel de requisitos, e incluso asumirán funcionalidades que anteriormente solo eran realizadas por la directiva o en un futuro por los administradores de la web, como por ejemplo, cerrar actas o modificar el tiempo de finalización de las evaluaciones. Por otro lado, los profesores tendrán alguna funcionalidad menos a la hora de consultar y modificar datos, con el fin de respetar las decisiones jerárquicas de la escuela.

A nivel de interfaces, se plantean tres interfaces principales hechas a mano y en papel con el objetivo de validar que estaba tomando el camino correcto y no se estaba obviando ninguna funcionalidad. Estas tres interfaces son **Poner notas**, **Listar estudiantes** y **Buscar estudiantes** además de un menú lateral con el que se podrá navegar por las interfaces y un menú en la parte superior donde se podrá cambiar de usuario, cambiar de idioma y ver las notificaciones.

Una vez validada la propuesta inicial por parte de mi tutora, el siguiente paso es crear las interfaces de forma digital con la ayuda de Figma [Fig24].

3.3 Diseño detallado

En este apartado se presenta una visión general de las interfaces elaboradas con Figma en los que se reflejan parte de los requisitos planteados anteriormente por los clientes. Además, se presentará el diseño UML [UML24] del aspecto que tendrán las tablas de la base de datos.

La Figura 3.1 representa el prototipo de **Poner notas**. En esta interfaz los profesores serán capaces de visualizar y modificar las notas de los alumnos filtrados según los cuatro *dropdowns*. En el filtro de Asignatura, cada profesor podrá ver únicamente a sus alumnos, el curso va desde Primero EE hasta Cuarto EE, los grupos son A, B y C, aunque si se selecciona una asignatura de instrumento, ésta no tiene grupos, por lo que el *dropdown* saldrá desactivado. Por último, la evaluación variará entre primera, segunda y tercera evaluación. Como se aprecia en la esquina inferior derecha, aparece el botón de “Cerrar

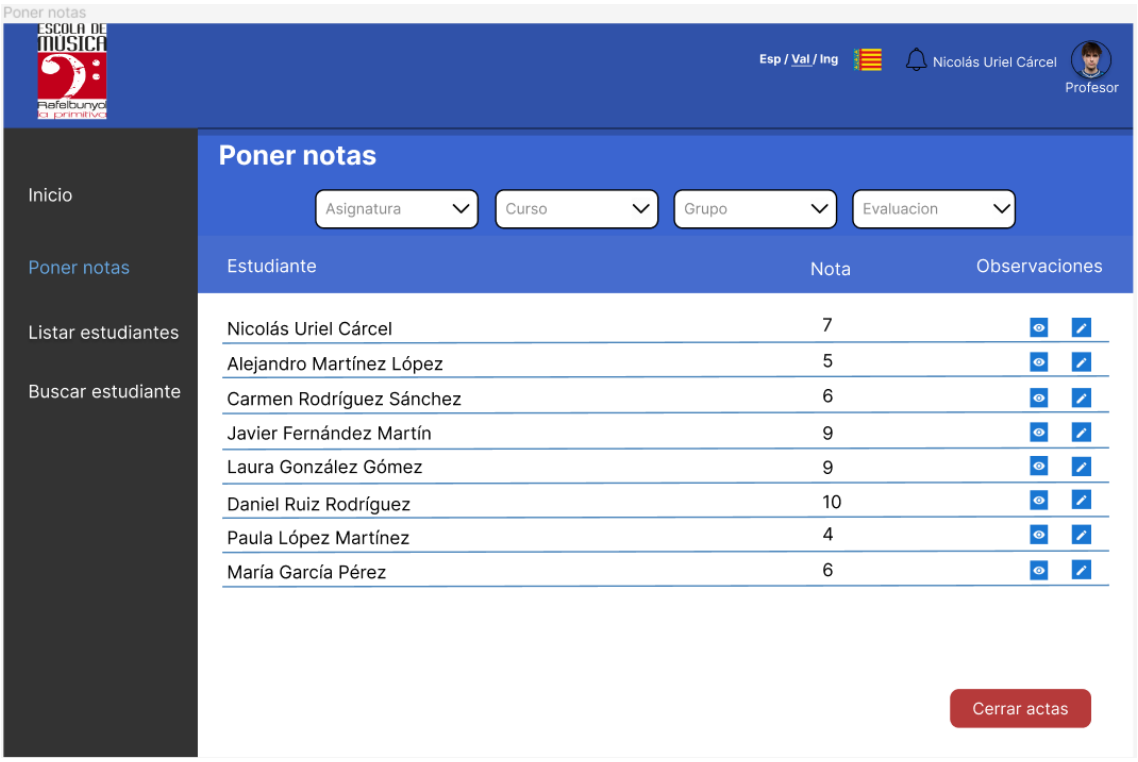


Figura 3.1: Prototipo de *Poner notas*

actas" que sirve a los tutores para confirmar que las notas del alumno son correctas y ya no se van a modificar.

En este primer prototipo se puede ver por primera vez el menú lateral de navegación, con el que el profesor podrá cambiar de interfaz libremente, pues este menú es fijo y siempre aparecerá. De la misma forma, también aparece el menú superior, donde aparecen las funcionalidades de cambiar idioma, el icono de notificaciones y el rol del usuario conectado en la aplicación. Además clicando el logo en la esquina superior izquierda desde cualquier interfaz, nos llevará al "inicio" de la aplicación (interfaz simbólica que representa el *home* de cualquier aplicación).

La Figura 3.2 corresponde con la segunda interfaz **Listar estudiantes** con la que el profesor puede, mediante tres filtros, listar a sus alumnos para obtener datos personales de estos. Estos tres filtros son iguales a los de la interfaz anterior: **Poner notas**.

La Figura 3.3 plasma el diseño presentado a la escuela mediante el cual los profesores podrán, mediante tres filtros, buscar a un alumno y ver todas sus notas. Estos filtros serán un buscador en el que se introducirá el nombre del alumno a consultar y el año académico del que se quiere consultar las notas. Además también se tendrá la posibilidad de modificar tanto las notas como las observaciones escritas previamente por los profesores.

Por último, como se aprecia en la Figura 3.4, para llevar a cabo una visualización y edición de las calificaciones más individual, se plantean dos menús modales con los cuales se podrá visualizar de forma completa la nota y la observación del alumno para cada evaluación. De la misma forma, habrá un modal encargado de poder modificar estos datos para todas las evaluaciones. Esta funcionalidad aparecerá tanto en la interfaz de **Poner notas** como en **Buscar estudiantes**

El aspecto de las tablas de la base de datos debería ser como el de la Figura 3.5, pero más adelante en el apartado de Desarrollo de la solución se verá como se toma una

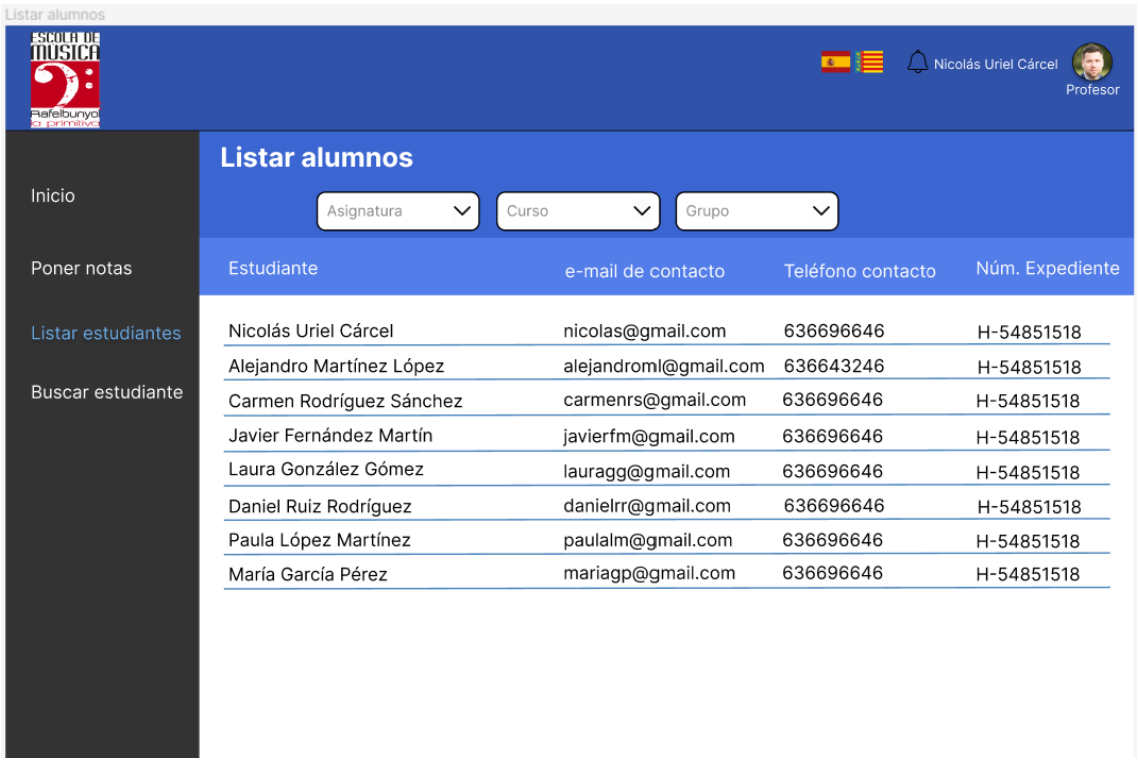


Figura 3.2: Prototipo de *Listar estudiantes*

Figura 3.3: Prototipo de *Buscar estudiantes*

decisión de simplificar la estructura de datos con el objetivo de no añadir complejidad innecesaria al desarrollo del prototipo.

En este primer diseño de las tablas de la base de datos, aparecen cuatro tablas:

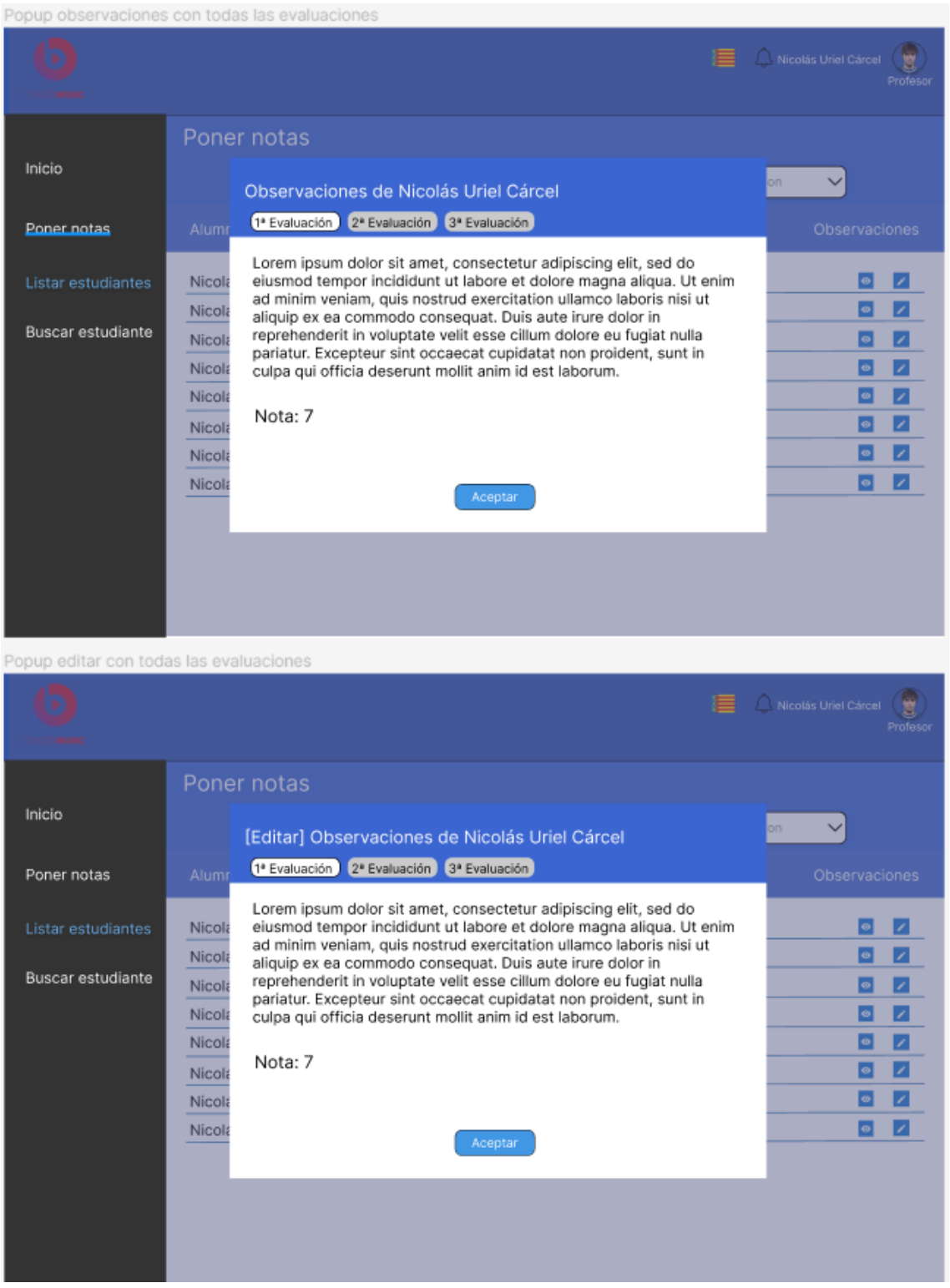


Figura 3.4: Prototipo de los menús modales

- Tabla Estudiante: esta tabla representa a cada alumno y está compuesta de un identificador, su nombre, su instrumento principal, su profesor, su número de matrícula y su número de teléfono. Tiene una relación de cero a muchos (0:*) con la tabla Nota y de uno a muchos (1:*) con la tabla Profesor.

- Tabla Asignatura: representa cada una de las asignaturas que tiene la escuela. Se compone de un identificador y del nombre de la asignatura. Tiene una relación de cero a muchos con Nota (0:*) y de uno a muchos con Profesor (1:*).
- Tabla Nota: compuesta por el identificador de alumno, el identificador de asignatura, el curso, la evaluación, la nota y las observaciones. Sirve para crear la relación entre cada estudiante y cada asignatura, asignándole para cada asignatura, curso, grupo y evaluación una nota y observación diferente. Tiene una relación de uno a uno con Estudiante (1:1) y de uno a uno con Asignatura (1:1).
- Tabla Profesor: Por último, la tabla Profesor representa a los usuarios de la aplicación. Está compuesta por un identificador propio, un identificador de la asignatura que imparten, su nombre, un booleano que comprueba si es tutor (será *true* en caso de que su asignatura sea una de las de instrumento) y si es administrador, es decir, cuenta con todos los permisos de la aplicación. Tiene una relación de uno a mucho con Estudiante (1:*) y de uno a muchos con Asignatura (1:*).

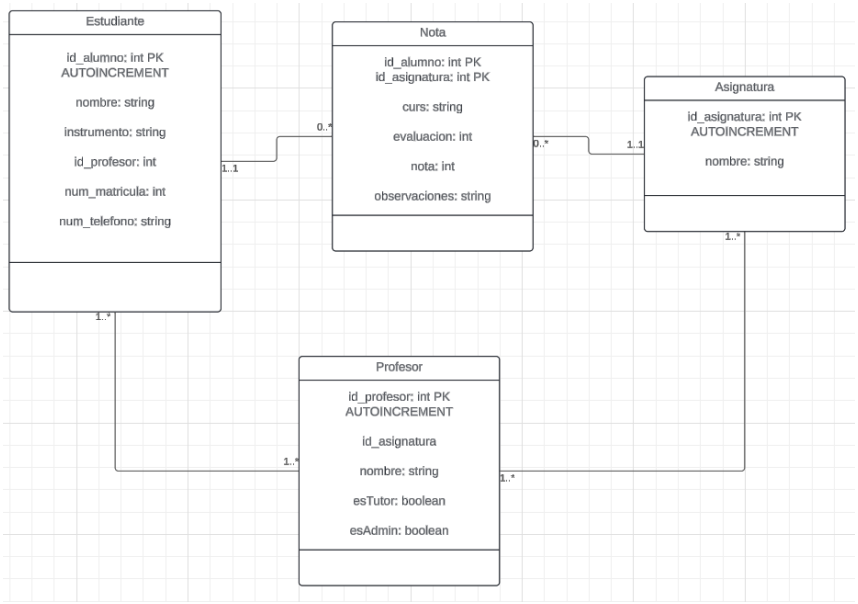


Figura 3.5: Diseño inicial de las tablas de la base de datos

Tras validar las interfaces y el diseño de la base de datos con mi tutora, se tuvo la reunión que se describe en la siguiente sección, en la que se juntó los directores de la escuela de música para presentarles los diseños y tener su aprobación antes de comenzar con la implementación.

3.4 Validación con los directores de la escuela

La primera reunión con el equipo directivo de la escuela consistió en conocer a los directores, comentarme sus deseos y lo que pretendían conseguir con mi trabajo, así como validar mis tareas realizadas hasta el momento.

Planteé la reunión como un *Sprint Review*, en la que, en forma de demo, presentaba el proyecto haciendo un recorrido por las interfaces creadas en Figma descritas en este capítulo y planteaba las diferentes funcionalidades no visibles en los diseños que quería añadir. Durante la presentación, los directores tuvieron la oportunidad de hacer preguntas, comentarios o sugerencias sobre los diseños, las cuales iba implementando al momento en las interfaces, para ver qué les parecía mejor. Entre los cambios que se realizaron sobre las interfaces fueron:

- Los idiomas de la aplicación serán el castellano y el valenciano, por lo que se elimina el inglés del selector de idioma.
- La funcionalidad de "Cerrar actas" se moverá de la interfaz **Poner notas** a la interfaz **Buscar estudiantes**.
- Se añadirá la funcionalidad de seleccionar el fin de la evaluación en la interfaz Buscar estudiantes, donde los administradores establecerán el plazo de cada evaluación.

Este intercambio de ideas fue importante para empezar el desarrollo lo más alineado posible con las expectativas de la escuela.

Al final de la reunión se acordó que Alicia Villanueva, debido al conocimiento de la situación y de las necesidades de la escuela, desempeñaría también el rol de cliente del producto para poder solucionarme las dudas necesarias al final de cada sprint y no necesitar la disponibilidad de los directores de la escuela cada dos semanas.

Tras esta reunión, daba por finalizada la etapa de diseño de la solución y comenzaba a plantear cómo iba a llevar a cabo el desarrollo de la aplicación.

CAPÍTULO 4
Tecnologías

Para el desarrollo del proyecto se ha decidido utilizar una serie de tecnologías, lenguajes y frameworks específicos con el objetivo de alcanzar la funcionalidad esperada, eficiencia y mantenibilidad. Estas tecnologías son: React [Rea24] para crear las interfaces de la web, Typescript [Typ24] y CSS [CSS24] como lenguajes para la parte *frontend* y Python [Pyt24b] junto con Flask [Fla24] para la parte del *backend* del proyecto. Para la base de datos se ha usado DB Browser [Bro24] y para los bocetos de las interfaces que fueron validados por los clientes y usados como referencia para diseñar la web, se usó Figma [Fig24].

En los siguientes apartados se describirán estas herramientas así como se presentará una alternativa de tecnologías también válidas para llevar a cabo el proyecto. Este análisis servirá para entender las razones por las que se han elegido estas tecnologías, dando así un sentido a las decisiones tomadas para llevar a cabo el proyecto.

4.1 React

El primer motivo para justificar el uso de React es su eficiencia y rendimiento, aspectos muy importantes a la hora de crear aplicaciones web que garanticen una experiencia de usuario fluida y satisfactoria. Gracias a su arquitectura basada en componentes, se puede reutilizar código, lo que simplifica el proceso de desarrollo y mantenimiento a largo plazo. Cada componente funciona de manera independiente, lo que mejora la capacidad de pruebas en el código. Además, debido a la experiencia adquirida en mis prácticas de empresa, he podido comenzar el proyecto con cierta base de conocimiento que me ha facilitado el desarrollo de la aplicación, al menos en la parte del *frontend*.

Comparándolo con Angular [Ang24], uno de los frameworks web más usados en el mercado, React tiene una curva de aprendizaje más suave [Rad24], lo que quiere decir se tarda menos en familiarizarse con los conceptos básicos de la tecnología. Además React se integra fácilmente con otras librerías y herramientas, lo que facilita el desarrollo de la aplicación.

En conclusión, la elección de React para este proyecto se basa en su eficiencia y simplicidad y la experiencia que tengo con la tecnología. Estas características me han garantizado llevar a cabo un desarrollo de calidad y mantenible a largo plazo. Es por ello que React ha sido una tecnología perfecta para cumplir con los objetivos esperados del proyecto.

4.2 Typescript y CSS

Como lenguajes de programación principales para llevar a cabo el desarrollo del *frontend*, se ha elegido Typescript y CSS para aplicar los estilos. La elección Typescript frente a Javascript [Jav24] se fundamenta en los siguientes motivos. Typescript es un lenguaje construido sobre Javascript, es decir es un superconjunto de Javascript que ofrece características adicionales. Ofrece un sistema de tipos estáticos que permite detectar errores en tiempo de compilación, lo que facilita la escritura y mejora la calidad del código. El hecho de que se base en Javascript hace que todo lo aprendido en la carrera lo pueda usar sin problemas, además de poder aportar también mi conocimiento adquirido durante las prácticas de empresa como desarrollador *frontend*.

Además, se está observando que en los últimos años el interés en España por Typescript está aumentando considerablemente [Cam24], por lo que formarme en esta tecnología era interesante a nivel personal.

4.3 Python y Flask

La elección de Python y Flask para el desarrollo del *backend* y de la API para la comunicación entre el *frontend* y la base de datos se basa en ciertas consideraciones. La primera es la breve experiencia que tuve junto a mi equipo de la asignatura de Proyecto de Ingeniería (PIN) al usarlo en la aplicación que desarrollamos. A pesar de haber trabajado con este lenguaje en el pasado, considero que mi nivel con este lenguaje es bastante bajo, por lo que uno de los objetivos era aprender a trabajar con él y ver todas las opciones que ofrece ya que es uno de los lenguajes más populares de la industria.

En cuanto a la elección Flask como framework web para la construcción de la API, me he basado en su simplicidad, lo que me venía perfecto para hacer un proyecto pequeño. Además, también fue el framework que usamos en la aplicación de PIN anteriormente nombrada, por lo que tenía fuentes en las que encontrar ayuda en caso de que lo necesitara.

Comparando Flask con Django [Dja24], otro framework muy popular de Python para desarrollo web, la decisión era clara y me baso en lo nombrado anteriormente: su simplicidad y conocimiento previo. Recomendaría el uso de Django en caso de querer llevar a cabo proyectos más grandes en los que se necesite más escalabilidad y seguridad, ya que por ejemplo, tiene características de seguridad incorporadas contra ataques CSRF, inyección SQL y XSS [Kin24].

4.4 DB Browser

Para la gestión de los datos de los alumnos de la escuela se ha optado por usar DB Browser [Bro24] por dos motivos principales. El primero es mi experiencia previa haciendo uso de esta herramienta en un proyecto personal que lleve a cabo hace tiempo, por lo que ya estaba familiarizado con la interfaz y sus funcionalidades. La otra razón para elegir este software es su simplicidad y eficiencia a la hora de tratar los datos. La herramienta proporciona una interfaz en la cual se puede crear, consultar y modificar las bases de datos con facilidad por lo que se ajusta a la perfección a lo necesario para el proyecto.

4.5 Figma

Para llevar a cabo el proyecto, lo primero que necesitaba era la validación de los clientes en las interfaces de la web. Para ello hice uso de Figma [Fig24], un software online que permite la creación de interfaces de forma muy sencilla. Esta aplicación te permite además, ponerle todos los estilos que necesites, tanto de tamaño como de color y luego exportarlo a CSS, por lo que era una opción perfecta. Además de estos motivos, la colaboración en tiempo real, es decir, que hayan varias personas observando (o editando) el mismo proyecto me ayudó a enseñar la propuesta a los clientes, ver sus ideas de cambios y poder modificar al momento las interfaces.

CAPÍTULO 5
Arquitectura

Este proyecto sigue una arquitectura cliente-servidor donde el *frontend*, desarrollado con React y Typescript, se comunica con el *backend* realizado en Python y Flask. La base de datos se maneja a través de DB Browser, que proporciona una solución sencilla para almacenar y recuperar los datos. La comunicación mediante estos componentes se realiza mediante llamadas a las API, creando una separación clara de responsabilidades entre los diferentes componentes, permitiendo así, la escalabilidad, un mantenimiento más sencillo, y la capacidad de realizar pruebas de forma independiente a cada módulo.

5.1 *Frontend*

La interfaz de usuario está construida con React y Typescript donde la estructura de carpetas de cada componente se divide en *containers* y en *components*.

- Container: contiene la lógica de negocio y la gestión del estado. Se encarga de hacer la llamada a la API y pasar los datos a los componentes de vista.

- Component: Definen la estructura visual de la aplicación. Reciben datos y funciones como props desde los *containers*.

Para hacer la comunicación con el *backend* he optado por usar "fetch" en vez de "Axios" por las necesidades de mi proyecto. Con fetch se aseguraba un correcto funcionamiento sin la necesidad de añadir dependencias adicionales. Además, las solicitudes que se necesitaban realizar no tenían una gran complejidad y "Axios" está más recomendado para aplicaciones más complejas.

5.2 *Backend*

El *backend* está desarrollado en Python junto al framework de desarrollo web Flask. Sus funcionalidades principales son las de definir los endpoints de la API, procesar las solicitudes HTTP que se reciben del *frontend* y realizar las operaciones CRUD (create, read, update, delete) en la base de datos. Se usa la librería sqlite3 para interactuar con la base de datos.

5.3 Base de datos

La base de datos se gestiona con DB Browser con SQLite, que proporciona una interfaz gráfica sencilla para la creación y gestión de la base de datos. Su funcionalidad es la de almacenar los datos de las asignaturas, los estudiantes y sus notas. El poblado de datos se ha realizado desde scripts escritos en Python.

5.4 Flujo de datos

El flujo de datos mediante esta arquitectura se puede resumir de la siguiente manera. El usuario interactúa con la interfaz mediante la aplicación web, realizando acciones como filtrar estudiantes para obtener sus datos personales o modificando las calificaciones de los mismos. Estas interacciones hacen que se envíen solicitudes HTTP al *backend* Flask a través de la API. Una vez recibidas, Flask las procesa y realiza las consultas SQL necesarias a la base de datos, obteniendo así una respuesta. A continuación, Flask responde las solicitudes HTTP con los datos obtenidos en formato JSON. Por último, el *frontend* recibe estos datos solicitados y se muestran en la web dinámicamente y de forma ordenada.

CAPÍTULO 6
Desarrollo de la solución

En este capítulo de la memoria se explicará todo el proceso del desarrollo de la aplicación y las decisiones tomadas durante este proceso. El orden que se ha seguido para llevar a cabo la aplicación ha sido: plantear primero todo el *frontend*, crear y poblar la base de datos, crear la API con los métodos necesarios de extracción de datos de la base de datos y por último, integrar la API en el *frontend* para hacer las llamadas a la base de datos desde la propia web. Por tanto, se va a seguir un orden similar a la hora de explicar todo el desarrollo, empezando primero por la estructura del proyecto dentro de Visual Studio Code.

6.1 Estructura del proyecto

Lo que representa la Figura 6.1 es la estructura de carpetas del proyecto dentro de *VSCode* [Cod24]. La carpeta *api* contiene todo lo relativo al *backend*, que se explicará más adelante. La carpeta *assets* está compuesta por todas las imágenes que contiene la aplicación. El hecho de agruparlas todas en una misma carpeta nos ayuda a mantener un buen orden dentro del proyecto y poder tener un mayor control a la hora de añadir o eliminar imágenes. La carpeta *idioms* contiene todo lo necesario para llevar a cabo la traducción de la aplicación entre castellano y valenciano. El resto de carpetas representan los componentes de la aplicación. Como se puede observar, se ha adoptado una estructura de archivos en la que se distingue, dentro de cada componente, una carpeta llamada *Components* y otra llamada *Containers*. En esta metodología, los *containers* se centran en la lógica del componente, por lo que dentro de ellos está la lógica de negocio, las llamadas a la API y la gestión de estados. Mientras que los *components* reciben mediante *props* los datos del *container* y se centran la parte visual y de interacción del usuario. Dentro de esta carpeta se encuentran dos tipos de archivos, el archivo .tsx que contendrá el código html y el archivo .css que contendrá los estilos que use el componente.

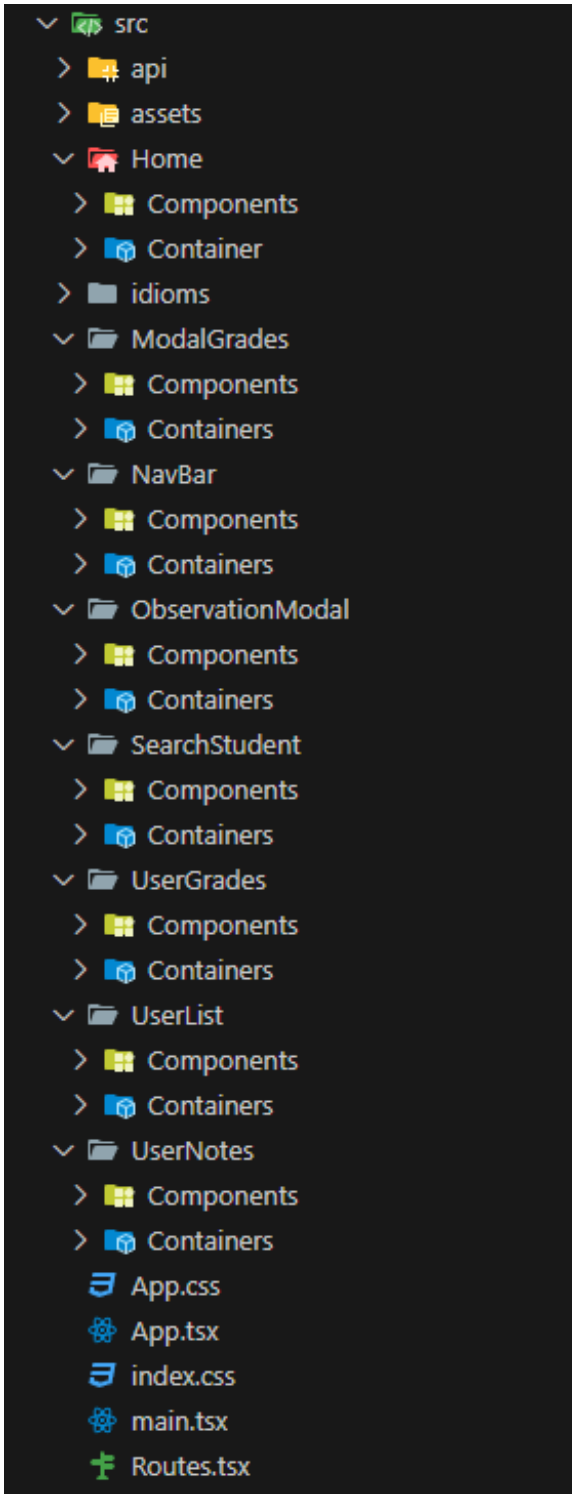


Figura 6.1: Estructura de carpetas de VSCode

Esta separación entre la lógica y la vista es una estructura muy usada en React, ya que permite que los componentes sean reutilizables e independientes de la lógica, por lo que facilita el mantenimiento y la escalabilidad del proyecto en caso de que sea necesario. [Pat24]

6.2 Desarrollo del *frontend*

6.2.1. Menú lateral y menú superior

En la aplicación se distinguen dos menús fijos (siempre están presentes) que son cruciales para tener una experiencia de uso satisfactoria, ya que estos proporcionan la navegación y las funcionalidades adicionales del usuario.

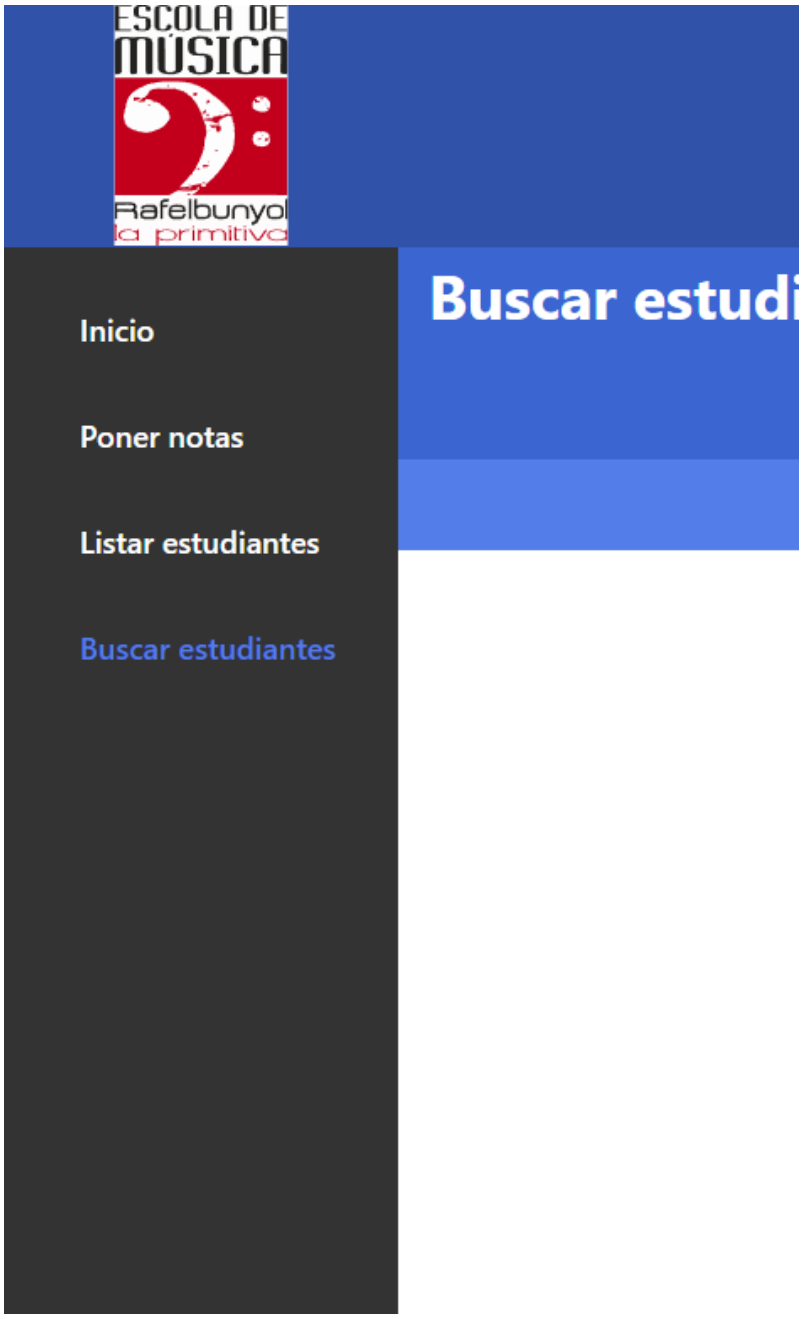


Figura 6.2: Menú lateral de navegación

En la Figura 6.2 se puede ver el menú lateral de navegación, con él se puede navegar por las diferentes funcionalidades principales de la aplicación. Se observa que tiene un diseño sencillo e intuitivo, donde se resalta color del título de la interfaz sobre la que se está actualmente.

```
<li>
  <NavLink
    to="/inicio"
    style={({ isActive }) =>
      isActive ? activeNavLinkStyle : navLinkStyle
    }
  >
    {t("inicio")}
  </NavLink>
</li>
```

Figura 6.3: NavLink hacina de inicio

Para llevar a cabo esta funcionalidad, se ha usado la librería "React Router" y en este caso el componente <NavLink>que es un tipo de <Link>que tiene la capacidad de saber si está "activo". Por ello, en la implementación, como se ve en la Figura 6.3, mediante un ternario se comprueba si se encuentra activo o no y se elegirá un estilo u otro para resaltar el color del link como se ha visto anteriormente en la imagen del Menú lateral de navegación.

Figura 6.4: Menú superior

Lo que se ve en la Figura 6.4 representa el menú superior de la aplicación, el cual ofrece una serie de funcionalidades claves para el correcto uso de la aplicación. Entre estas funcionalidades se encuentran las notificaciones, donde mediante un pequeño icono al que le aparecerá una burbuja de color rojo indicando que hay mensajes, sobre el que se puede clicar para desplegar un pequeño menú donde aparecerá una lista de notificaciones. A la derecha hay un selector de idiomas entre castellano y valenciano, que al clicar, automáticamente se traducirán todos los textos de la aplicación al idioma seleccionado. La implementación de la traducción se ha llevado a cabo con la librería "react-i18next". Esta librería es un potente framework de internacionalización para React basado en la librería para Javascript "i18next". El módulo proporciona un componente para asegurar que las traducciones necesarias se carguen correctamente y que el contenido de la aplicación se renderice correctamente cuando se cambie el idioma. Para asegurar que el idioma seleccionado se mantiene aunque se recargue la página se ha hecho uso de "localStorage", la cual es una característica del navegador que permite almacenar datos de forma local en el navegador del usuario. A diferencia de las cookies, los datos almacenados en "localStorage" no tienen fecha de expiración y persisten incluso si el usuario cierra el navegador. Su uso se puede ver en las Figuras 6.5 y 6.6.

Se ha optado por usar un JSON [JSO24] para el diccionario de cada idioma. Dentro del componente, haciendo uno de la key 'es' para castellano y 'val' para el valenciano, se selecciona un idioma u otro.

Por último, a la derecha del menú se encuentra el nombre del usuario logueado en el momento actual, pudiendo ver su nombre, su imagen y su rol. Rol que más adelante se verá la importancia que tiene en la aplicación y la decisión que se ha tomado para llevar a cabo la implementación.

```
export const TopBarComponent = (props: Props) => {
  const [selectedLanguage, setSelectedLanguage] = useState<string>(() => {
    const savedLanguage = localStorage.getItem("selectedLanguage");
    if (savedLanguage) {
      return savedLanguage;
    } else {
      return "val";
    }
  });
```

Figura 6.5: Uso de localStorage para selectedLanguage

```
import esTranslations from './es.json'; // JSON diccionario español
import valTranslations from './val.json'; // JSON diccionario valenciano

const initialLanguage = localStorage.getItem('selectedLanguage') || 'val';

i18n
  .use(initReactI18next)
  .init({
    resources: {
      es: { translation: esTranslations }, // Diccionario español
      val: { translation: valTranslations }, // Diccionario valenciano
    },
    lng: initialLanguage, // idioma por defecto
    fallbackLng: 'val', // idioma alternativo por si falla la key de una traducción
    interpolation: {
      escapeValue: false,
    },
  });

export default i18n;
```

Figura 6.6: Traducción con react-i18next

Figura 6.7: Menú superior con notificaciones

Se puede ver el resultado del menú con notificaciones, con otro idioma y con una persona de otro rol logueada en la siguiente figura 6.7.

6.2.2. Interfaz *Poner notas*

La funcionalidad **Poner notas** se corresponde con la interfaz mediante la cual los profesores pueden introducir las notas de los estudiantes de una forma sencilla tras haber filtrado los alumnos.

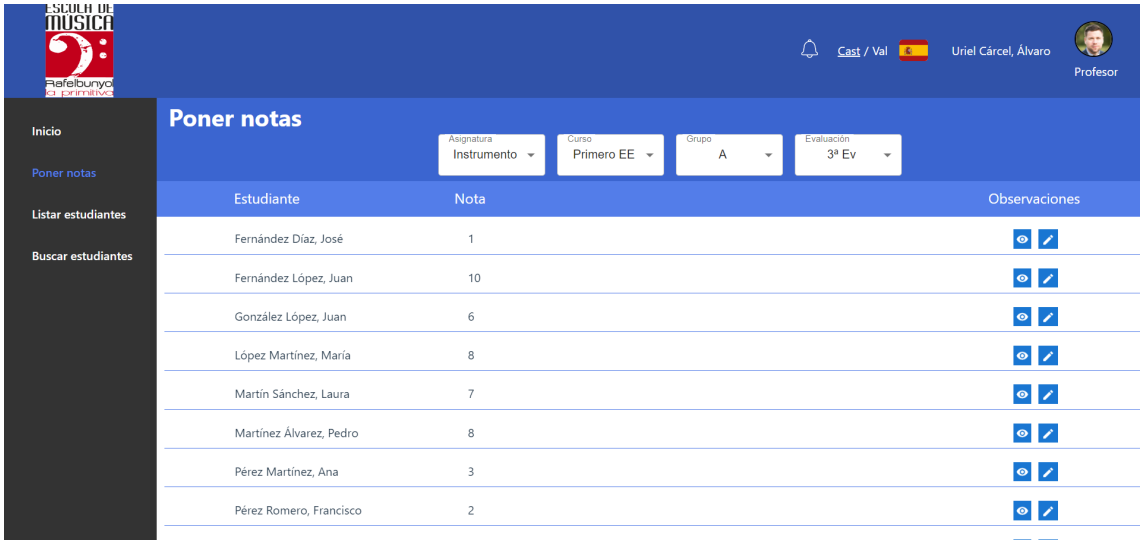


Figura 6.8: Interfaz *Poner notas*

Esta Figura 6.8 representa la visión completa que tendría el usuario al hacer uso de la funcionalidad **Poner notas**. Se observan dos secciones principales que se explican a continuación.

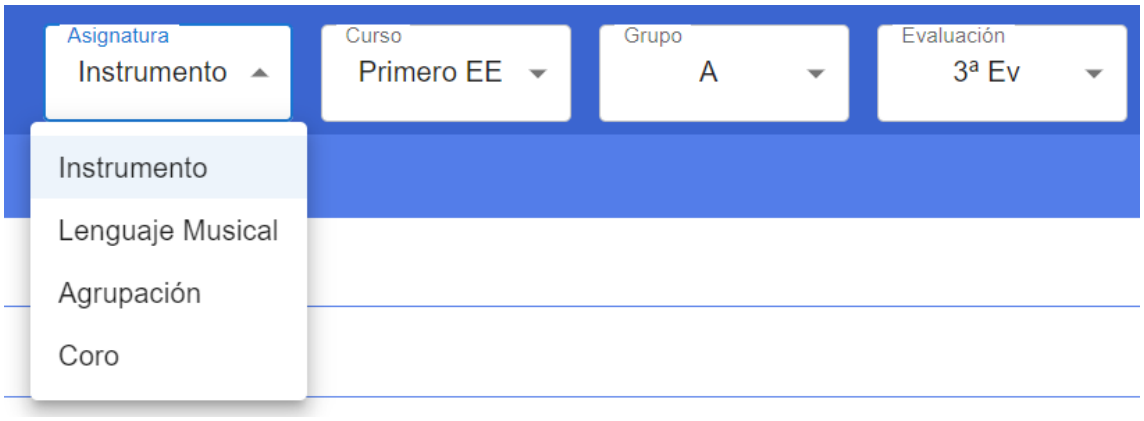


Figura 6.9: Filtros de *Poner notas*

En la primera sección del componente, como se ve en la Figura 6.9, el profesor debe filtrar en los *dropdowns* cuatro tipos de datos: la asignatura que quiere evaluar, el curso de la asignatura (las asignaturas tienen el mismo nombre a lo largo de los cursos) el grupo de sus alumnos y la evaluación.

En la Figura 6.10, se puede ver un ejemplo de la implementación de uno de los filtros en forma de *dropdown* o menú desplegable. Para ello se ha usado la librería "Material UI", una librería para React que ofrece componentes listos para ser usados. Estos componentes van desde un simple botón a componentes mucho más complejos según la necesidad que se tenga en el proyecto.

```
<FormControl sx={{ pr: 2 }}>
  <InputLabel id="cursoDropdown">{t("curso")}</InputLabel>
  <Select
    labelId="cursoDropdown"
    id="curso"
    className="dropDown"
    value={curso}
    label="Curso"
    onChange={handleChangeCurso}
  >
    <MenuItem value="Primero EE">{t("primero")}</MenuItem>
    <MenuItem value="Segundo EE">{t("segundo")}</MenuItem>
    <MenuItem value="Tercero EE">{t("tercero")}</MenuItem>
    <MenuItem value="Cuarto EE">{t("cuarto")}</MenuItem>
  </Select>
</FormControl>
```

Figura 6.10: *Dropdown* con MUI

Estudiante	Nota	Observaciones
González López, Juan	10	
González Martínez, Laura	7	
López Martínez, María	7	
Martínez Álvarez, Pedro	4	
Martínez Ruiz, Diego	2	

Figura 6.11: Listado de alumnos en *Poner notas*

La Figura 6.11 representa la segunda sección de la interfaz, donde tras haber hecho uso de los filtros, se carga en forma de tabla una serie de alumnos. De estos únicamente se muestra su nombre, ordenados alfabéticamente por el apellido, la nota de la última evaluación en caso de ya haber sido evaluados y dos botones, uno con un ojo y otro con un lápiz. Estos dos botones representan dos menús modales en los que los profesores podrán visualizar de una forma más individual la información de los alumnos, pudiendo ver también las observaciones escritas y con la posibilidad de navegar por las tres evaluaciones. El segundo botón contiene la funcionalidad de añadir o modificar tanto la nota como las observaciones del alumno. Se puede ver el aspecto de los menús en las Figuras 6.12 y 6.13.

Como se observa, la diferencia entre las dos imágenes es mínima, únicamente cambia el hecho de que los campos son editables, cumpliendo así con uno de los requisitos funcionales especificados por la escuela. El profesor al entrar puede comprobar que está en el menú modal del alumno correspondiente ya que su nombre aparece en la parte superior. En caso de querer editar las observaciones, el profesor verá la nota y observaciones actuales y en caso de querer añadir uno de los dos campos (o los dos) estos campos tendrán un *placeholder* que ayudará a entender para qué sirve cada campo. En el caso de las

Figura 6.12: Modal para observar las evaluaciones

Figura 6.13: Modal para editar las evaluaciones

observaciones aparecerá este *placeholder*: "Añade observaciones sobre el estudiante..." . En el campo de la nota simplemente aparecerá un 0 como nota modelo. Al aceptar, la información se introducirá en la base de datos y se mostrará por pantalla de forma correcta al recargar la página. En caso contrario, el profesor únicamente tendrá que darle al botón "Cancelar" para cerrar el menú.

6.2.3. Interfaz *Listar estudiantes*

La Figura 6.14 representa la segunda funcionalidad principal de la aplicación. **Listar estudiantes** está diseñada para que los profesores puedan acceder a una lista con información más ampliada de los alumnos. De la misma forma que la anterior interfaz, la forma de cargar a los estudiantes correspondientes es mediante filtros en forma de *dropdown*. En este caso son tres filtros los necesarios: asignatura, curso y grupo.

Una vez aplicados los filtros, aparecerá una lista con alumnos ordenados alfabéticamente que contendrá, la siguiente información: El instrumento principal del alumno (los alumnos pueden tener hasta dos instrumentos asignados a ellos mismos, pero en esta lista solo se necesita el primer instrumento), el nombre de su tutor (que coincide con su profesor de instrumento), su número de matrícula en el curso actual, el número de expediente (que representa un número único e invariable a lo largo de los años del estudiante dentro de la escuela) y por último, el número de teléfono de contacto de los padres o tutores legales.

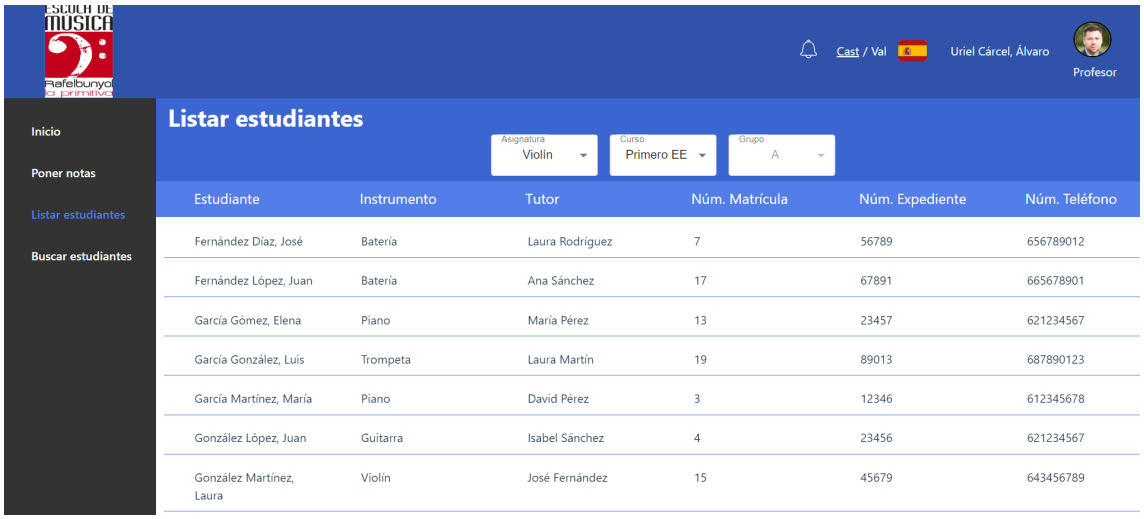


Figura 6.14: Interfaz de *Listar estudiantes*

Una implementación interesante que tiene esta interfaz se puede ver en la Figura 6.15:

```
<a
  href={`http://localhost:5175/search-students?name=${encodeURIComponent(
    student.alumno
  )}`}
  className="listItemListStudents1"
>
  {student.alumno}
</a>
```

Figura 6.15: Atajo para Buscar estudiante

Se ha añadido la posibilidad de poder clicar en el nombre completo de un alumno ya filtrado para acceder a la información de notas, redirigiendo la aplicación a la interfaz **Buscar estudiantes** que se explica en la siguiente sección.

6.2.4. Interfaz *Buscar estudiantes*

La última interfaz, representada en la Figura 6.16 es la de **Buscar estudiantes**. Esta proporciona una herramienta para los profesores con la cual pueden encontrar información sobre las calificaciones de los alumnos en un curso determinado.

De la misma forma que las anteriores interfaces, el profesor deberá introducir una serie de datos en los filtros para conseguir la información. En este caso, se cuenta con una barra de búsqueda y un *dropdown* para seleccionar el año académico del que se quiere obtener la información, ya que se almacenan los datos de todos los años cursados de los alumnos.

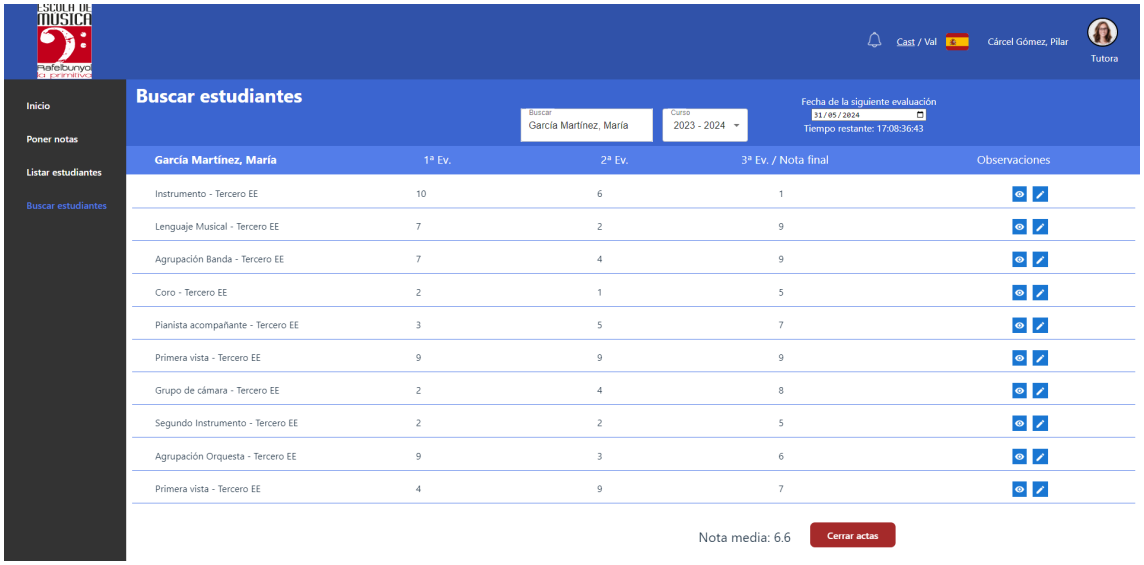


Figura 6.16: Interfaz *Buscar estudiantes*

En la barra de búsqueda se debe escribir el nombre exacto, tal cual aparece en la base de datos o si no, no aparecerá ningún resultado, para evitar que aparezca un alumno no deseado y se le introduzca una calificación errónea. Para que esta tarea sea menos tediosa, como se ha nombrado en el apartado anterior, desde la interfaz **Listar estudiantes** se puede clicar en un alumno ya filtrado y se escribirá automáticamente en el buscador de la interfaz de **Buscar estudiantes**.

Una vez aplicados los filtros, aparecerá una lista con todas las asignaturas que ha cursado el alumno ese año, junto con las notas de las tres evaluaciones. Como ya se ha visto en la interfaz de **Poner notas**, en esta también se encuentran los botones de visualizar y editar evaluaciones, ya que se puede dar la situación de necesitar modificar notas después de una evaluación o a final de curso.

Además de las notas individuales, debajo de la columna de la 3ª evaluación (o evaluación final) aparece la nota media del curso (calculada únicamente con las notas de esa evaluación), lo que proporciona al profesor una referencia rápida del rendimiento del alumno durante ese año. Además, mediante la nota media se establece el orden de elección de horario del año siguiente, por lo que es una información importante y no tiene que haber ningún error a la hora de calcularlo. Además, a la derecha de la nota aparece el botón de "Cerrar actas" con el cual el tutor daría por finalizada la evaluación.

Se observa por último, a la derecha de los filtros, la funcionalidad que tendrán los administradores para crear una nueva fecha de evaluación, como se ve en la Figura 6.17. En esta primera versión del prototipo se ha incorporado al rol del tutor para poder simular su funcionamiento.

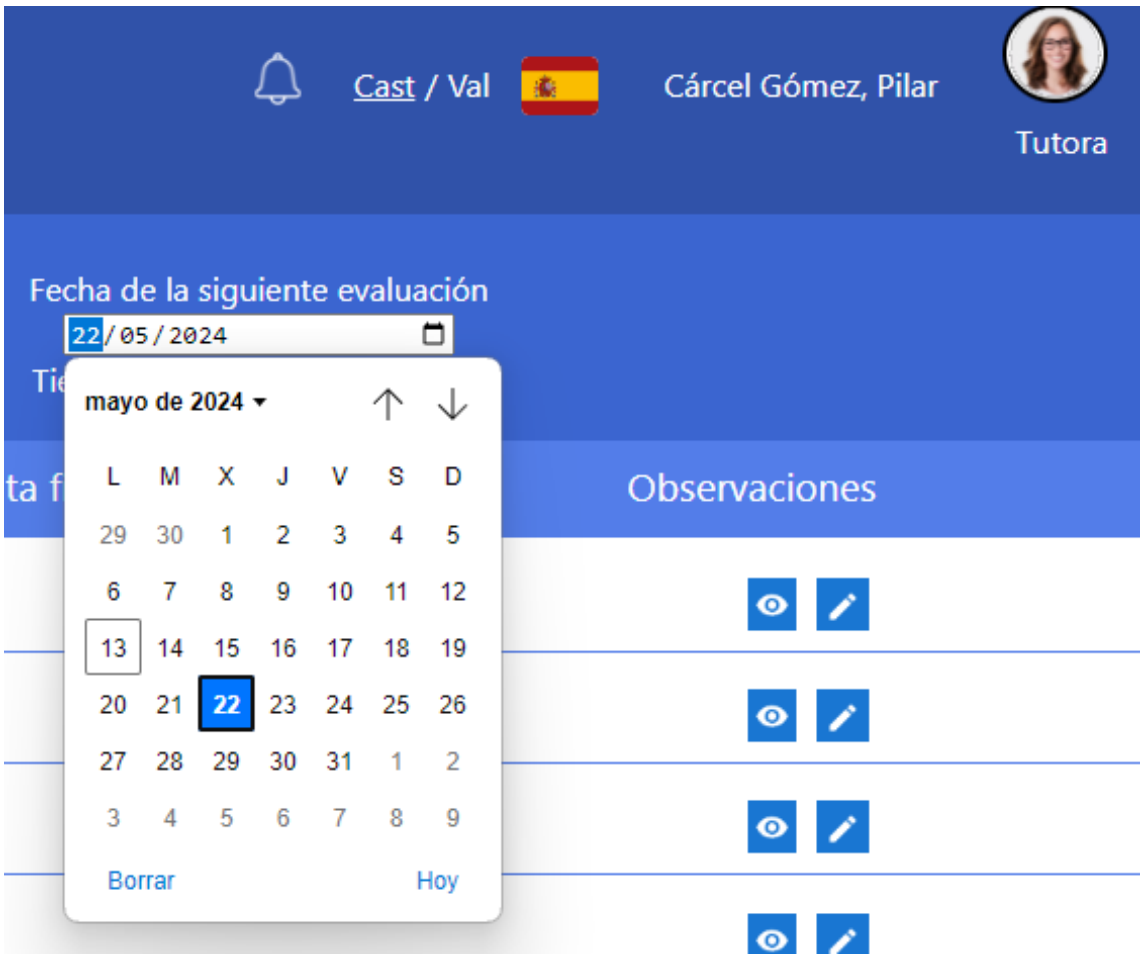


Figura 6.17: Crear fecha de evaluación

Aparece un menú desplegable en el que el tutor puede seleccionar el día de la siguiente evaluación. Una vez seleccionado aparece un contador en tiempo real que indica en días, horas, minutos y segundos el tiempo restante.

6.3 Desarrollo del *backend*

Para llevar a cabo el *backend* de la aplicación se ha decidido usar Python junto con el framework Flask para crear la API. La base de datos se ha gestionado con SQLite a través de DB Browser y se ha poblado usando scripts de Python.

6.3.1. Base de datos

El diseño de la base de datos se basa en tres tablas principales: Estudiante, Asignatura y Nota. A diferencia del diseño UML presentado en el capítulo Diseño de la solución, la estructura de las tablas de la base de datos se ha simplificado a la hora de desarrollar el prototipo. Es importante aclarar que esta herramienta es una prueba de concepto y el desarrollo del software final será independiente de este, por lo que no se consideró prioritario añadir la tabla de profesores en la base de datos. La forma en la que se resuelve el comportamiento de los roles se explica más adelante en la sección Roles: Tutor y Profesor.

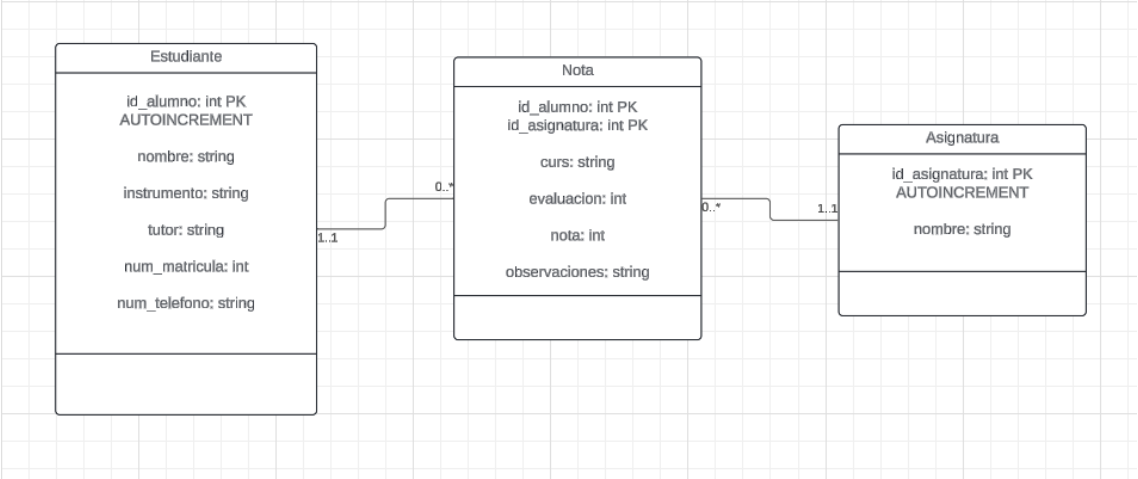


Figura 6.18: UML de la base de datos

Como se ve en la Figura 6.18, en las tablas existe la relación entre Estudiante y Nota y la relación entre Nota y Asignatura.

- **Estudiante y Nota**: Cada estudiante puede tener múltiples notas asociadas a diferentes cursos, asignaturas y evaluaciones. La relación es de cero a muchos (0:*), donde un Estudiante puede tener muchas Notas, pero cada Nota está asociada a un solo Estudiante.
- **Nota y Asignatura**: Cada asignatura puede tener múltiples notas registradas para diferentes estudiantes. La relación es de uno a muchos (0:N), donde una Asignatura puede tener muchas Notas, pero cada Nota está asociada a una sola Asignatura.

Con esta estructura, la tabla Nota se asegura que cada combinación de estudiante y asignatura sea única, permitiendo almacenar múltiples notas para diferentes evaluaciones y cursos.

Para la creación de las tablas, DB Browser ofrece una interfaz sencilla e intuitiva como se ve en la Figura 6.19.

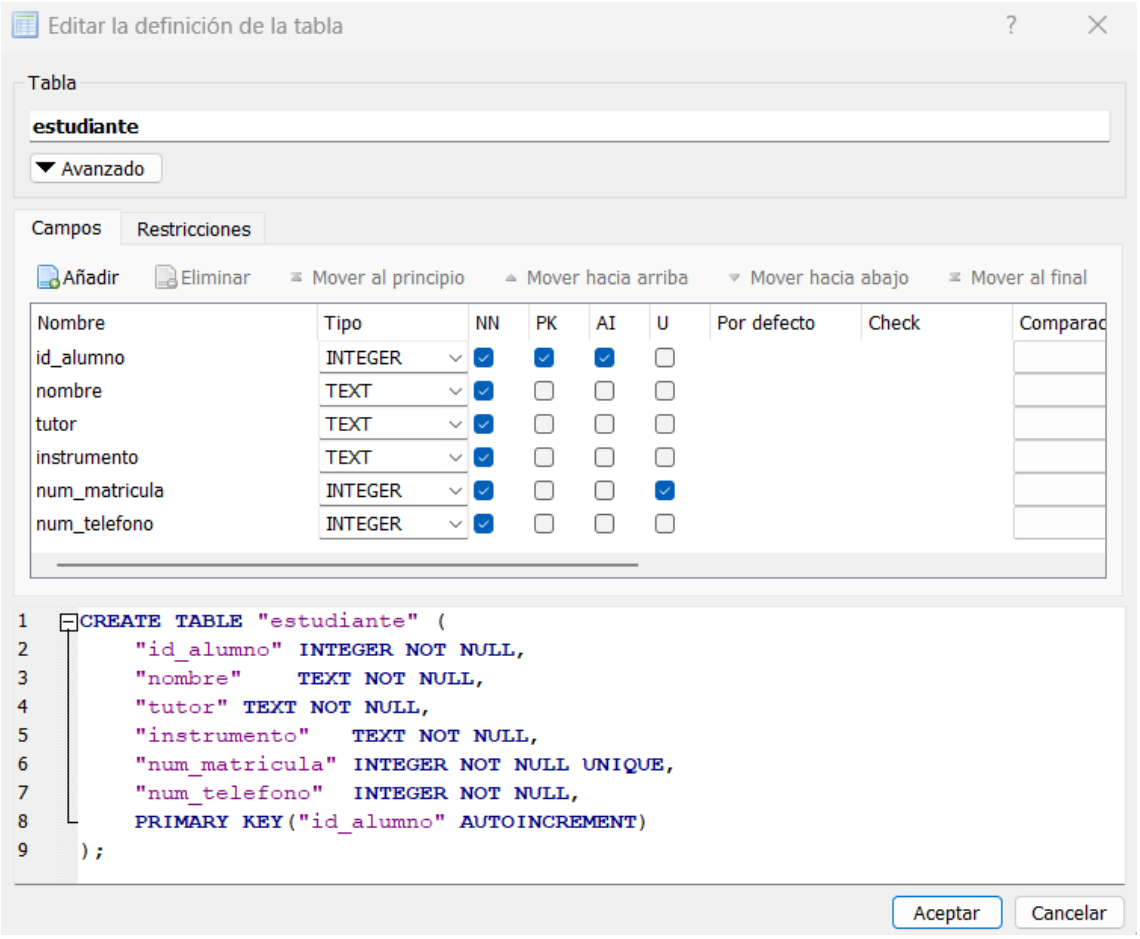


Figura 6.19: Creación de la tabla Estudiante en DB Browser

Como se observa, con el botón "Añadir" se puede añadir columnas a la tabla y se selecciona el Tipo de dato, además de si es: *not null*, *primary key*, *autoincrement* o *unique*.

Además de crear las tablas de esta manera, también se ha incluido la creación de las tablas en el código de la API, como se ve en la Figura 6.20.

Con esto se aseguran dos cosas importantes:

1. La creación de tablas en el código de la API asegura que, cada vez que la API se ejecuta, las tablas necesarias estarán presentes. Esto es especialmente útil en entornos de desarrollo y pruebas, donde la base de datos puede ser reinicializada con frecuencia.

2. Usar CREATE TABLE IF NOT EXISTS garantiza que las tablas solo se crean si no existen, evitando errores en la ejecución de la aplicación debido a la falta de tablas necesarias.

Para poblar la base de datos hay dos opciones: hacerlo a mano desde DB Browser o usar un script, en nuestro caso, con Python. Poblar la base de datos de forma manual es cómodo si no se necesita introducir muchos datos o si la tabla contiene pocas filas. Por ejemplo, como se ve en la Figura 6.21, la tabla asignatura fue rellenada a mano.

Con el botón de arriba a la derecha con el símbolo de más "+" se añade una fila vacía a la tabla y únicamente hay que añadir los datos a mano teniendo en cuenta el tipado.

```
# tabla asignatura
cursor.execute('''CREATE TABLE IF NOT EXISTS asignatura (
                    id_asignatura INTEGER PRIMARY KEY AUTOINCREMENT,
                    nombre TEXT NOT NULL
                )''')

# tabla estudiante
cursor.execute('''CREATE TABLE IF NOT EXISTS estudiante (
                    id_alumno INTEGER PRIMARY KEY AUTOINCREMENT,
                    nombre TEXT NOT NULL,
                    tutor TEXT NOT NULL,
                    instrumento TEXT NOT NULL,
                    num_matricula INTEGER NOT NULL UNIQUE,
                    num_telefono INTEGER NOT NULL
                )''')

# tabla nota
cursor.execute('''CREATE TABLE IF NOT EXISTS nota (
                    id_alumno INTEGER NOT NULL,
                    id_asignatura INTEGER NOT NULL,
                    curso TEXT NOT NULL,
                    nota INTEGER NOT NULL,
                    evaluacion INTEGER NOT NULL,
                    observaciones TEXT,
                    grupo TEXT,
                    FOREIGN KEY (id_alumno) REFERENCES estudiante(id_alumno),
                    FOREIGN KEY (id_asignatura) REFERENCES asignatura(id_asignatura),
                    PRIMARY KEY (id_alumno, id_asignatura)
                )''')
```

Figura 6.20: Creación de las tablas desde la API

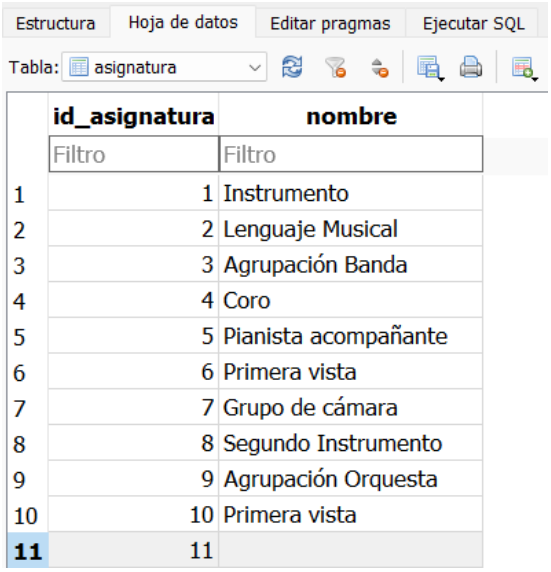


Figura 6.21: Añadir filas en la base de datos desde DB Browser

Para poblar la tabla de Notas donde se necesitan gran cantidad de datos ya que es necesario ponerle notas a todos los alumnos, para todas las asignaturas, en todos los cursos y en las tres evaluaciones, se ha optado por realizar un script en Python, como se puede ver en la Figura 6.22.

```python
def generate_observation():
    observations = [
        'Se porta mal y no hace los deberes',
        'Participa activamente en clase',
        'Necesita mejorar su atención en clase',
        'Tiene un buen desempeño académico',
        'Presenta dificultades en la materia'
    ]
    return random.choice(observations)

def generate_grade():
    return random.randint(1, 10)

def generate_curso():
    cursos = ['A', 'B', 'C']
    return random.choice(cursos)

def generate_data():
    data = []
    for student_id in range(1, 23):
        for subject_id in range(1, 11):
            for year in ['Primero EE', 'Segundo EE', 'Tercero EE', 'Cuarto EE']:
                for evaluation in range(1, 4):
                    observation = generate_observation()
                    grade = generate_grade()
                    curso = generate_curso()
                    data.append((student_id, subject_id, year, grade, evaluation, observation, curso))
    return data
```

Figura 6.22: Generación de datos con Script

Esto genera una lista con más de 2500 filas que posteriormente, añadiéndolas al fragmento de código de la Figura 6.23 y ejecutándolo, se añaden directamente a la base de datos.

```python
cursor.executemany('INSERT INTO nota (id_alumno, id_asignatura, curso, nota, evaluacion, observaciones, grupo) VALUES (?, ?, ?, ?, ?, ?, ?)',
                   [(1, 1, 'Primero EE', 5, 1, 'Presenta dificultades en la materia', 'C'),
                   ])
```

Figura 6.23: Código para introducir datos en la base de datos

6.3.2. Configuración de la API

Como se ha nombrado anteriormente, se ha usado Flask [Pyt24c], un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Como se observa en la figura 6.24 lo primero que hacemos es crear una instancia de la aplicación Flask, seguido de `conn = sqlite3.connect(...)` que se encarga de abrir una conexión a la base de datos ubicada en mi ordenador y de `cursor = conn.cursor()` el cual crea un cursor que se usa para ejecutar comandos SQL en la base de datos.

```
from flask import Flask, jsonify
from flask_cors import CORS
from flask import request
import sqlite3
import json

app = Flask(__name__)
CORS(app)
conn = sqlite3.connect("C:/Users/leiru/Desktop/escuelaDeMusica.db")
cursor = conn.cursor()
```

Figura 6.24: Configuración API con Flask

Para interactuar con los datos de la base de datos se han creado varios endpoints. A continuación se verá y analizará con detalle uno de ellos, en concreto el endpoint que se usa para filtrar a los estudiantes y posteriormente evaluarlos en la interfaz **Poner notas**.

```
@app.route("/filterGrades", methods=["GET"])
def filter_data_grades():
```

Figura 6.25: Definición del endpoint

El decorador de la Figura 6.25 define una ruta HTTP /filterGrades y especifica que solo acepta solicitudes GET. La función filter-data-grades es el controlador que se ejecutará cuando se acceda a esta ruta.

```
asignatura = request.args.get('asignatura')
curso = request.args.get('curso')
evaluacion = request.args.get('evaluacion')
grupo = request.args.get('grupo')
```

Figura 6.26: Parámetros de consulta

En la Figura 6.26 se obtienen los parámetros de consulta de la solicitud. Estos parámetros se utilizan para filtrar los datos y serán los parámetros recogidos por los *dropdowns*.

```
if not asignatura or not curso or not evaluacion:
    return jsonify({"error": "Missing required parameters"}), 400
```

Figura 6.27: Verificación de parámetros obligatorios

Se verifica que los parámetros obligatorios (asignatura, curso, evaluación) estén presentes, como se observa en la Figura 6.27. Si falta alguno, se devuelve un error 400 (Bad Request) con un mensaje de error en formato JSON.

```
query = '''
    SELECT
        e.id_alumno,
        e.nombre AS alumno,
        n.nota,
        n.observaciones
    FROM
        estudiante e
        INNER JOIN nota n ON e.id_alumno = n.id_alumno
        INNER JOIN asignatura a ON n.id_asignatura = a.id_asignatura
    WHERE
        a.id_asignatura = ? AND
        n.curso = ? AND
        n.evaluacion = ?
'''

params = [asignatura, curso, evaluacion]
```

Figura 6.28: Consulta SQL

Se construye la consulta SQL para seleccionar los estudiantes y sus notas que coincidan con los criterios especificados. Los parámetros se añaden a una lista llamada "params", como se observa en la Figura 6.28.

```
if grupo:
    query += ' AND n.grupo = ?'
    params.append(grupo)
```

Figura 6.29: Filtro opcional por Grupo

La Figura 6.29 representa el código necesario para comprobar si el parámetro grupo está presente en el *dropdown*, ya que filtrar por grupo es opcional en algunos casos. En caso de usar el filtro, se añade una condición adicional a la consulta SQL y se agrega el valor de grupo a la lista de parámetros.

```
conn = sqlite3.connect("C:/Users/leiru/Desktop/escuelaDeMusica.db")
cursor = conn.cursor()
cursor.execute(query, params)
rows = cursor.fetchall()
conn.close()
```

Figura 6.30: Ejecución de la consulta SQL

En primera instancia se establece una conexión con la base de datos SQLite. Después se crea un cursor y se ejecuta la consulta SQL con los parámetros. Por último se obtienen todos los resultados de la consulta y se cierra la conexión a la base de datos con "conn.close()" como se observa en la Figura 6.30.

```
data = [{
    'id_alumno': row[0],
    'alumno': row[1],
    'nota': row[2],
    'observaciones': row[3]
} for row in rows]
```

Figura 6.31: Serialización de datos a JSON

Se transforma cada fila de los resultados obtenidos en un diccionario con claves descriptivas (id_alumno, alumno, nota, observaciones). Estos diccionarios se almacenan en una lista llamada data como se ve en la Figura 6.31.

```
json_data = json.dumps(data, ensure_ascii=False).encode('utf-8')

return json_data, 200, {'Content-Type': 'application/json; charset=utf-8'}
```

Figura 6.32: Respuesta JSON

Se convierte la lista de diccionarios data a una cadena JSON y se devuelve esta cadena JSON como respuesta, con un código de estado HTTP 200 (OK) y el encabezado Content-Type especificado como application/json; charset=utf-8 para asegurar que el contenido se interprete correctamente como JSON y no haya problemas con las tildes ni con ciertos caracteres. Esto se observa en la Figura 6.32.

En la Figura 6.33 se ve un ejemplo de lo que devuelve la API en formato JSON:

6.4 Roles: Tutor y Profesor

En la aplicación desarrollada se han incluido dos roles distintos, representados en las Figuras 6.34 y 6.35: el Tutor y el Profesor. La diferencia esencial entre ambos es a nivel de permisos y responsabilidades que cada uno puede realizar dentro de la aplicación. Esta diferenciación basada en la jerarquía actual de la escuela de música, permite adaptar la experiencia del usuario y restringir el acceso a ciertas funcionalidades según el rol del usuario que está usando la aplicación.

Para gestionar estos permisos se ha implementado un sistema basado en un estado global, donde una variable binaria (*true* o *false*) determina el rol del usuario.

La forma de cambiar de usuario en la herramienta actualmente es clicando en la palabra 'Profesor' o 'Tutor/a', según el usuario que haya actualmente. Se ha optado por hacerlo de esta sencilla forma debido a que el objetivo era presentar la funcionalidad y que los clientes vieran realmente la diferencia dentro de la aplicación entre los dos roles.

Al cambiar de usuario, la aplicación adapta dinámicamente su comportamiento y presenta las opciones y funcionalidades correspondientes a cada rol. Al igual que con el idioma seleccionado, para los roles también se usa "localStorage" para no perder el usuario actual al recargar la página. La implementación, como se ve en la Figura 6.36, es similar a la hecha para las traducciones de la pagina.

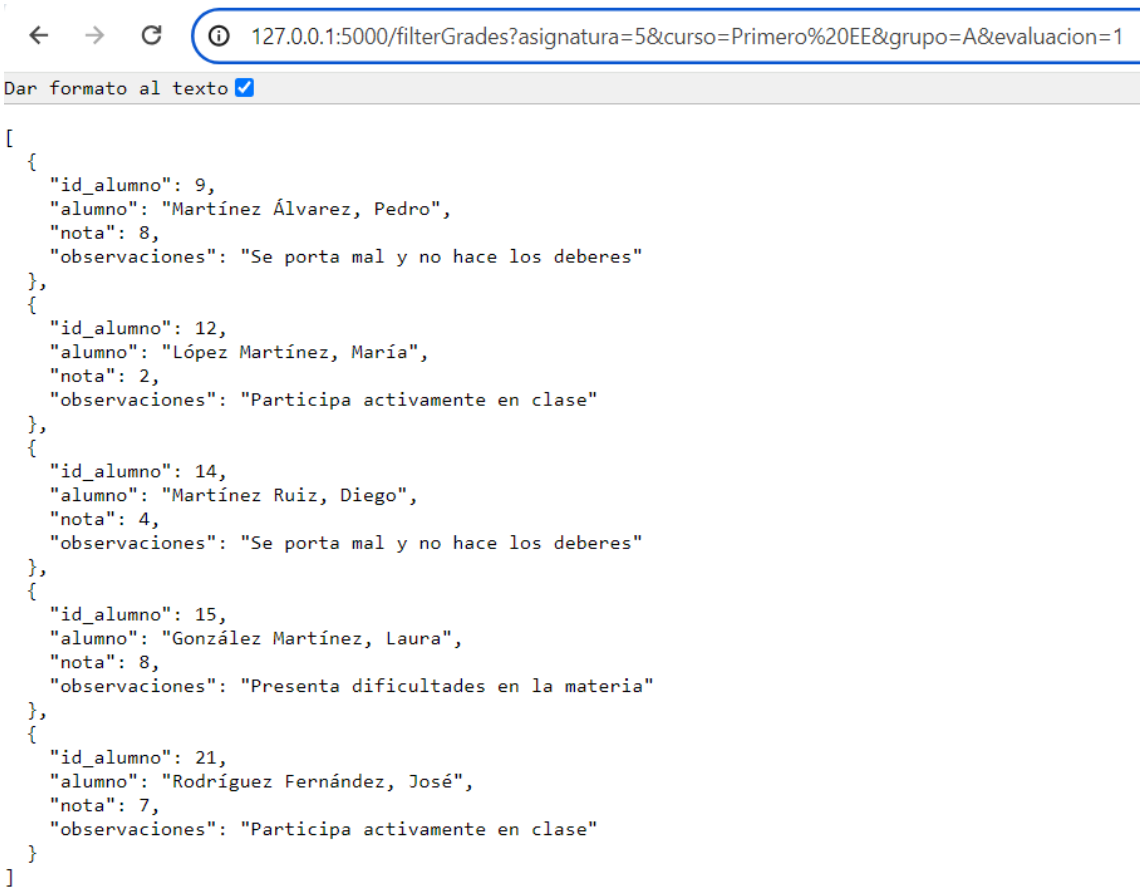


Figura 6.33: Respuesta de la API en el navegador

Figura 6.34: Cambiar entre Tutor y Profesor 1

Figura 6.35: Cambiar entre Tutor y Profesor 2

Tanto los profesores como los tutores únicamente pueden ver las asignaturas que imparten, la diferencia principal entre ambos es que los tutores imparten alguna asignatura de instrumento (Violín, piano, etc).

```
const [isProfessor, setIsProfessor] = useState<boolean>(() => {
  const savedValue = localStorage.getItem("isProfessor");
  return savedValue ? JSON.parse(savedValue) : false;
});

useEffect(() => {
  localStorage.setItem("isProfessor", JSON.stringify(isProfessor));
}, [isProfessor]);
```

Figura 6.36: localStorage para gestionar los roles al recargar la página

Otra diferencia importante se encuentra en la interfaz de **Buscar estudiantes**. Aquí el tutor es el único que tiene la capacidad de establecer la fecha de la siguiente evaluación y cerrar actas, por lo que el profesor tendrá estas acciones desactivadas, como se ve en la Figura 6.37.

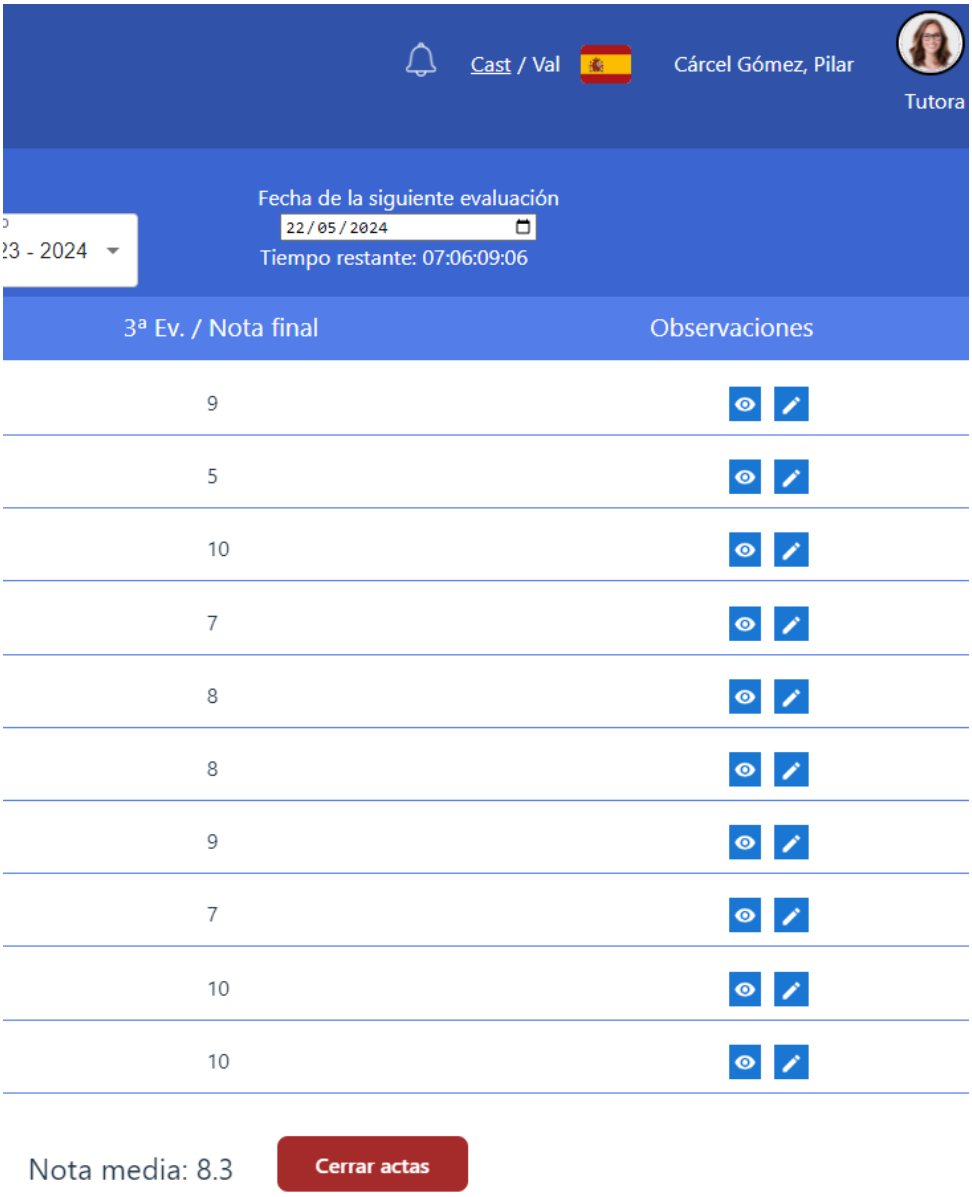


Figura 6.37: Vista del tutor en *Buscar estudiantes*

En el caso del profesor, verá el botón de Cerrar actas desactivado, de la misma forma que también verá el calendario de cambiar la fecha de la siguiente evaluación desactivado. En caso de que se haya acabado la evaluación y el tiempo restante del contador esté a cero, el profesor no podrá editar las notas, como se observa en la Figura 6.38, a diferencia del tutor que siempre tiene la capacidad de modificar las notas de sus alumnos. Esto se debe a que se puede dar la situación de que una vez acabada la evaluación, haya revisiones de notas, segundas correcciones de exámenes, etc.

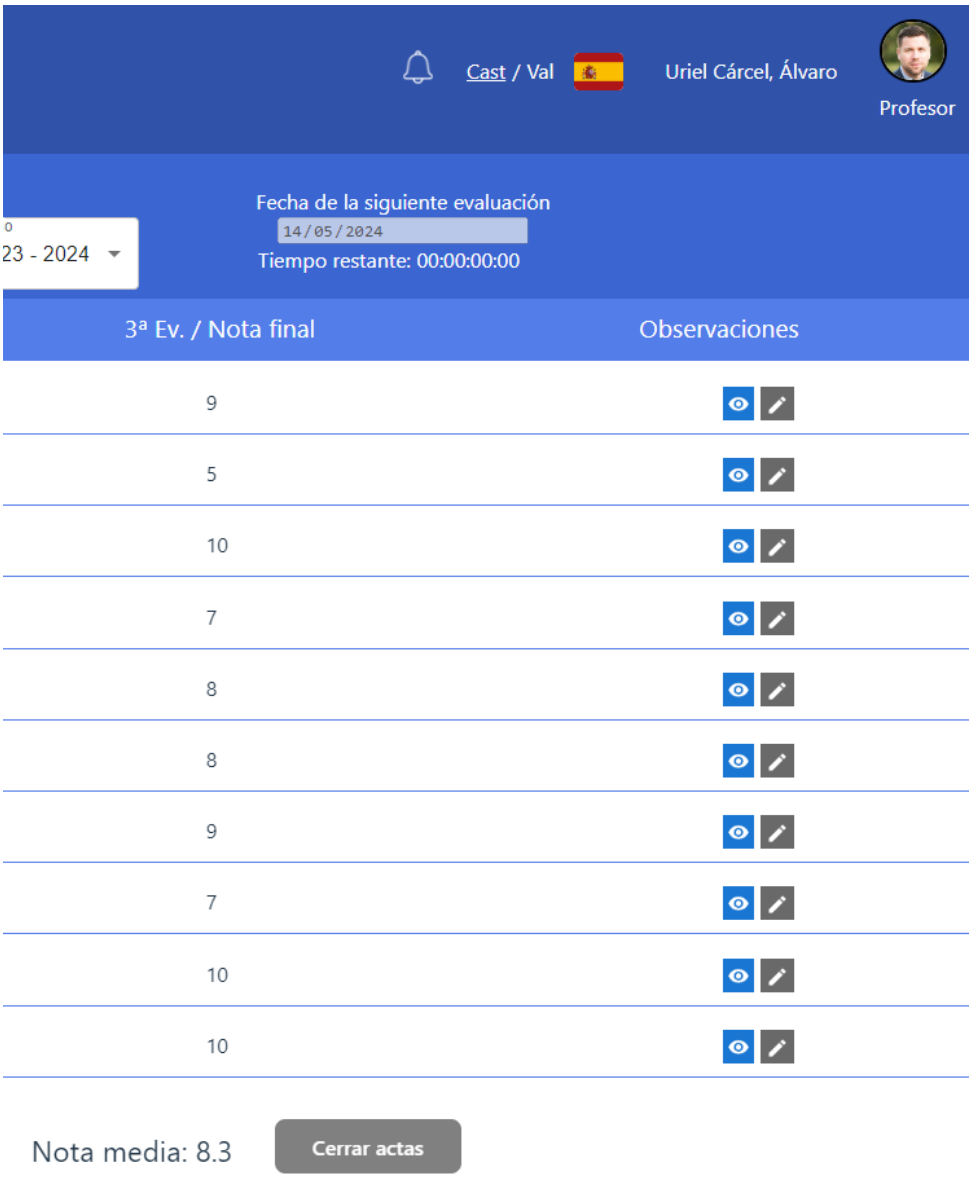


Figura 6.38: Vista del profesor en *Buscar estudiantes*

6.5 Integración *frontend* y *backend*

En esta sección se detallará cómo se han hecho las llamadas a la API desde el *frontend* y cómo se ha implementado para interactuar con el *backend*. A continuación se verá un ejemplo de como la interfaz **Poner notas** interactúa con la base de datos.

```
const filterStudents = async (asignatura, curso, grupo, evaluacion) => {
  try {
    console.log(evaluacion);
    const response = await fetch(
      `http://127.0.0.1:5000/filterGrades?asignatura=${asignatura}&curso=${curso}&grupo=${grupo}&evaluacion=${evaluacion}`
    );
    if (!response.ok) {
      throw new Error("Network response was not ok");
    }
    const data = await response.json();
    setFilteredStudents(data);
  } catch (error) {
    console.error("There was a problem fetching data:", error);
  }
};
```

Figura 6.39: filterStudents en *Poner notas*

Como se ve en la Figura 6.39 se ha implementado una función asíncrona que realiza una solicitud a la API para filtrar estudiantes según los parámetros necesarios. En este caso, cuatro parámetros que se le pasan a la API son los introducidos por los *dropdowns* de asignatura, curso, grupo y evaluación. Los *values* obtenidos de los filtros se pasan como parámetros a la URL para hacer la consulta. Seguidamente se verifica si la respuesta de la llamada a la API es correcta. Si no lo es, se lanza un error. En la variable "data" se extraen los datos JSON de la respuesta proporcionada por la API y se pasan a `setFilteredStudents` quien actualiza el estado de `filteredStudents`. Por último, si ocurre cualquier error en la ejecución del bloque "try", se captura el "catch" y se imprime un mensaje de error.

En la Figura 6.40 se ve como en la estructura que proporciona MUI para crear los *dropdowns*, hay un campo value, que es el que se encarga de manejar el estado y el que se le pasa a la URL de la API para hacer la llamada a la base de datos.

A continuación, como se ve en la Figura 6.41, se define el hook de React 'useState' que permite gestionar los datos que se reciben en forma de json, que tiene una estructura similar a la nombrada anteriormente en la Figura 6.33.

La variable filteredStudents contiene la lista de estudiantes filtrados que se va a renderizar.

El código de la Figura 6.42 permite mostrar una lista de estudiantes ordenada alfabéticamente, mostrando además la nota que tiene en la evaluación y los dos botones con los que se accedería a los menús modales. Con este fragmento de código `.sort((a, b) =>a.alumno.localeCompare(b.alumno))` se ordena la lista de estudiantes alfabéticamente por el nombre de estudiante que corresponde con el campo "alumno" del JSON. `.map((student, index) =>(...)` itera sobre cada estudiante en la lista ordenada pudiendo así renderizar una fila para cada estudiante. "student" representa al estudiante actual en la iteración, por lo que haciendo uso de "student.alumno" o "student.nota" se accede a la información que contiene el JSON que nos devuelve la API.

```
<FormControl sx={{ pr: 2 }}>
  <InputLabel id="cursoDropdown">{t("curso")}</InputLabel>
  <Select
    labelId="cursoDropdown"
    id="curso"
    className="dropDown"
    value={curso}
    label="Curso"
    onChange={handleChangeCurso}
  >
    <MenuItem value="Primero EE">{t("primero")}</MenuItem>
    <MenuItem value="Segundo EE">{t("segundo")}</MenuItem>
    <MenuItem value="Tercero EE">{t("tercero")}</MenuItem>
    <MenuItem value="Cuarto EE">{t("cuarto")}</MenuItem>
  </Select>
</FormControl>

<FormControl sx={{ pr: 2 }}>
  <InputLabel id="grupoDropdown">{t("grupo")}</InputLabel>
  <Select
    labelId="grupoDropdown"
    id="grupo"
    className="dropDown"
    value={grupo}
    label="Grupo"
    onChange={handleChangeGrupo}
  >
    <MenuItem value="A">A</MenuItem>
    <MenuItem value="B">B</MenuItem>
    <MenuItem value="C">C</MenuItem>
  </Select>
</FormControl>
```

Figura 6.40: Values de los *dropdowns*

```
const [filteredStudents, setFilteredStudents] = useState([]);
```

Figura 6.41: Hook para gestionar los datos recibidos por la API

```
{filteredStudents
  .sort((a, b) => a.alumno.localeCompare(b.alumno))
  .map((student, index) => (
    <React.Fragment key={index}>
      <div className="listRowListComponent">
        <div className="listItem">
          <span className="studentName">{student.alumno}</span>
        </div>
        <div className="listItem">
          <div>
            <span className="grade">{student.nota}</span>
          </div>
        </div>
        <div className="listItem">
          <div className="listItemButtons">
            <img
              src={eye}
              alt="Button 1"
              className="buttonEye"
              onClick={() => toggleModal("grades", student)}
            />
            <img
              src={edit}
              alt="Button 2"
              className="buttonEdit"
              onClick={() => toggleModal("edit", student)}
            />
          </div>
        </div>
      </div>
```

Figura 6.42: Renderizado de estudiantes en *Poner notas*

CAPÍTULO 7
Pruebas

En este capítulo se abordará la importancia de realizar pruebas en el desarrollo de software y se describirán las diferentes pruebas realizadas en la aplicación, tanto en la parte del *backend* como en el *frontend*, cubriendo así el correcto funcionamiento de los requisitos funcionales y los requisitos no funcionales.

7.1 Pruebas en el *backend*

Para las pruebas en la parte del *backend* que se ha programado en Python se va a utilizar el software de automatización de pruebas Pytest [Pyt24a]. Se trata de una herramienta muy potente y efectiva y fácil de usar que simplifica considerablemente esta tarea. Se ha elegido Pytest debido a su diseño intuitivo y a su capacidad para manejar una variedad de casos de prueba. A continuación se verá cómo se ha integrado en el proyecto para realizar las pruebas.

En primer lugar hay que crear el archivo donde se escribirán las pruebas. El nombre de este archivo tiene que comenzar por "test_" ya que es lo que buscará Pytest en nuestro directorio para ejecutar las pruebas. En este caso se ha llamado "test_app.py" y se han definido las siguientes pruebas:

- `def test_filter_data(client)`
- `def test_filter_grades(client)`
- `def test_filter_grades_test(client)`
- `def test_filter_search(client)`
- `def test_edit_nota(client)`

En la Figura 7.1 se ve un ejemplo de código de una de las pruebas:

Pytest utiliza fixtures para configurar datos de prueba antes de ejecutar las pruebas, lo que garantiza un entorno consistente para cada una de ellas. Por ejemplo el que se observa en la Figura:

Como se observa, con Pytest se es capaz de hacer aserciones simples pero efectivas, por ejemplo con "assert response.status_code == 200" puede comprobar si ha existido la respuesta HHTP 200 y por tanto que la solicitud ha sido exitosa. En segundo lugar, test comprueba si el JSON devuelto por la API al hacer la llamada coincide con el "expected_data" proporcionado previamente. En caso de que la aserción sea falsa, Pytest

```python
def test_filter_grades_test(client):
    expected_data = [
  {
    "alumno": "Uriel Cárcel, Nicolás",
    "nota": "9",
    "observaciones": "Tienes un 9"
  }
]
    response = client.get('/filterGradesTest?idalumno=2&asignatura=1&curso=Primero+EE&evaluacion=1&grupo=A')
    assert response.status_code == 200
    assert response.json == expected_data
```

Figura 7.1: Test de filter_grades_test

```python
@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client
```

Figura 7.2: Fixture de configuración

informará el fallo y proporcionará detalles. Para ilustrar esto, se ha provocado un error pasándole una nota errónea en el"expected_data" donde en la base de datos "nota = 9", pero se le ha pasado "nota = 10". En la Figura 7.3 se observa la salida por consola de Pytest indicando el error.

```
=================================================== FAILURES ===================================================
_____________________________________________ test_filter_grades_test _____________________________________________

client = <FlaskClient <Flask 'app'>>

    def test_filter_grades_test(client):
        expected_data = [
      {
        "alumno": "Uriel Cárcel, Nicolás",
        "nota": "10",
        "observaciones": "Tienes un 9"
      }
    ]
        response = client.get('/filterGradesTest?idalumno=2&asignatura=1&curso=Primero+EE&evaluacion=1&grupo=A')
        assert response.status_code == 200
>       assert response.json == expected_data
E       AssertionError: assert [{'alumno': '...Tienes un 9'}] == [{'alumno': '...Tienes un 9'}]
E
E         At index 0 diff: {'alumno': 'Uriel Cárcel, Nicolás', 'nota': '9', 'observaciones': 'Tienes un 9'} != {'alumno'
: 'Uriel Cárcel, Nicolás', 'nota': '10', 'observaciones': 'Tienes un 9'}
E         Use -v to get more diff

test_app.py:221: AssertionError
=========================================== short test summary info ============================================
FAILED test_app.py::test_filter_grades_test - AssertionError: assert [{'alumno': '...Tienes un 9'}] == [{'alumno': '...T
ienes un 9'}]
========================================== 1 failed, 4 passed in 0.21s ==========================================
```

Figura 7.3: Test failed

En caso de que todos los tests sean correctos, la salida por consola de Pytest será la de la Figura 7.4:

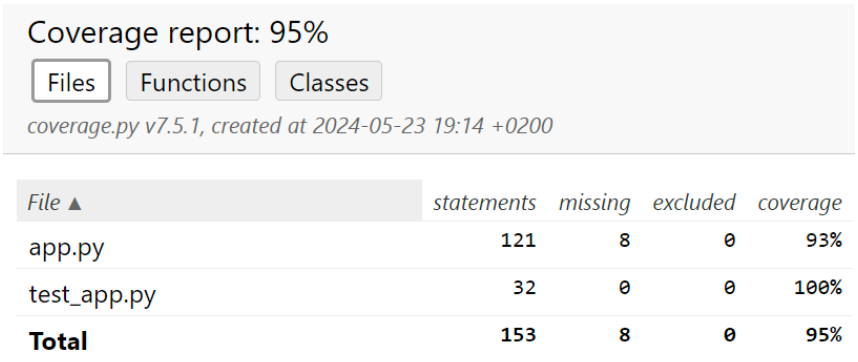
```
========================================== test session starts ==========================================
platform win32 -- Python 3.12.2, pytest-8.2.1, pluggy-1.5.0
rootdir: C:\Users\leiru\Desktop\tfg-musica\tfg-musica-frontend\src\api
collected 5 items

test_app.py .....                                                                              [100%]

========================================== 5 passed in 0.16s ==========================================
PS C:\Users\leiru\Desktop\tfg-musica\tfg-musica-frontend\src\api>
```

Figura 7.4: Pytest en la API

Para determinar la cobertura de las pruebas que se han realizado con Pytest se ha utilizado su extensión Coverage. Al lanzar la ejecución se generará un directorio ./htmlcov donde abriendo su archivo index.html se puede ver lo que se aprecia en la Figura 7.5:


Coverage report: 95%

Files Functions Classes

coverage.py v7.5.1, created at 2024-05-23 19:14 +0200

File ▲	statements	missing	excluded	coverage
app.py	121	8	0	93%
test_app.py	32	0	0	100%
Total	153	8	0	95%

Figura 7.5: Porcentaje de coverage

En este caso, con los tests realizados previamente hay un 93 % de cobertura en el código de la API (app.py). Desde el archivo index.html generado por coverage, se puede entrar en los diferentes archivos para ver la cobertura de cada uno de ellos.

Para realizar las pruebas de estrés, como requisito no funcional, se ha usado Locust [Loc24]. Locust es una librería de código abierto de Python que permite definir una serie de rutas a probar y configurando la ruta, el número de usuarios y el crecimiento de los mismos (simulando que es por segundo) se obtiene un informe del comportamiento de la aplicación en el entorno donde se está ejecutando.

En la Figura 7.6 se observa el ejemplo de una ruta que ejecutará el Locust.

```python
from locust import HttpUser, TaskSet, task, between

class UserBehavior(TaskSet):

    @task(1)
    def filter_data(self):
        self.client.get("/filter", params={
            "asignatura": 1,
            "curso": "Primero EE",
            "grupo": "A"
        })
```

Figura 7.6: Código para ejecutar Locust

Al ejecutar Locust desde la consola, aparece la ruta http://0.0.0.0:8089 desde donde se podrá configurar el entorno de Locust.

Una vez dentro se verá una pantalla como la de la Figura 7.7 donde habrá que configurar la aplicación. En ésta se puede configurar el número de usuarios totales, el número de usuarios que se crean por segundo y la ruta de entrada a la aplicación.

El análisis por pantalla será parecido al de la Figura 7.8, donde se puede ver las peticiones por segundo (RPS) y el número de fallos en caso de que los haya. En la gráfica de abajo se observa el tiempo de respuesta el milisegundos de la aplicación. La aplicación soporta sin problemas 200 usuarios concurrentes.

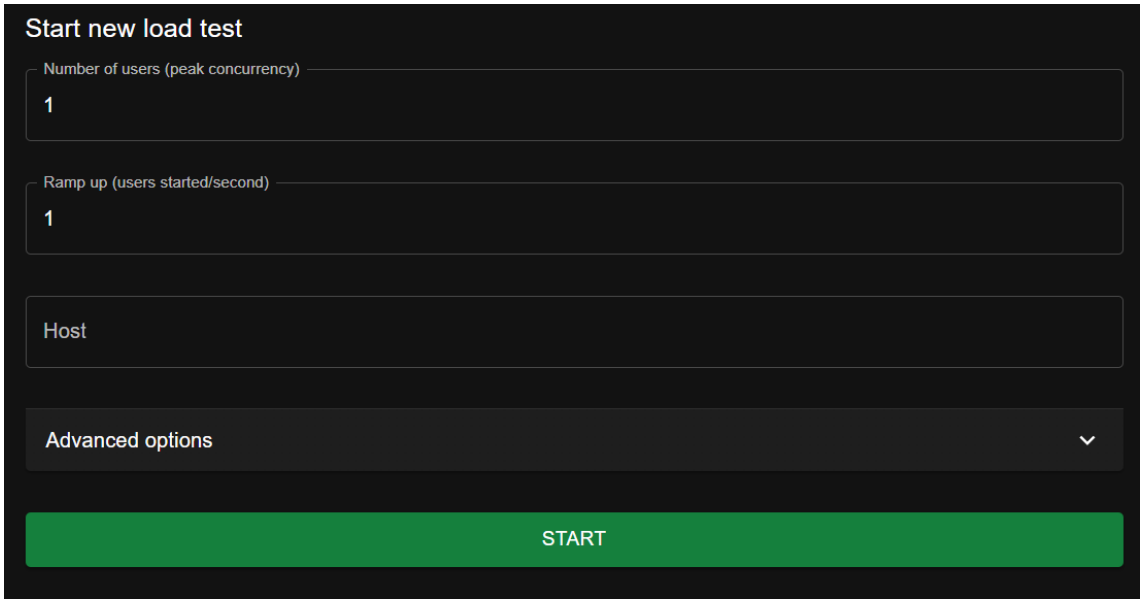


Figura 7.7: Configuración de Locust

Figura 7.8: Gráfico de Locust

Por tanto, se observa cómo se cumplen los requisitos no funcionales de soportar múltiples usuarios simultáneamente sin perder rendimiento, y el tiempo de respuesta de la aplicación responde las solicitudes en un tiempo muy considerable, con una media de 6ms por respuesta.

7.2 Pruebas del *frontend*

Para realizar las pruebas del *frontend* se han comprobado que se cumplen los requisitos esperados en las interfaces. La web funciona en Google Chrome, Microsoft Edge y en Mozilla Firefox, tres de los cuatro navegadores más usados en el mundo.

Para comprobar su usabilidad y ver cómo de intuitiva y fácil de usar es, se tuvo una reunión con la subdirectora de la escuela. En esta reunión se le proponía realizar una serie

de pasos dentro de la aplicación para ver cómo se manejaba un nuevo usuario usando la web por primera vez.

Los pasos que debía realizar eran los siguientes:

- Ponerle nota a Alicia Villanueva García en la 3ª evaluación de la asignatura de Violín, siendo ella una alumna de Cuarto EE.
- Encontrar el número de teléfono de María López Martínez, la cual cursa Lenguaje Musical y está en Segundo EE en el grupo B.
- Cambiarle cualquier nota a María López Martínez en el año académico 2023-2024.

Durante la prueba, se debía ir cambiando entre usuarios dentro de la aplicación para poder realizar todas las funcionalidades, ya que había algunas específicas del rol de tutor, como cambiar la fecha de la evaluación para poder editar las notas.

La subdirectora de la escuela pudo superar la prueba sin muchas complicaciones, pero sacamos algunas conclusiones. Los nombres del menú de navegación de la izquierda deberán pasar a llamarse de la siguiente forma:

- **Listar estudiantes** pasará a llamarse **Mis alumnos**.
- **Buscar estudiantes** pasará a llamarse **Informe notas**.

De esta manera, una persona que usa por primera vez la aplicación encuentra los nombres que aparecen en el menú más intuitivos. Otra de las cosas que se observó es el funcionamiento actual del buscador, ya que actualmente hay que escribir de forma completa y literal como está en la base de datos el nombre del alumno que se quiere buscar el acta. En un futuro se puede plantear un sistema de búsqueda más flexible. Por último, otro de los requisitos para un futuro es que al modificar las notas, se actualicen en el *frontend* sin necesidad de recargar la página. Debido a las fechas en las que se hicieron las pruebas, estos cambios estarán en la versión final cuando se entregue el prototipo al cliente, con fecha después de la entrega del TFG.

CAPÍTULO 8
Trabajo futuro

En cuanto al trabajos futuros de cara a mejorar la aplicación, me gustaría dividirlo en dos tipos: en Nuevas funcionalidades y en Funcionalidades pendientes.

8.1 Nuevas funcionalidades

En este apartado, pretendo aportar nuevas funcionalidades no planteadas desde un principio por la escuela, pero que considero muy importantes de cara a llevar a cabo una implementación de un software para la gestión de una escuela de música.

- También sería interesante implementar una funcionalidad que devolviera estadísticas de aprobados/suspendidos de cada asignatura y gráficos con la media de notas según los filtros que se deseara.
- La escuela está muy acostumbrada a usar Whatsapp para mantener las comunicaciones entre los profesores, pero creo que sería interesante implementar un sistema de chat o mensajería interna desde dentro de la aplicación. Esto daría mucha más importancia al sistema de notificaciones ya implementado, e incluso se podrían conectar los mensajes de dentro de la aplicación con el correo electrónico lo que mejoraría la eficacia de la comunicación.
- Añadir una nueva funcionalidad tanto para los alumnos como para los profesores donde pueden consultar y modificar a lo largo del curso los horarios de las clases y de los eventos de la escuela.

8.2 Funcionalidades pendientes

En esta sección se aportarán posibles mejoras sobre las tareas realizadas que deberán abordarse de cara a un desarrollo final y funcionalidades sugeridas por la escuelas que no se han implementado en un primer lugar en el prototipo.

- La primera funcionalidad y más importante sería añadir un sistema de gestión de usuarios para que cada profesor pudiera acceder a la aplicación mediante un usuario y contraseña.
- Considero que con el avance de la digitalización, sería interesante que los alumnos y las familias también tuvieran acceso a la web y pudieran consultar información sobre sus calificaciones.

- Actualmente en la escuela, la forma de realizar la matrícula año a año es mediante los formularios que ofrece Google. En estos recoge diferentes datos de los alumnos y finalmente se exportan a un Excel. El siguiente paso de la aplicación sería tener la capacidad de importar el xml generado de la matrícula y poder integrar automáticamente los datos en la base de datos por lo que el proceso de actualizar los datos de estudiantes actuales, o el de añadir nuevos estudiantes sería mucho más sencillo y rápido.
- Hoy en día, la forma que tiene la escuela de comunicar las calificaciones de los alumnos a los padres es mediante Whatsapp o por correo electrónico. Sería interesante plantear entregar las notas de forma física, como estamos acostumbrados a recibirlas en primaria y secundaría. Por ello, lo que propongo es implementar la opción de exportar a PDF las notas para poder imprimirlas.

CAPÍTULO 9
Estado del arte

En el ámbito de la educación, sobre todo en nuestro caso, en las escuelas de música, la digitalización ha tomado un papel muy importante en la gestión de las escuelas. Es por ello, que a lo largo de los años han aparecido numerosas herramientas que han tratado de dar solución a esta evolución.

Por esto, se van a analizar algunas herramientas actuales que implementan funcionalidades similares a las planteadas en este proyecto.

Kydemy [Get24] es una plataforma online que ofrece servicios de gestión académica para instituciones educativas, incluyendo escuelas de música. Algunas de sus características principales son:

- Gestión de alumnos y profesores.
- Planificación de clases.
- Gestión de evaluaciones.
- Seguimiento del progreso mediante estadísticas.
- Comunicación entre profesores y familias.

Como se ve en la Figura 9.1, Kydemy ofrece una interfaz agradable, pero considero que con una saturación de información y de botones, sobretodo en la parte superior de la aplicación, que pueden complicar la experiencia al usuario.

Mn program [MNp24] es un software para la gestión de empresas y de profesionales autónomos que ofrece diferentes funcionalidades como:

- Agenda y recordatorios.
- Gestión de clientes.
- Gestión de proyectos.
- Contabilidad y facturación.
- Soporte técnico y actualizaciones.

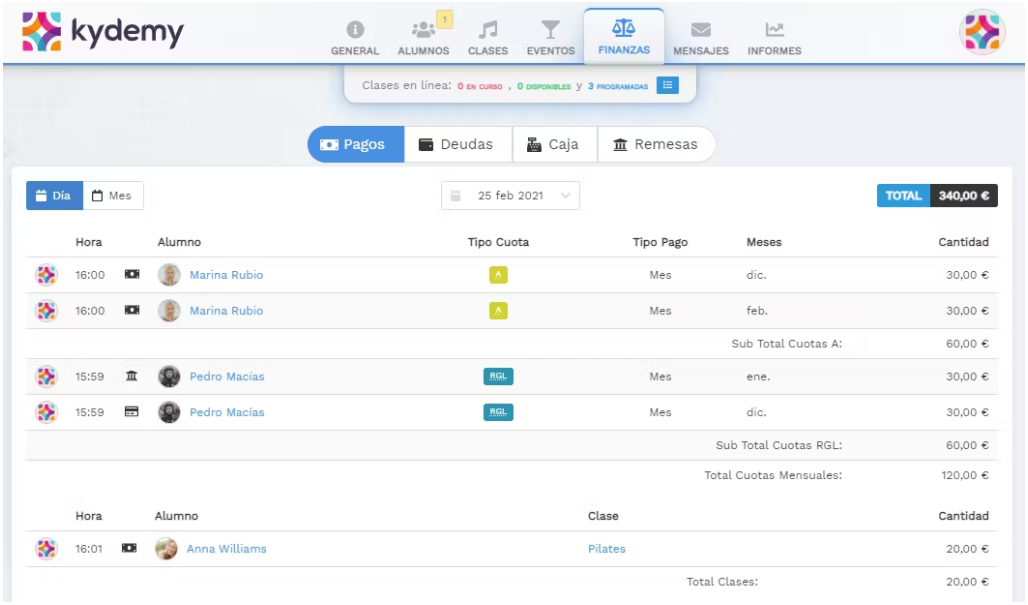


Figura 9.1: Kydemy interfaz Finanzas

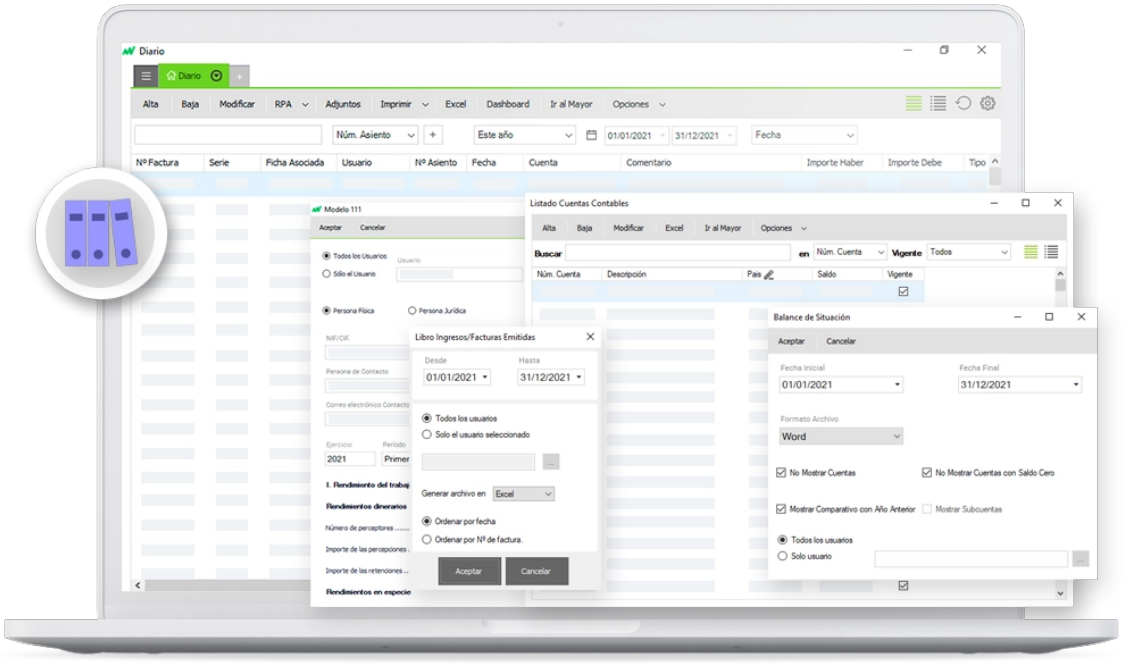


Figura 9.2: Interfaz de Mn Program

Como se ve en la Figura 9.2, este software tampoco se caracteriza por su simplicidad en las interfaces y facilidad de uso, por lo que considero que tampoco es una buena opción para cumplir con los requisitos que exige la escuela de música. Además, a pesar de que puede parecer que Mn Program cuenta con funcionalidades que pueden encajar para la gestión de una escuela de música, este no es un software específicamente para ello por lo que no cubriría correctamente todas las necesidades de una escuela.

PowerSchool [Cap24] es una plataforma de gestión escolar integral que incluye funciones para la gestión de estudiantes, personal, calificaciones, asistencia, y comunicación con padres y tutores.

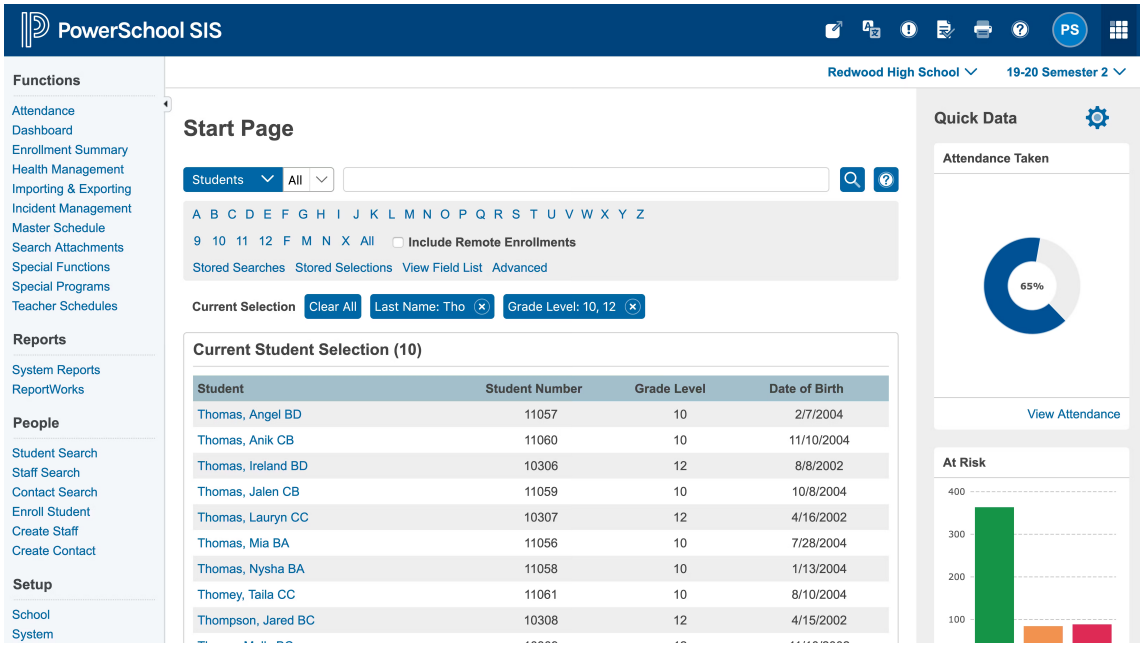


Figura 9.3: Interfaz de PowerSchool

Como se observa en la Figura 9.3, sigue teniendo el problema de contener demasiada información y links a los que poder clicar dentro de una misma interfaz por lo que el usuario puede sentirse abrumado con tanta información, lo que causa que tarde más tiempo en realizar las acciones que desea. Además tiene un proceso de configuración complejo y costoso que se aleja de los deseos de la escuela.

En conclusión, considero que no hay ninguna aplicación que cumpla con las especificaciones y requisitos demandados por la escuela de música. El deseo de la escuela de tener un software propio viene de la necesidad de implementar funcionalidades peculiares para soportar el comportamiento actual de la empresa. Por ejemplo, el hecho de que todos los profesores puedan consultar las notas de todas las asignaturas de los alumnos, es algo complicado de encontrar en un software no hecho a medida para ellos. Existen softwares que cumplen con algunas funcionalidades de las esperadas por la escuela, pero todas añaden cierta dificultad o demasiada información innecesaria para esta escuela en concreto.

En mi caso, mi aplicación busca satisfacer las necesidades de la escuela de una forma sencilla, sin cargar de información al usuario y sin proporcionar más acciones dentro de la web de las necesarias, por lo que el usuario podrá realizar todas las acciones que necesita en el menor tiempo posible. Además la interfaz está pensada para que tenga un diseño sencillo y agradable a la vista, además de un toque profesional, pudiéndose personalizar con toques de la escuela, como su logo.

CAPÍTULO 10
Conclusiones

Este trabajo ha representado un antes y un después en mí y en la forma que tendré en un futuro en afrontar los retos. A lo largo del proyecto he tenido la oportunidad de enfrentarme a ciertos desafíos que seguro que me van a hacer crecer tanto a nivel personal como profesional.

En primer lugar, el hecho de haber cumplido con el plazo de tiempo que me había propuesto para finalizar el proyecto es un hecho que me da una gran satisfacción por lo que significa a nivel laboral. Comenzar desde cero el proyecto y llevarlo a cabo de manera individual ha sido una experiencia muy enriquecedora. He querido usar tecnologías con las que no tenía apenas experiencia para aumentar aún más el reto personal que me había propuesto. Esto ha resultado en ampliar mis habilidades y conocimientos en programación y teniendo una visión más amplia del desarrollo de software que seguro que me ayuda en proyectos futuros.

Con este proyecto, lejos de sentirme intimidado, como hubiera sucedido en un pasado, me ha hecho ver que a los obstáculos y a los errores que persisten durante días se les acaba encontrando solución. Esa capacidad de resolver problemas de manera autónoma ha sido uno de los aprendizajes más valiosos que he adquirido desarrollando el proyecto.

En conclusión estoy contento porque considero que en general se han cumplido las expectativas y los requisitos de la escuela. No obstante, hay algunas correcciones pendientes que se resolverán después de la entrega del TFG, pero antes de la entrega definitiva a los clientes. Esto asegurará que se les entregue una herramienta eficiente y completamente adaptada a sus necesidades y que tengan total capacidad para decidir si quieren implementarla en la escuela.

CAPÍTULO 11
Agradecimientos

Como última tarea antes de acabar la memoria, me gustaría dar mi más sincero agradecimiento a todas las personas que han colaborado y que me han ayudado durante la realización del TFG.

En primer lugar, a mi tutora, Alicia, por ser la mejor tutora que podría haber tenido. Gracias por tu constante ayuda, por estar siempre disponible para resolverme las dudas y guiarme a lo largo de este proyecto. Sobretodo gracias por la implicación los últimos días previos a la entrega del trabajo. Me gustaría que no se te olvidara el comentario de Gema en la última reunión, quién sin duda acertó diciendo que había tenido mucha suerte al tenerte de tutora.

Agradecer también a la escuela de música "La primitiva", por su colaboración durante todo el proceso y por proporcionarme toda la información necesaria para el correcto desarrollo del proyecto. Gracias por el interés mostrado en las reuniones, que desde luego fueron clave para finalizar el proyecto.

Por supuesto agradecer a mis amigos Edu, Adri y Gabri, quienes me han echado una mano en los momentos que más atascado estaba y han hecho mucho más ameno estos meses mientras pasábamos el rato diciendo tonterías por Discord.

Por último, agradecer a mis padres y a mi hermano por todo el apoyo que siempre me han dado y por siempre confiar en mi. Gracias por intentar que siempre siga el mejor camino y por ser mis mayores apoyos.

Bibliografía

[And10] David J Anderson. *Kanban: Cambio evolutivo exitoso para su negocio de tecnología*. Blue Hole Press, 2010.

[Ang24] Angular. Angular, May 2024. https://angular.dev/.

[Bro24] DB Browser. DB Browser for SQLite, May 2024. https://sqlitebrowser.org/.

[Cam24] CampusMVP. TypeScript contra JavaScript: ¿cuál deberías utilizar?, May 2024. https://www.campusmvp.es/recursos/post/typescript-contra-javascript-cual-deberias-utilizar.aspx/.

[Cap24] Capterra. PowerSchool Student Information System, May 2024. https://www.capterra.es/software/154883/powerschool-student-information-system/.

[Cod24] Visual Studio Code. Visual Studio Code - Code Editing. Redefined, May 2024. https://code.visualstudio.com/.

[CSS24] CSS. Cascading Style Sheets (CSS), May 2024. https://es.wikipedia.org/wiki/CSS/.

[Del24] Deloitte. Ceremonias Scrum, May 2024. https://www2.deloitte.com/es/es/pages/technology/articles/ceremonias-scrum.html/.

[Dja24] Django. Django, May 2024. https://www.djangoproject.com/.

[Fig24] Figma. Figma: The collaborative interface design tool, May 2024. https://www.figma.com/.

[Fla24] Flask. Flask Documentation (3.0.x), May 2024. https://flask.palletsprojects.com/en/3.0.x/.

[Get24] GetApp. Kydemy Software, May 2024. https://www.getapp.es/software/115123/kydemy/.

[Ion24] Ionos. Flask vs Django: ¿Cuál es Mejor para tu Proyecto Web en 2024?, May 2024. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/flask-vs-django/.

[Jav24] JavaScript. JavaScript, May 2024. https://es.wikipedia.org/wiki/JavaScript/.

[JSO24] JSON. JSON (JavaScript Object Notation), May 2024. https://www.json.org/json-es.html.

[Kin24] Kinsta. Flask vs Django: ¿Cuál es el Mejor Framework Python en 2024?, May 2024. https://kinsta.com/es/blog/flask-vs-django/.

[Loc24] Locust. Locust: Load Testing Tool, May 2024. https://locust.io/.

[MNp24] MNprogram. Programa de Contabilidad, May 2024. https://www.mnprogram.com/funcionalidades/programa-contabilidad/.

[Pat24] Patterns.dev. Presentational and Container Components in React, May 2024. https://www.patterns.dev/react/presentational-container-pattern/.

[Per24] Natalia Perosio. Scrum y su impacto en el mundo actual, May 2024. https://www.linkedin.com/pulse/scrum-y-su-impacto-en-el-mundo-actual-natalia-perosio/.

[Pyt24a] Pytest. pytest Documentation (8.2.x), May 2024. https://docs.pytest.org/en/8.2.x/.

[Pyt24b] Python. Welcome to Python.org, May 2024. https://www.python.org/.

[Pyt24c] PythonMania. What is Flask in Python? An In-Depth Guide to Web Development Framework, May 2024. https://pythonmania.org/what-is-flask-in-python-an-in-depth-guide-to-web-development-framework/.

[Rad24] Radixweb. React vs Angular: Choose The Best Framework in 2024, May 2024. https://radixweb.com/blog/react-vs-angular#choose/.

[Rea24] React. React: A JavaScript library for building user interfaces, May 2024. https://es.react.dev/.

[SB01] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*. Prentice Hall PTR, 2001.

[Typ24] TypeScript. TypeScript, May 2024. https://www.typescriptlang.org/.

[UML24] UML. Unified Modeling Language (UML), May 2024. https://www.uml.org/.

APÉNDICE A
Objetivos de Desarrollo Sostenible

Reflexión sobre la relación del TFG con los ODS más relacionados ordenándolos por grado de cumplimiento:

- ODS 4. Educación de calidad: el desarrollo de la aplicación web para la escuela de música tiene una alta relación con esta ODS. Se debe a que el hecho de digitalizar las acciones de la escuela como puede ser la de evaluar a los alumnos permitirá una educación de mayor calidad ya que se ofrece una herramienta con la que llevar a cabo funciones de una forma fácil y eficiente. Además los alumnos en un futuro podrán realizar la matrícula o consultar los horarios de una forma mucho más cómoda.
- ODS 8. Trabajo decente y crecimiento económico: al mejorar la calidad administrativa de la escuela, atraerá posiblemente a más alumnos y docentes, lo que se resumirá en un crecimiento económico. Además la posible contratación de técnicos informáticos para el soporte de la aplicación será un punto positivo para esta ODS. Proporcionar a todo el docente una herramienta en la que se proporciona información clara y sencilla a todos los docentes, cumpliendo con el trabajo digno y decente.
- ODS 12. Producción y consumo responsables: la digitalización de las funciones administrativas de la escuela reducirá significativamente el uso de papel y de recursos materiales, promoviendo así una práctica sostenible.
- ODS 5. Igualdad de género: esta ODS está relacionada en menor medida con el proyecto pero aun así se ha tenido en cuenta que no quepa la posibilidad en mostrar ningún tipo de desigualdad por género en ninguna de sus funcionalidades ni interfaces.
- ODS 10. Reducción de las desigualdades: ofrecer la capacidad de realizar la matrícula de forma online hace que personas de cualquier condición física puedan realizar la gestión sin problemas. Además en un futuro, se podrá implementar la funcionalidad de proporcionar acceso a recursos educativos gratuitos.

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
• Fin de la pobreza.				X
• Hambre cero.				X
• Salud y bienestar.				X
• Educación de calidad.	X			
• Igualdad de género.			X	
• Agua limpia y saneamiento.				X
• Energía asequible y no contaminante.				X
• Trabajo decente y crecimiento económico.	X			
• Industria, innovación e infraestructuras.				X
• Reducción de las desigualdades.			X	
• Ciudades y comunidades sostenibles.				X
• Producción y consumo responsables.		X		
• Acción por el clima.				X
• Vida submarina.				X
• Vida de ecosistemas terrestres.				X
• Paz, justicia e instituciones sólidas.				X
• Alianzas para lograr objetivos.				X

Figura A.1: Tabla con las ODS y su grado de relación con el TFG