UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Asistente Web para la extracción y clasificación de exámenes desde ficheros pdf.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fiocca Paglia, Fernando Guillermo G

Tutor/a: Martí Campoy, Antonio

CURSO ACADÉMICO: 2021/2022

Resumen

En este trabajo se ha desarrollado una aplicación web que facilita al profesorado compartir los exámenes realizados por sus alumnos de manera individual y generar una lista con las calificaciones para subir a PADRINO.

El diseño está centrado en el usuario y pretende dar una buena experiencia de uso, sobre todo en el proceso de asociación de los exámenes.

La aplicación cuenta con un sistema de usuarios con la gestión de sus asignaturas y exámenes. También incluye las funcionalidades de cargar un listado de alumnos mediante un lector de archivos CSV, cargar los documentos resultados de escanear las pruebas de evaluación realizadas y descargar toda la información de un examen.

Finalmente, tras la asociación de todos los exámenes de una prueba escrita, se permite la descarga de toda la información. Por un lado, los documentos de los exámenes de los alumnos de manera individual donde se mantienen los comentarios del profesor y, por otro lado, las listas con las calificaciones de cada alumno.

Las principales tecnologías utilizadas son Angular, Express y MySQL. En la implementación también se incluye el uso de comunicación HTTP y TCP/IP y lanzamientos de comandos Linux en nodos hijos controlados por NodeJS.

Palabras clave: UI/UX, Cliente, Servidor, Base de datos, Sockets, Lector CSV, PDF, Comandos Linux, Correo.

Abstract

In this work, a web application has been developed that makes it easier for teachers to individually share the exams taken by students and generate a list with the grades to upload to PADRINO.

The design is user-centered and aims to provide a good user experience, especially in the exam association process.

The application provides management for users, courses and exams. It also includes the functionalities of uploading a list of students through a CSV file reader, uploading the documents resulting from scanning the evaluation tests carried out and downloading all the information of an exam.

Finally, after the association of exams to students, the download of all the information is allowed. On the one hand, the documents of the individual student exam where the teacher's comments are kept and, on the other hand, the lists with the grades of each student.

The main technologies used are Angular, Express and MySQL. Also included in the implementation is the use of HTTP and TCP/IP communication and execution of Linux commands on child nodes controlled by NodeJS.

Keywords: UI/UX, Frontend, Backend, DB, Sockets, CSV reader, PDF, Linux commands, Email.

Tabla de contenidos

Tabla de figuras

1. Introducción

Varios profesores del Departamento de Informática de Sistemas y Computadores de la Universitat Politècnica de València decidieron mejorar el sistema de revisión de los exámenes para los alumnos. La idea era poder compartir el examen del alumno mediante el apartado de "espacio compartido" que tiene PoliformaT[1] para cada asignatura y también obtener un archivo para subir a PADRINO y oficializar las notas. La idea se llevó a cabo desarrollándose diferentes aplicaciones software, utilizando *scripts* o las herramientas ofrecidas por los sistemas operativos.

De este modo cada profesor realiza la tarea de compartir los exámenes del modo o con la herramienta que más le acomoda, aunque ninguno de ellos está completamente satisfecho.

Este trabajo pretende unificar al máximo todos los aspectos positivos de las diferentes herramientas y procedimientos, crear una única aplicación capaz de, no solo facilitar los exámenes de los alumnos en documentos individuales y una lista de notas para poder subir a PADRINO, sino también de dar al profesor la mayor agilidad y una experiencia agradable a la hora de gestionar toda esta información.

1.1. Objetivos

Se pretende crear una aplicación web donde se pueda cargar la lista de alumnos de una asignatura, los documentos PDF con los exámenes escaneados, obtener un PDF de cada examen de cada alumno y una lista con las notas para subir a PADRINO. Además, la aplicación web debe:

- Permitir la gestión de las asignaturas que imparte el profesor y los exámenes que se realizan en cada asignatura.
- Dar una experiencia de usuario satisfactoria y agilidad a la hora de asignar los exámenes calificados a cada alumno.
- Entregar al profesor una lista de documentos con los exámenes de los alumnos de manera individual listos para subir al espacio compartido de PoliformaT.
- Entregar al profesor un archivo CSV con las notas de cada alumno para subir a la aplicación PADRINO.

[1] https://poliformat.upv.es

1.2. Estructura

La memoria de este trabajo fin de grado se estructura de la siguiente manera. El capítulo de "Estado del arte" muestra brevemente las herramientas disponibles y sus principales inconvenientes. En el siguiente capítulo se describe la metodología utilizada para desarrollar la aplicación objeto de este trabajo. En los capítulos de "Análisis y especificación de requisitos" y de "Casos de uso" se muestra el resultado del análisis y la captura de especificaciones, realizada con el tutor y otros profesores del DISCA. El capítulo de "Diseño" especifica y justifica las decisiones de desarrollo, tanto gráficas, como en el tratamiento de datos. La "Implementación y tecnologías" es un capítulo en el que se explican las tecnologías usadas para el desarrollo de la aplicación web, la estructura del proyecto, la manera en la que se implementan las funciones y las bibliotecas para conseguir cumplir todos los requisitos. Seguido están los capítulos de "Pruebas" y "Manuales de instalación y de usuario" donde se encuentran las pruebas realizadas y un manual para instalar la aplicación web, así como la guía para usarla. Finalmente están las "Conclusiones" obtenidas a lo largo del desarrollo.

2. Estado del arte

El problema que se plantea para este trabajo es muy específico: partiendo de un documento PDF que contiene todos los exámenes escaneados de los alumnos, extraer las páginas correspondientes a cada alumno y crear un fichero con el DNI del alumno. Esta asociación examen-alumno se hace, en principio, de forma manual utilizando una herramienta gráfica donde se mostraba la primera página del examen y el profesor seleccionaba de una lista el alumno correspondiente.

Posteriormente algunos profesores desarrollaron alternativas basadas en *scripts* y hojas de cálculo. La clasificación de los exámenes seguía siendo manual, pero permitía la subida de notas a PADRINO de forma más eficiente, e interrumpir el trabajo y reanudarlo posteriormente.

La automatización de la clasificación de los exámenes pasa por el reconocimiento del nombre o DNI manuscrito en cada examen. Existen tecnologías capaces de realizar este tipo de funciones, como, por ejemplo, el lector de la aplicación de traducción de Google o la aplicación ScanMath que es capaz de leer y resolver ecuaciones manuscritas. Sin embargo, estas herramientas tienen coste económico y también es necesario enviar los exámenes y los datos de los alumnos a servidores externos a la UPV, lo que no es conveniente.

Basada en el reconocimiento automático, la última herramienta desarrollada y a la vez la más prometedora es el trabajo final de máster de un alumno del Máster en Ingeniería Informática [1] donde se utilizaba una biblioteca para el reconocimiento del DNI del alumno, y de esta forma clasificar automáticamente los exámenes. En este mismo TFM se puede encontrar una descripción detallada de todas las herramientas disponibles para la clasificación de los exámenes.

2.1. Crítica al estado del arte

Las aplicaciones descritas anteriormente fueron utilizadas, aunque no de forma satisfactoria por varios motivos.

Como se ha comentado anteriormente, las tecnologías que permiten ser usadas para la detección de texto manuscrito tienen un coste monetario lo cual hace que sea una posibilidad descartada.

Usar bibliotecas, código externo o herramientas de terceros produce riesgos con respecto a la seguridad y protección de datos. Los documentos que se desean tratar contienen información personal de los alumnos y no se puede permitir que terceros accedan a esta información.

Las herramientas que se usan actualmente para poder clasificar manualmente los exámenes son lentas y tediosas, con una usabilidad no satisfactoria. Además, el profesorado debe gestionar manualmente los exámenes que corrige, luego entre herramientas, *scripts* y otro tipo de *software* realizar todas las funciones e ir pasando la información a mano de herramienta a herramienta para poder conseguir el objetivo de subir las notas y mostrar a cada alumno su examen.

Finalmente, la última herramienta desarrollada en el marco de un TFM y que incluye el reconocimiento automático del DNI, solo alcanza una tasa de acierto del 50%. La aplicación se centró, principalmente, en esta función, por lo tanto, como no se contemplaba el error en el reconocimiento, realizar la clasificación de los datos a mano no era requisito de la aplicación y es aquí donde no es eficiente, por lo cual se descarta su uso. La poca usabilidad de la aplicación en caso de error fue suficiente para que el profesorado prefiriera usar sus métodos (varias herramientas, *scripts*…) en lugar de esta aplicación.

2.2. Propuesta

La solución a todos los problemas que presentan las herramientas y maneras de proceder para clasificar los exámenes y subir las notas a PADRINO, es recopilar todas las carencias encontradas y desarrollar una aplicación nueva, centrada en el usuario y en el trabajo que debe realizar. Una aplicación capaz de organizar los exámenes, de subir los documentos, asignar de una manera muy cómoda la información del examen a cada alumno y finalmente descargar esta información.

Es decir, una aplicación con gestión de asignaturas y exámenes, asignación de información centrada en la usabilidad, descarga de información para poder subir las notas a PADRINO y finalmente la descarga de exámenes individuales para cada alumno.

3. Metodología

El trabajo que se propone tiene un único desarrollador sin dependencias de trabajos externos, es decir, el desarrollo entero del trabajo se hará en base a un cliente y un desarrollador.

La motivación y propuesta de solución surge de las carencias detectadas en trabajos anteriores, por tanto, adelantándose al análisis del proyecto se sabe con certeza qué funcionalidades no pueden faltar al igual de qué líneas de diseño seguir para hacer el trabajo del usuario más eficiente y cómodo. A falta del análisis, la experiencia del cliente o en este caso del profesor tutor, da un claro camino a seguir una metodología de desarrollo clásico o en cascada.

Por tanto, el desarrollo se guiará por las fases definidas por el modelo clásico (análisis de/requisitos, diseño, implementación, verificación, y despliegue y mantenimiento) [2].

Como pasa en la gran mayoría de campos, del concepto teórico al práctico tiende a haber ciertas modificaciones. El desarrollo del software lleva tiempo y es común que aparezcan variaciones con el tiempo, ya sea por petición del cliente o por contratiempos que pueden surgir (bibliotecas obsoletas, cambio de diseño…). El modelo de desarrollo en cascada más simple salta de una fase a la siguiente hasta llegar a la fase final como se ve en la Figura 1. Sin embargo, es muy común ver estos esquemas con comprobaciones siempre que se salta de fase, sobre todo para validar el trabajo hecho antes de continuar con la siguiente fase (Figura 2). En este caso se prevén validaciones en cada fase antes de continuar con la siguiente, con el añadido de constantes comprobaciones y confirmaciones en los apartados de diseño gráfico y procedimientos de asociación de examen-alumno. De esta manera, se asegura al máximo la usabilidad para el usuario, evitando así esa experiencia no satisfactoria de los anteriores trabajos.

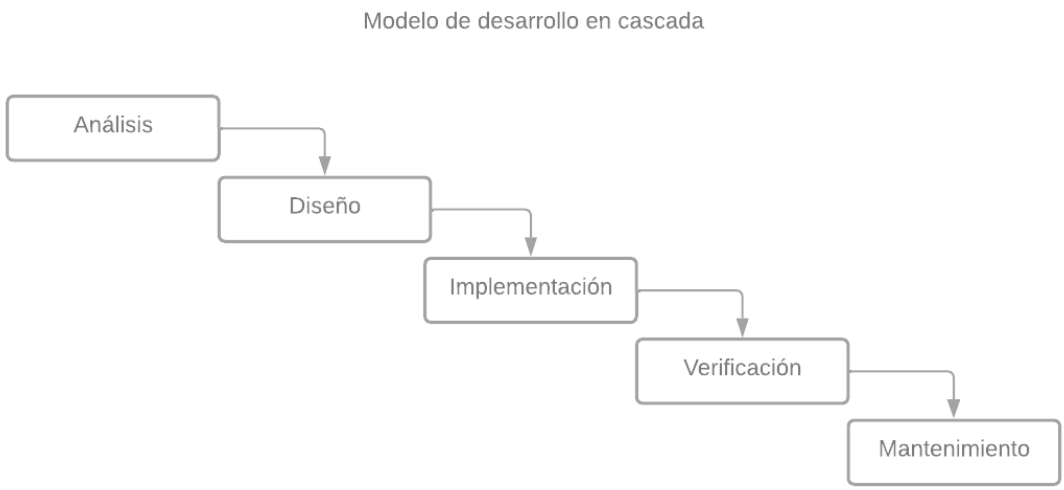


Figura 1 Modelo de desarrollo en cascada.

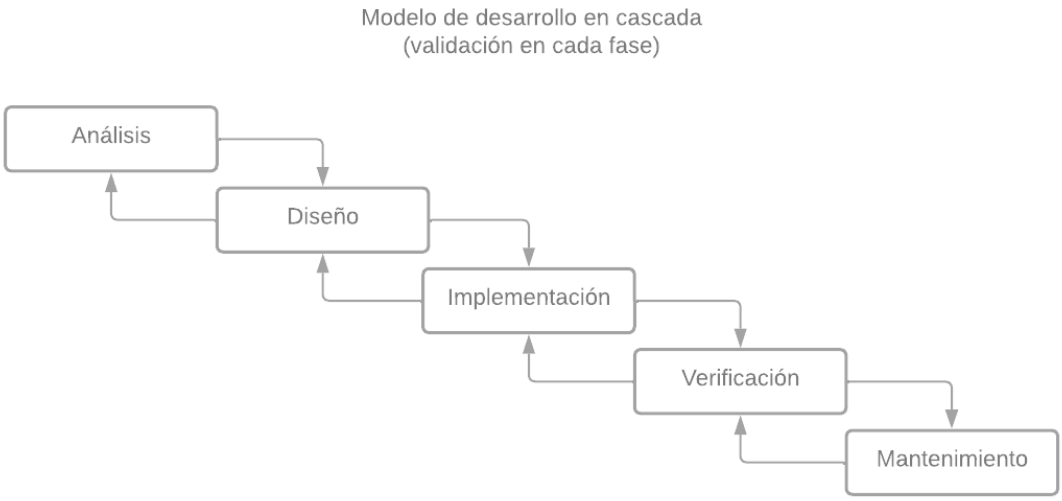


Figura 2 Modelo de desarrollo en cascada con validación al cambio de fase.

4. Análisis y especificación de requisitos

El punto de que se parte para el desarrollo de esta aplicación es de la necesidad de combinar varias herramientas obteniendo una única solución a un problema dado. El objetivo principal es poder distribuir a los alumnos su examen corregido. Además, facilitar la introducción de las notas en la aplicación PADRINO, utilizada en la UPV para la gestión de notas.

4.1. Análisis del proceso manual

El procedimiento que se pretende mejorar depende de cada profesor y cómo haya encontrado una solución que le parezca cómoda. En este caso, se analiza el procedimiento de un profesor concreto. Sin embargo, la metodología que emplea también es fruto de la recopilación de los diferentes métodos usados por sus compañeros, esto significa que el usuario a analizar tiene la función de portavoz de un grupo de profesores y a partir de este análisis principal se irán estudiando las funciones necesarias para una aplicación definitiva que cumpla con el objetivo del trabajo.

El procedimiento analizado consiste en varios pasos.

4.1.1. Escaneo de exámenes y corrección

El primer paso es escanear los exámenes que los alumnos entregan en papel. Para esto se requiere de una herramienta hardware, un escáner que realiza su función de forma satisfactoria. Aun así, se debe puntualizar que la herramienta para escanear puede no permitir hacer una digitalización de todos los exámenes de una vez, por tanto, hay que tener en cuenta la posibilidad de tener los exámenes en más de un archivo.

Una vez escaneados los exámenes, el profesor los corrige utilizando el ordenador o una tableta.

También es posible invertir los dos pasos anteriores, es decir, corregir los exámenes en papel y luego escanearlos. El resultado es el mismo.

4.1.2. Asignación de nota y examen al alumno.

Para asignar las calificaciones y las hojas del PDF que corresponden a un alumno concreto, el profesor abre el archivo o los archivos PDF con los exámenes y ajusta la ventana para que

ocupe la media pantalla izquierda. Por otro lado, abre una hoja de cálculo con la lista de los alumnos y sus números identificativos. Esta ventana ocupa la mitad derecha de la pantalla. Con las dos ventanas ocupando la pantalla, el profesor busca la hoja del examen donde se encuentra el nombre del alumno (la primera hoja del examen del alumno), con el nombre localizado busca el alumno en la hoja de cálculo con la lista de alumnos. En caso de que el nombre no sea de fácil lectura, el profesor aprovecha el visor para hacer zoom y poder leer el nombre. Con el nombre encontrado en la lista de alumnos, se le asigna la nota en una columna y la página donde empieza su examen en otra. Luego busca la página donde acaba el examen del alumno y la pone en otra columna. Hay veces que esto último no debería ser necesario, ya que el número de hojas de un examen puede ser siempre el mismo para todos los alumnos. El resultado de este proceso se puede ver en la Figura 3.

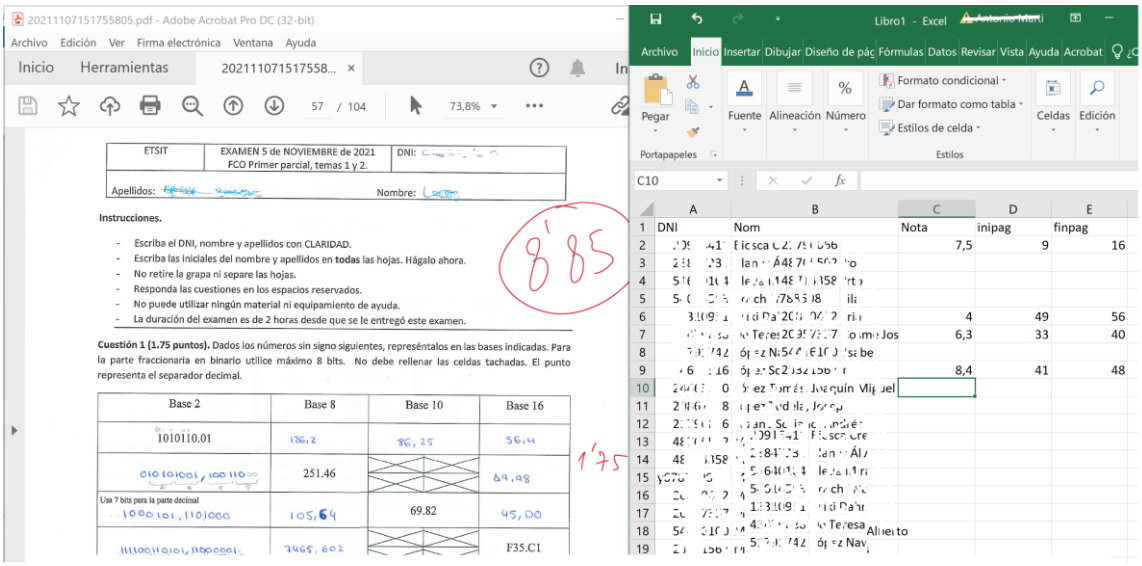


Figura 3 Proceso de asociación manual.

Se pudo observar que el trabajo realizado por el profesorado es costoso a nivel temporal. Por tanto, no es extraño dejar una corrección a medias y continuarla posteriormente, una posibilidad que no ofrecían otras herramientas disponibles.

4.1.3. Publicar notas y compartir exámenes

Con la lista de alumnos completa con la información necesaria, se sube a PADRINO para hacer las notas oficiales. Por otro lado, el archivo CSV se recorre con un script que divide los ficheros PDF en archivos individuales por cada alumno para, posteriormente, con un programa Java (Platero, Figura 4), se suben estos archivos al espacio compartido de PoliformaT correspondiente de cada alumno.

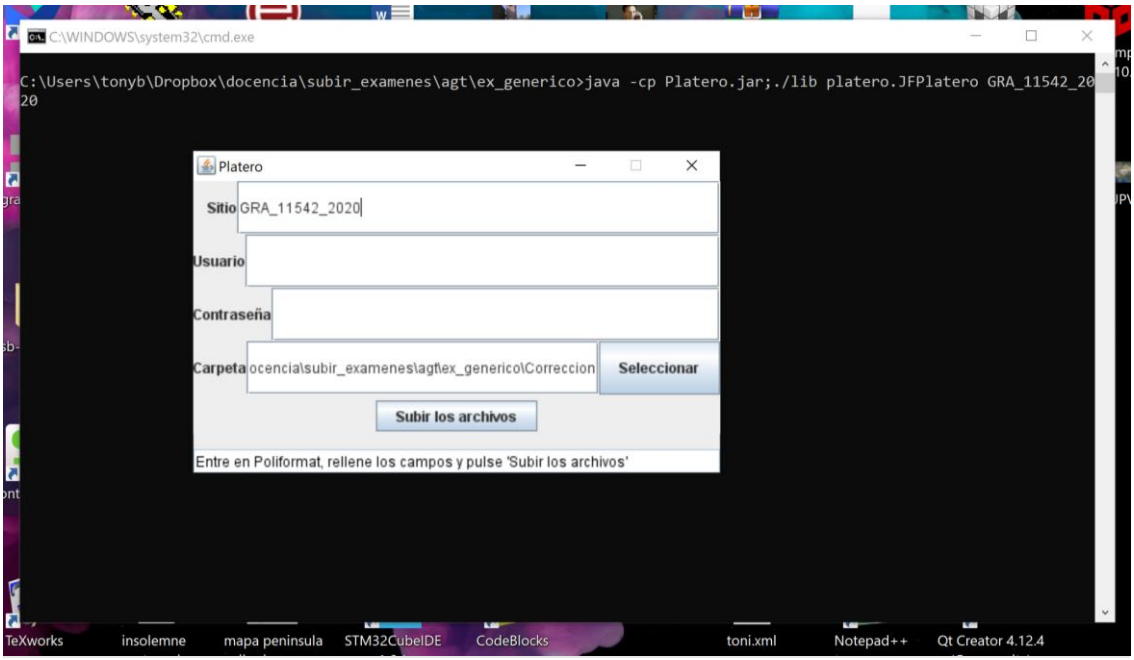


Figura 4 Interfaz gráfica de Platero.

4.2. Arquitectura y despliegue

Las herramientas utilizadas para asignar las calificaciones, como se puede ver, son herramientas de escritorio. No hay que olvidar, que además se hace referencia a varias herramientas. Si se pone como ejemplo la situación de que el profesor decida hacer un borrado completo del ordenador por limpieza, recuperar estas herramientas puede ser tedioso.

El uso de aplicaciones de escritorio no es algo incómodo, por tanto, el primer paso es reducir el número de herramientas a una. Con esto planteado, se da el siguiente paso, la aplicación de escritorio no es una mala arquitectura y para una herramienta como la que se plantea puede ser perfectamente válida, pero presenta la desventaja de no ser multiplataforma. Para solucionar esto lo mejor es usar una tecnología compatible con todos los sistemas operativos como, por ejemplo, los navegadores web. Resumiendo, se precisa una aplicación de uso individual, multiplataforma y de fácil acceso. Una aplicación web sería lo más indicado. Una aplicación donde los datos dependen únicamente del usuario y donde el servicio esté activo sin la necesidad de que el usuario se esté preocupando de su gestión y accesibilidad. Además, ofrecería la ventaja de que la disponibilidad y la seguridad se podría delegar a los servidores de la universidad, dejando así una aplicación para el profesorado donde únicamente actúan como usuarios estándar.

4.3. Usuarios, persistencia y seguridad

Como el planteamiento de la arquitectura se basa en una aplicación web, debe existir la creación de usuarios que pueden acceder a la aplicación y a sus datos, de manera que la información permanezca guardada y visible solamente para el usuario que la crea.

La existencia de usuarios requiere de gestión para estos, pero también resuelve el problema de la persistencia, manteniendo la información y los datos de los usuarios almacenados en el servidor. La necesidad de tener usuarios requiere cierta seguridad, ya que los datos deben ser privados y cifrados. La contraseña debe ser cifrada y una buena práctica sería cifrar el correo electrónico asociado, un correo que se usaría en caso de que se necesiten confirmaciones o enviar avisos.

4.4. Organización y gestión

En las herramientas utilizadas anteriormente no existe una organización de archivos. El modo de mantener un lugar de trabajo ordenado depende de cada profesor, pero todos comparten la misma metodología, el uso del árbol de carpetas. La Figura 5 muestra un ejemplo de cómo se organiza un profesor mediante el uso de carpetas. Por tanto, una forma de organizar los archivos que se puede implementar fácilmente es creando un sistema de dependencia en torno al usuario, donde un acto de evaluación[2] depende de una asignatura y una asignatura (que pertenece a un curso) que dependa de un usuario.

Aparecen los conceptos asignaturas y actos de evaluación. A nivel de programación podemos llamarlos objetos. Objetos que dependen de forma jerárquica hasta llegar al usuario, de manera que la información es privada y visible solo para el profesor. Estos objetos también deben tener la capacidad de ser gestionados, es decir, poder crearlos, editarlos y/o borrarlos. No es necesario y mucho menos en todos los casos dar todas las opciones, pero sí es un buen planteamiento si se piensa en la experiencia de usuario, poder darle el máximo control sobre los datos al usuario. Las asignaturas deberán tener su información básica, el nombre, un acrónimo, el curso de la asignatura y para el objetivo de la aplicación, la lista de alumnos, de esta manera, cada asignatura tendrá su propio listado y cuando se quiera asignar la nota de un examen, no habrá que volver a cargar la lista de alumnos, dado que se usará la de la asignatura. Para las pruebas de evaluación es más simple, es suficiente con un nombre y los exámenes PDF, la lista de alumnos se leerá de la asignatura. Se debe tener en cuenta que estos exámenes pueden estar en más de un archivo y que la lista de alumnos no es modificada, esta lista se lee y la asignación de información se hará en una lista en paralelo que se genera con la prueba de evaluación.

[2] Se reserva la palabra examen para las hojas correspondientes a las respuestas de un alumno, mientras acto de evaluación, comprende el total de exámenes realizados por los alumnos.

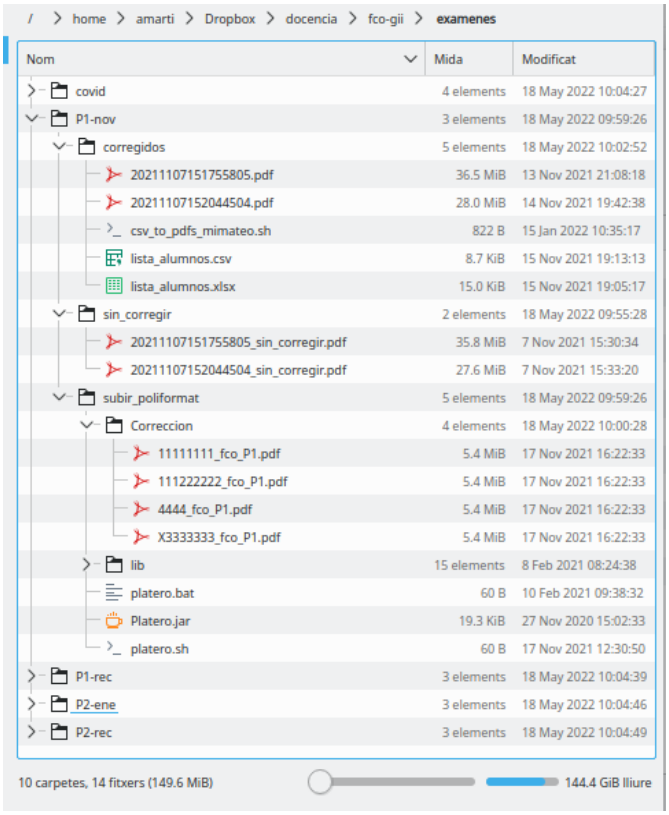


Figura 5 Organización de carpetas de un profesor.

Por último, pensando a futuro y la ventaja de poder servir persistencia a los datos, se debe plantear una forma de organizar la información dentro de la aplicación. Las asignaturas y actos de evaluación deberían estar separadas por cursos, mostrando en primer plano el actual. Para cada curso, mostrar las asignaturas y exámenes. Además, sería bueno usar elementos identificativos como los acrónimos de las asignaturas o un código de color para una identificación más cómoda. En un curso no existen muchas asignaturas, un profesor puede participar, como mucho, en unas 4 o 5 asignaturas, pero en los actos de evaluación, la información crece bastante. Hay que calcular que cada asignatura puede tener una media de 4 actos de evaluación, que deberían tener la posibilidad de ser ordenados, ya sea por nombre o por otro campo de información. También sería conveniente poder mostrar los exámenes de una única asignatura. Por ejemplo, un profesor con 4 asignaturas, cada asignatura con dos parciales y un examen de laboratorio, todos con sus recuperaciones, puede llegar a tener 24 exámenes en un curso. Filtrar esto es necesario. Como función final, se debería dejar mostrar los cursos que interesen, ya que como se ha visto, la información en la pantalla puede crecer muchísimo.

Tras el análisis, se elabora el diagrama de la Figura 6 que representa los cimientos de la estructura de datos que debería tener la aplicación.

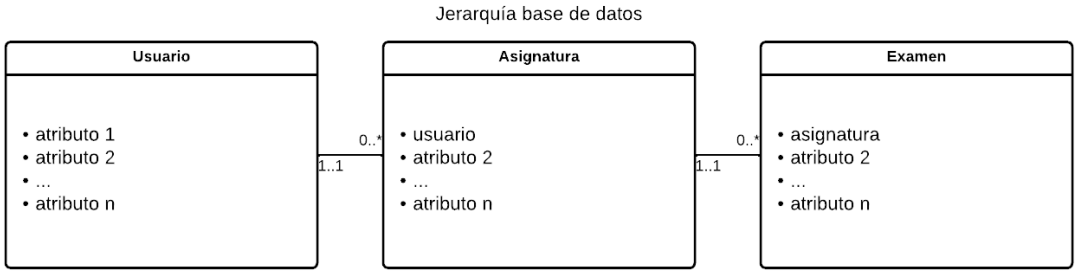


Figura 6 Diagrama UML de la jerarquía de la base de los datos.

4.5. Importación de alumnos

La lista de alumnos puede descargarse desde diferentes aplicaciones que ofrece la propia universidad. Esto significa que los formatos de los archivos que se descargan con las distintas herramientas no tienen por qué coincidir. El carácter separador de columnas puede ser diferente, al igual que si lleva o no cabecera, o el tipo de codificación en el que está almacenado el archivo. Se muestra un ejemplo de dos listas de tipo CSV generadas por dos herramientas diferentes en la Figura 7.

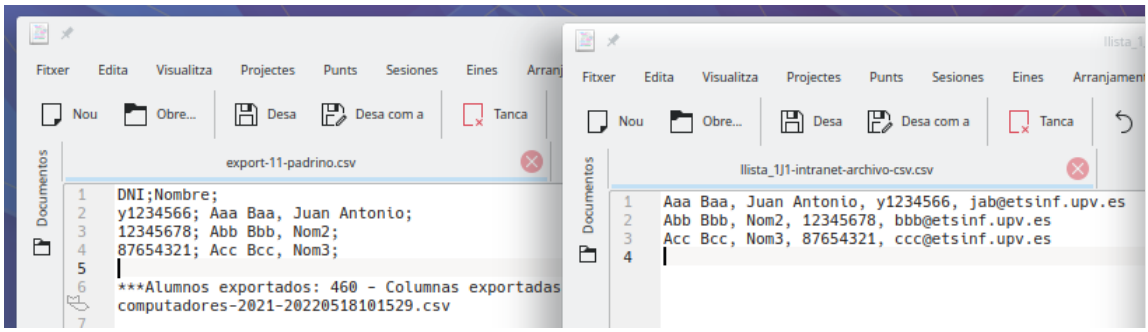


Figura 7 Lista de alumnos con diferentes formatos en archivos de tipo CSV.

Toda la información que necesita la aplicación deberá estar en el servidor, incluyendo la lista de alumnos. El problema para guardar la lista de alumnos es el uso de archivos CSV con diferentes formatos. No es adecuado pedir el uso de un archivo extraído de una herramienta concreta, y además esta podría cambiar su formato en un futuro. La solución y a la cual se dio especial importancia es permitir al usuario especificar el formato del archivo e incluyendo su codificación. Son varias las características de un archivo CSV que deben tenerse en cuenta antes de extraer la información que contiene la lista:

- Si tiene o no cabecera.
- Cuál es el separado de columnas.
- La codificación. En estos momentos conviven diferentes esquemas de codificación, como el Latin 1 y el UTF-8. En un idioma donde los nombres y apellidos suelen tener acentos, es fundamental utilizar la codificación correcta.

- El orden de las columnas donde se encuentra la información. Para identificar a un alumno no se necesita más que el número identificativo (DNI o NIE), el nombre y sus apellidos. Sin embargo, después de analizar varias listas de alumnos, la información puede estar presentada con formatos totalmente diferentes, como los apellidos separados por columnas, o el nombre y apellidos en la misma columna.

En el momento de importar a la aplicación la lista de alumnos es necesario que el usuario pueda indicar todas las características del fichero CSV.

4.6. Asociación de examen y su proceso

El proceso manual para clasificar los exámenes y asignar las notas a los alumnos está definido por lo profesores. Este proceso no se va a intentar cambiar, el usuario lleva años funcionando de la misma manera y un cambio podría resultar lo suficientemente incómodo como para decidir no usar la aplicación. En su lugar, se partirá de la base del proceso manual y se automatizará la metodología.

Primero, se debe mantener la disposición de las ventanas. Si el profesor ha encontrado la posición más cómoda para cada ventana, se debería mantener, por eso, el visor de los exámenes se mantiene a la izquierda y la lista de alumnos a la derecha. Por supuesto, el visor tendría que replicar los aspectos básicos del lector de archivos PDF, es decir, una barra de navegación para poder desplazarse por los exámenes.

En la derecha estaría la lista de alumnos. En este caso, el uso de una lista es innecesario ya que se sustituye por un campo de texto donde el usuario pueda indicar el nombre del alumno. Este campo será autocompletable, es decir, según se introducen las letras del apellido, nombre o incluso los dígitos del DNI del alumno, se muestran los alumnos que coincidan con esta información introducida en una lista desplegada desde el campo de texto. Con esto se ahorra la búsqueda de cada alumno en la lista. Otro campo permitirá al usuario introducir la nota. Por último, dos campos indican las páginas del examen del alumno. Estas páginas no siempre se seleccionan manualmente si se presenta un diseño eficaz para esta función. Aunque se suprima la lista de alumnos de la pantalla, no es mala idea mantener una tabla donde se guarden los alumnos con sus datos asociados. Esta tabla contendría la información del alumno y se podría aprovechar para dar la opción de editarla. Esto último es requisito del profesorado, ya que en algunas de las herramientas anteriores no podían volver atrás o editar un alumno una vez asignado un examen. El problema ocurría cuando el profesor se equivocaba y asignaba la información a otro alumno. Cuando se daba cuenta del error, no podía corregirlo y debía apuntárselo para más tarde cambiarlo manualmente. Con esta tabla, no solo podría ver la información a tiempo real que se está guardando, sino que le daría la opción de editarla.

Un prototipo que se tomará como base para el diseño de la interfaz se muestra en la Figura 8, donde se posicionan los componentes que debería tener la pantalla según el análisis hecho.

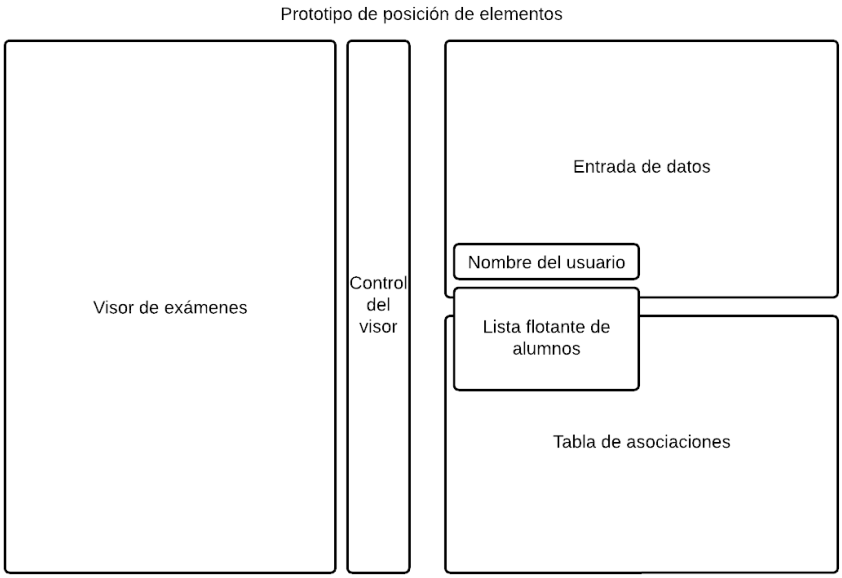


Figura 8 Prototipo de disposición de elementos basado únicamente en el análisis.

El resultado debe ser una acción simplificada a cuatro pasos. Localizar el nombre del alumno en la página inicial del examen gracias al visor dispuesto a la izquierda, escribir su nombre con ayuda de autocompletado, poner la nota del alumno y seleccionar las páginas del examen. Esto último se puede reducir en seleccionar la página final ya que la inicial estaría localizada, aun así, también hay que dar la posibilidad de no tener que seleccionar la página final en caso de que todos los exámenes cuenten con el mismo número de hojas.

4.7. Visor de los exámenes

Una primera aproximación sería integrar o replicar un visor de documentos PDF e implementarlo en la aplicación. Sin embargo, al no ser un objetivo del trabajo, la replicación de una herramienta convencional de lectura de documentos PDF se simplifica a lo imprescindible para poder completar el procedimiento de asociación de información. Un profesor, a la hora de hacer las asignaciones alumno-examen, de un lector de documentos no necesita más que avanzar o retroceder las páginas y la posibilidad de ampliar el documento en caso de que la identificación del alumno sea difícil de leer.

La identificación de un examen se hace por dos tipos de datos, o el nombre del alumno, o su DNI. Ambos siendo texto manuscrito. El problema que se ha podido observar es el de un texto ilegible o con muy mala caligrafía. En ese caso el profesor debe detener el procedimiento para hacer zoom con la herramienta para visualizar el PDF y leer más cómodamente los datos del alumno. Además, este escenario es tan frecuente que se especificó como un requisito. Uno de los objetivos de este trabajo es automatizar el procedimiento y evitar pasos innecesarios, por lo que se propone una función de lupa, diferente de un zoom convencional. La propuesta basada en el análisis del uso del zoom es dar la posibilidad de amplificar una zona concreta del examen, y mantener esa zona amplificada en todas las páginas. Es decir, como prototipo, un rectángulo

amplificado donde se pueda ver el nombre del alumno y que pueda posicionar el profesorado dentro de la hoja del examen. Es conveniente que este rectángulo se mantenga para todas las hojas, de manera que sea una función de zoom lo más eficiente para este escenario.

4.8. Resultado y formato de la información

El resultado final de la aplicación debe ser un archivo CSV y una carpeta con documentos. El objetivo es facilitar todo un proceso en el que se asocian alumnos y exámenes. Sin embargo, este proceso se hace para tener la información digitalizada. De esta manera, se puede obtener un archivo el cual se puede subir a la herramienta PADRINO y hacer oficiales las notas de los alumnos, además de obtener los exámenes en documentos individuales para poderlos compartir con los correspondientes alumnos.

Tras varias propuestas para obtener esta información, el profesor prefirió la opción de realizar una única descarga, es decir, descargar un único archivo comprimido con toda la información del acto de evaluación. Una descarga que debería estar siempre activa ya sea por dejar un trabajo a medias o porque el profesor quiera corregir algo.

La estructura que sigue este fichero comprimido está especificada tanto por el profesor, como por las herramientas externas que se usarán (PADRINO y Platero). Toda la información debe estar dentro de una carpeta con el nombre del acto de evaluación. Esta carpeta contendrá otras dos llamadas “csv” y “examenes”, de manera que se identifica fácilmente qué contiene cada una. Una contendrá dos archivos de tipo CSV, uno de los cuales se usará para que PADRINO lo lea y suba las notas, y otro con toda la información (identificación del alumno, nota y números de página de inicio y de fin del examen en el PDF. Este fichero no tiene un uso específico, este archivo es a petición del profesorado. La siguiente carpeta almacena todos los exámenes de los alumnos en un archivo PDF por cada alumno.

La Figura 9 es un esquema de cómo debe estar organizada la información dentro del archivo comprimido.

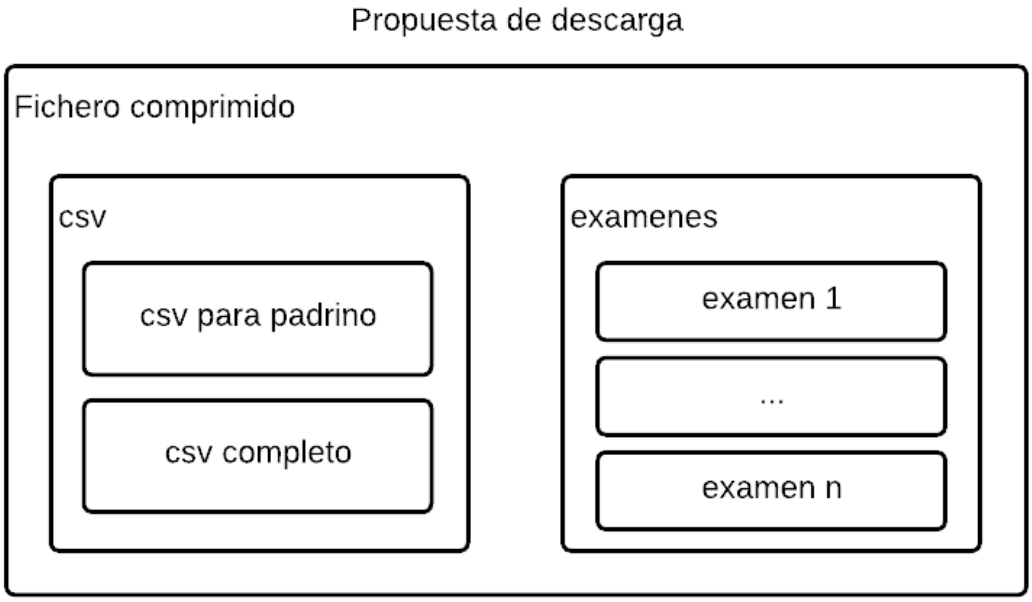


Figura 9 Propuesta de la organización del fichero comprimido para descargar.

4.8.1. Carpeta “csv”

Este directorio debe contener un archivo CSV que se pueda subir a la herramienta PADRINO para poder publicar las notas de los alumnos. Para generar el archivo correcto para PADRINO debe contener al menos dos columnas: el número de identidad del alumno y su nota. Estas columnas están separadas por el carácter propio de los archivos CSV, el punto y coma. PADRINO acepta como separador de decimales tanto la coma como el punto. El archivo, además debe tener las cabeceras de las columnas. El resultado debe ser un archivo con el nombre del acto de evaluación y a continuación, las palabras “notas” y “padrino”, donde los espacios se sustituyen por el carácter de guion bajo para así evitar problemas con el sistema de archivos.

Finalmente, el segundo archivo, contiene los detalles de los alumnos. En el proceso de análisis, el profesor señaló que podría resultar útil otro archivo donde apareciera toda la información del alumno, tanto su nombre completo, como su nota y páginas de examen. Una situación que se puso como ejemplo, fue a la hora de las revisiones de los exámenes, en caso de que el profesor tuviera que identificar a un alumno, tendría una lista con los nombres y las notas, o en caso de que el examen compartido no fuera el del alumno, se podría comprobar por qué no es el suyo.

4.8.2. Carpeta “examenes”

El acceso al espacio compartido de cada alumno está restringido. Se requiere de credenciales y pasar mecanismos de seguridad de la universidad. El profesorado hace uso de Platero para poder compartir los exámenes, por tanto, simplemente se facilitará el uso de este programa. Para esto, la carpeta debe contener los documentos con los exámenes de los alumnos. Estos documentos tienen como nombre, la concatenación del número de identidad del alumno, el nombre del examen y el nombre del alumno para que puedan ser leídos tanto por la aplicación Platero como por el profesor. Como pasa en el caso del archivo CSV, los espacios en blanco son remplazados por guion bajo, además los caracteres como los acentos, son sustituidos a su forma básica, de manera que se evitan problemas de lectura con caracteres especiales.

4.9. Requisitos funcionales

Tras al análisis realizado anteriormente se procede a exponer todos aquellos requisitos que deberá cumplir la aplicación para alcanzar su objetivo. Se presentan estos requisitos agrupados en cuatro grandes campos.

4.9.1. Usuarios y navegación

Identificador	1
Requisito	Registrar usuario
Descripción	El usuario podrá introducir un nombre identificativo único, un correo electrónico y una contraseña. El usuario quedará registrado en la base de datos con el correo y la contraseña cifrada. Posteriormente se le redireccionará para iniciar sesión.
Prioridad	Alta

Identificador	2
Requisito	Iniciar sesión
Descripción	Un usuario registrado podrá iniciar sesión en la aplicación introduciendo su nombre identificativo y su contraseña asociada para validar el usuario y acceder a la aplicación.
Prioridad	Alta

Identificador	3
Requisito	Cambiar contraseña
Descripción	Se debe permitir cambiar la contraseña del usuario mediante el uso del correo electrónico.
Prioridad	Media

Identificador	4
Requisito	Cerrar sesión
Descripción	Cerrar la sesión del usuario y evitar el acceso a esta. Envía al usuario a la

	pantalla de iniciar sesión.
Prioridad	Baja

Identificador	5
Requisito	Retroceso
Descripción	Regresar a la página anterior en las páginas que sean descendientes de otra.
Prioridad	Baja

Identificador	6
Requisito	Cambiar correo
Descripción	Dentro de la aplicación debe permitirse cambiar el correo del usuario.
Prioridad	Baja

4.9.2. Gestión

Identificador	7
Requisito	Crear nuevo curso
Descripción	Muestra una nueva sección para el curso siguiente al último creado.
Prioridad	Alta

Identificador	8
Requisito	Crear una asignatura
Descripción	Mostrar una ventana con la información necesaria para registrar una nueva asignatura. La asignatura debe tener: nombre, acrónimo, curso y color identificativo. Además, debe permitir cargar un archivo CSV con el listado de alumnos matriculados.
Prioridad	Alta

Identificador	9
Requisito	Lectura de archivos CSV
Descripción	Lector de archivos de tipo CSV. El usuario debe poder seleccionar la codificación del archivo, el separador de columnas y si contiene cabecera. La selección del formato sería conveniente que tuviera persistencia. Debe poder elegir las columnas que contengan información útil para la aplicación (número de identidad, nombre y apellidos). Además, debe mostrar el texto en plano y la tabla con la lista leída tras aplicar las opciones seleccionadas por el usuario.
Prioridad	Alta

Identificador	10
Requisito	Editar una asignatura
Descripción	Se permite editar todos los datos de la asignatura incluyendo la lista de alumnos, a excepción del curso.
Prioridad	Alta

Identificador	11

Requisito	Borrar asignatura
Descripción	Mostrar confirmación para el borrado de la asignatura. Para poder borrar la asignatura, esta no debe contener exámenes. Tras la confirmación, debe borrar toda la información asociada.
Prioridad	Media

Identificador	12
Requisito	Crear un examen
Descripción	Mostrar una ventana con la información necesaria para registrar un nuevo examen. El examen debe tener: nombre, fecha y el o los archivos PDF con los exámenes escaneados. Debe permitirse cargar uno o más archivos y poder ordenarlos conforme el usuario quiera.
Prioridad	Alta

Identificador	13
Requisito	Acceder a un examen
Descripción	Tras el procesado y guardado correcto de un examen, debe mostrarse en el listado de exámenes de un curso, dando la opción de acceder a este.
Prioridad	Media

Identificador	14
Requisito	Borrar examen
Descripción	Mostrar confirmación para el borrado del examen. Tras la confirmación, debe borrar toda la información asociada.
Prioridad	Media

4.9.3. Organización

Identificador	15
Requisito	Ordenar exámenes por fecha
Descripción	Ordenar los exámenes por fecha de manera ascendente o descendente de todos los exámenes manteniendo la separación por cursos.
Prioridad	Baja

Identificador	16
Requisito	Ordenar exámenes por nombre
Descripción	Ordena los exámenes alfabéticamente ascendente o descendentemente de todos los exámenes manteniendo la separación por cursos.
Prioridad	Baja

Identificador	17
Requisito	Mostrar u ocultar un curso
Descripción	Esconder o mostrar toda la información de un curso.
Prioridad	Baja

Identificador	18
Requisito	Mostrar u ocultar los exámenes de una asignatura
Descripción	Esconder o mostrar todos los exámenes de una asignatura.
Prioridad	Baja

4.9.4. Asociación de exámenes y descarga

Identificador	19
Requisito	Visor de exámenes con control manual
Descripción	Mostrar las páginas de los exámenes a la izquierda de la pantalla. Este visor debe contar con las opciones de avanzar o retroceder página. Además, de poder indicar manualmente las páginas de inicio y fin del examen de un alumno.
Prioridad	Alta

Identificador	20
Requisito	Zoom en el visor
Descripción	El visor de páginas debe tener la capacidad de mostrar un rectángulo con la zona de la página aumentada. Este rectángulo sirve para ampliar el nombre del alumno y se debe poder posicionar donde el usuario desee. Debe mantenerse a pesar del cambio de página.
Prioridad	Alta

Identificador	21
Requisito	Examen con el mismo número de páginas
Descripción	Se debe poder indicar si los exámenes cuentan todos con el mismo número de páginas, y cuál es este número.
Prioridad	Alta

Identificador	22
Requisito	Selección del alumno
Descripción	Escribir el número de identidad o el nombre del alumno en un campo de texto. Este campo debe contar con una lista de posibilidades que se filtre mediante la escritura y permita la elección de un alumno para evitar escribir el nombre completo.
Prioridad	Alta

Identificador	23
Requisito	Nota del alumno
Descripción	La nota del alumno puede escribirse con decimales, y separar la parte decimal con punto o coma para facilitar el trabajo al profesorado.
Prioridad	Media

Identificador	24
Requisito	Selección de página final del examen
Descripción	Dado que la página de inicio de un examen coincide con la siguiente a la página final del anterior, solo se pide indicar las páginas finales. La página final no

	será necesaria indicarla en caso de que esté activo el campo que define el número constante de páginas de los exámenes.
Prioridad	Alta

Identificador	25
Requisito	Guardar asociación
Descripción	Guardar alumno con su examen. Se permite no tener nombre o nota. El alumno, una vez guardado se mostrará en una tabla. Posteriormente se debe preparar la aplicación para seleccionar el siguiente alumno.
Prioridad	Alta

Identificador	26
Requisito	Proceso por teclado
Descripción	El proceso de guardar alumno debe constar de cinco pasos: localizar el nombre en el visor, seleccionar nombre, nota, página final (si es necesario) y guardar. Para automatizar el proceso se debe permitir realizarlo solo con controles de teclado. Por tanto, el proceso debe ser: leer el nombre en el examen, escribir un par de letras del nombre y con las flechas y el "intro" seleccionar el nombre en una lista desplegable, tabular y escribir la nota, tabular y con las flechas o el "intro" desplazarse hasta la página final (si fuera necesario), tabular y pulsar "intro" para guardar. Después se limpian los campos, se muestra la siguiente página indicándose como página inicial (y en caso de que esté activo, indicando también la página final) y finalmente, el foco debe situarse en el campo de texto de búsqueda del nombre del alumno.
Prioridad	Alta

Identificador	27
Requisito	Uso del tabulador
Descripción	El uso del tabulador entre campos de la asociación. El tabulador debe asignar su foco de manera circular (cuando llegue al final, vuelve al principio) siendo el recorrido el siguiente: texto del nombre, valor de la nota, selección de página final y guardar.
Prioridad	Alta

Identificador	28
Requisito	Tabla de alumnos
Descripción	Mostrar una tabla con los alumnos guardados (se ha realizado la asignación alumno - examen). La tabla muestra la información completa del alumno, además de las páginas de inicio y final, y la nota. Debe dar la opción de editar un alumno. Esta tabla puede ordenarse por el campo que se desee.
Prioridad	Alta

Identificador	29
Requisito	Editar alumno
Descripción	Recuperar y poner la información de un alumno en los campos del formulario. Una vez modificados, cuando se guarde, la aplicación continuará el proceso de asociación por donde se había dejado.
Prioridad	Alta

Identificador	30
Requisito	Persistencia para cada alumno
Descripción	Cada alumno asociado y mostrado en la tabla debe estar guardado en la base de datos. De esta manera, el trabajo se puede dejar y continuar posteriormente. Cuando se acceda a un examen sin completar, el visor se posicionará en la página correspondiente para continuar el trabajo.
Prioridad	Alta

Identificador	31
Requisito	Descarga de información
Descripción	Se debe poder descargar la información en todo momento en un archivo comprimido. El archivo debe tener una carpeta principal con el nombre del examen. Dentro de esta debe haber dos carpetas: “csv” y “examenes”. La primera tiene dos archivos CSV, uno con el nombre del examen (los espacios en blanco sustituidos por guion bajo) seguido de “_notas_padrino.csv” y otro “detalles.csv”. El primer archivo tiene dos columnas con cabeceras “ID estudiante” y “Nota” separados por punto y coma. El segundo, con cabeceras correspondientes a la información del alumno (separado por punto y coma): ID, apellidos, nombre, nota, página inicio y final. La segunda carpeta contiene los ficheros PDF con los exámenes individuales de cada alumno. El nombre de estos documentos debe ser el número de identidad del alumno seguido por el nombre del examen y finalmente por el nombre del alumno. Todo esto separado por guion bajo.
Prioridad	Alta

5. Casos de uso

Tras el análisis de requisitos de la aplicación, se estudia la forma de proceder del profesor. Es decir, a continuación, se estudian los escenarios donde el usuario se puede llegar a encontrar a la hora de usar la aplicación web. Tras esto, se generan los diagramas de casos de uso que envuelve la aplicación, así como la descripción de estos [3].

5.1. Diagrama de contexto e inicial

El desarrollo está enfocado a un único contexto, el uso de la aplicación web. Se puede plantear el uso de herramientas externas como el escaneado de los exámenes o el uso del programa Platero, pero esos escenarios son ajenos al desarrollo y no afectan a este. La aplicación debe funcionar de manera independiente. Por tanto, solo se estudia un contexto, el escenario donde el profesor usa la aplicación web.

Como se puede ver en la Figura 10, solo hay un actor, el profesor. La aplicación está enfocada a este actor, es decir, todos los casos de uso dependen de un único usuario. Estos casos se pueden agrupar en dos tipos: los relacionados con el usuario y los de gestión dentro de la aplicación. Por un lado, la aplicación dispone de una pequeña gestión en torno a la sesión, ya sea para darse de alta o para cambiar el correo introducido, entre otras. Por otro lado, están todos los casos que permiten al actor gestionar su información y manipularla con una amplia libertad. Por tanto, el diagrama inicial de la aplicación quedaría como muestra la Figura 11.

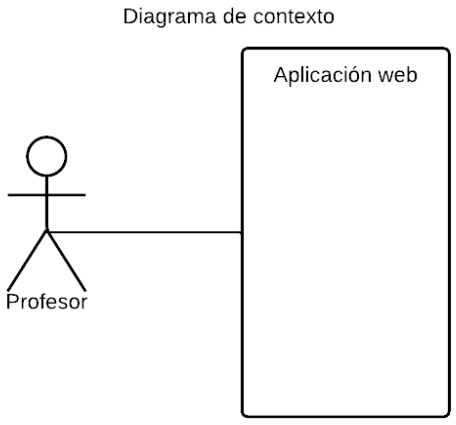


Figura 10 Diagrama de contexto.

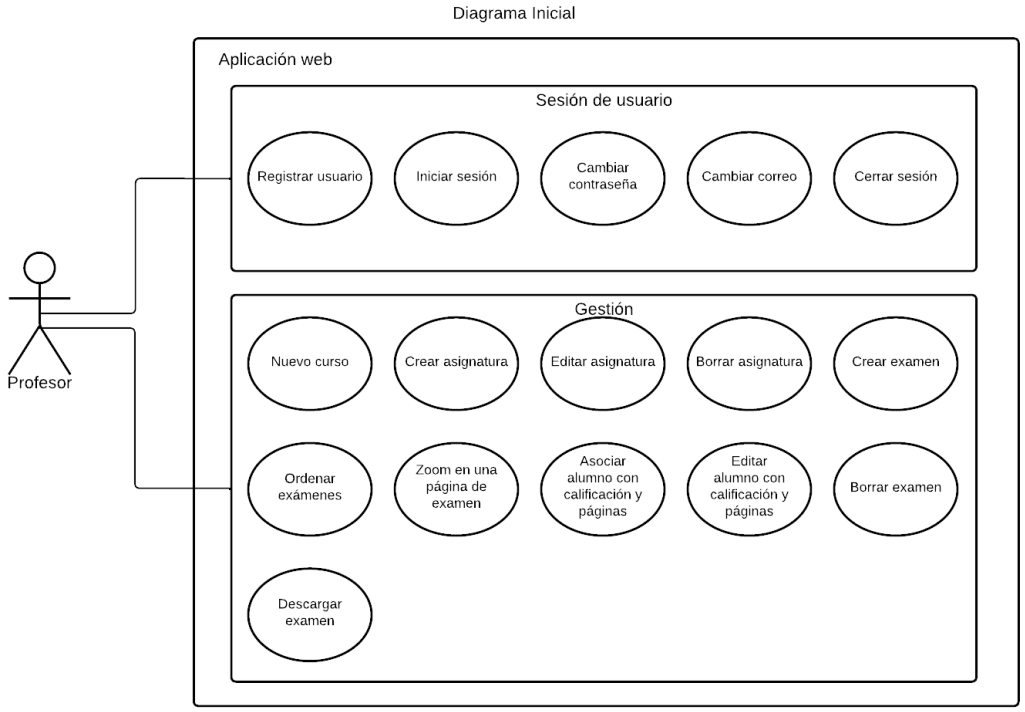


Figura 11 Diagrama inicial.

5.2. Modelo de casos de uso

Al haber un único actor que actúa con la aplicación, todos los casos de uso van dirigidos al profesor (Figura 12). Aunque para llegar al objetivo que tiene la aplicación hay que pasar por varios casos, cada caso de uso es independiente en la funcionalidad de los otros creando un modelo de casos muy simple donde entre estos no se extienden o incluyen funcionalidades.

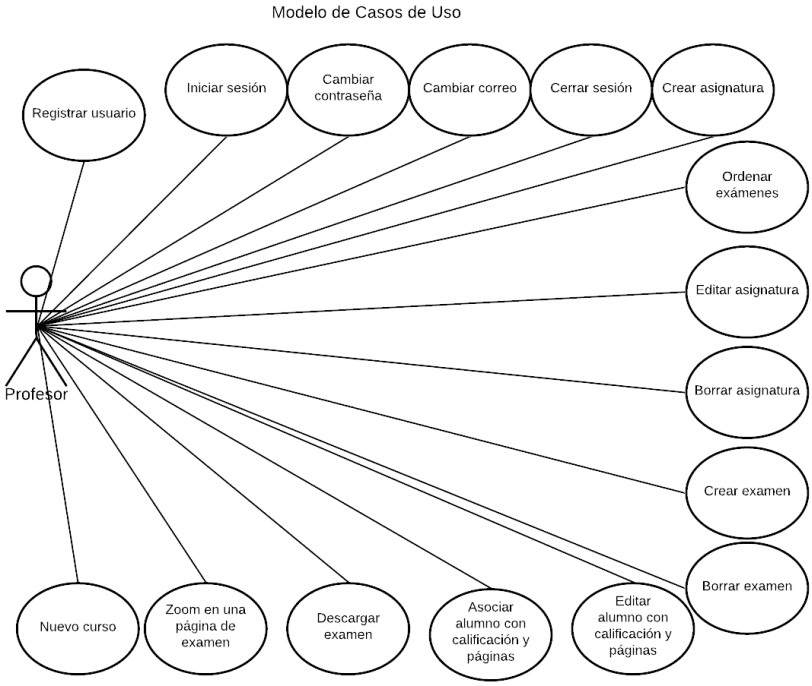


Figura 12 Modelo de Casos de Uso.

5.3. Descripción de casos de uso

A continuación, se estudian y describen todos los casos de uso planteados anteriormente:

Caso de uso	Registrar usuario
Actores	Profesor
Propósito	Dar de alta a un profesor en la aplicación
Resumen	Un profesor nuevo se quiere registrar en la aplicación. Accede a la aplicación web y pulsa en “Registrarse”. A continuación, tendrá un formulario donde introducirá el nombre de usuario que quiere, un correo electrónico del domino “@upv.es” o “@upv.edu.es” para recuperar la contraseña en caso de pérdida y una contraseña para acceder.
Precondición	El profesor no debe estar registrado.
Postcondición	Se guarda el profesor como un usuario de la aplicación donde la contraseña y el correo quedan cifrados.

Caso de uso	Iniciar sesión
Actores	Profesor
Propósito	Acceder a la aplicación
Resumen	Un profesor entra a la página web y pone sus credenciales para iniciar sesión.
Precondición	Estar dado de alta en la aplicación.
Postcondición	Al usuario se le asigna un identificador de sesión iniciada y se redirige a la página inicial de la aplicación donde se encuentra la información principal del usuario.

Caso de uso	Cambiar contraseña
Actores	Profesor
Propósito	Cambiar la contraseña de un profesor de manera segura
Resumen	El profesor ha olvidado o quiere actualizar su contraseña. El profesor solicita cambiar la contraseña y mediante el correo electrónico accede a la web para introducir una nueva contraseña.
Precondición	Estar dado de alta en la aplicación.
Postcondición	Se actualiza la contraseña del usuario.

Caso de uso	Cambiar correo
Actores	Profesor
Propósito	Cambiar el correo de un profesor
Resumen	El profesor quiere actualizar su correo. El profesor accede al menú de opciones dentro de la aplicación y cambia el correo.
Precondición	Tener la sesión iniciada.
Postcondición	Se actualiza el correo del usuario.

Caso de uso	Nuevo curso
Actores	Profesor
Propósito	Habilitar un nuevo curso en la lista
Resumen	El profesor quiere comenzar a preparar el nuevo curso, por tanto, añade un nuevo curso a la lista de cursos para poder empezar a introducir las asignaturas que imparte.
Precondición	Tener la sesión iniciada.
Postcondición	Para que un nuevo curso quede guardado debe crear mínimo una asignatura.

Caso de uso	Crear asignatura
Actores	Profesor
Propósito	Crear una asignatura que imparte el profesor
Resumen	El profesor quiere registrar las asignaturas que imparte. Crear una nueva asignatura indicándole el nombre, acrónimo, un color identificativo y cargando un archivo CSV con la lista de alumnos de la asignatura.
Precondición	Tener la sesión iniciada y mínimo un curso en la lista de cursos.
Postcondición	Se guarda la información de la asignatura y todos sus alumnos. La asignatura se mostrará deshabilitada hasta que la aplicación termine de guardar todos los datos.

Caso de uso	Editar asignatura
Actores	Profesor
Propósito	Editar la información de una asignatura
Resumen	El profesor puede modificar todos los datos de la asignatura, excepto el curso al que pertenece.
Precondición	Tener la sesión iniciada y la asignatura creada.
Postcondición	Se guarda la información de la asignatura. En caso de que no se haya actualizado los alumnos, no se guardarán de nuevo.

Caso de uso	Borrar asignatura
Actores	Profesor
Propósito	Eliminar una asignatura

Resumen	El profesor decide eliminar una asignatura de su lista de asignaturas para que desaparezca de la aplicación. Se elimina la asignatura después de confirmar la acción mediante una ventana emergente.
Precondición	Tener la sesión iniciada y que la asignatura no tenga exámenes asociados.
Postcondición	Se borra la asignatura y toda su información asociada sin posibilidad de recuperarla.

Caso de uso	Crear examen
Actores	Profesor
Propósito	Crea un examen para una asignatura
Resumen	Tras la realización de un acto de evaluación, se quiere clasificar los exámenes de los alumnos. El profesor selecciona la asignatura a la que pertenece el examen y crea este examen. Introduce el nombre y la fecha de realización, además carga una lista de ficheros PDF que contienen los exámenes de los alumnos.
Precondición	Tener la sesión iniciada y al menos una asignatura creada.
Postcondición	Se guarda la información, se añade el examen a la lista de exámenes y queda deshabilitado hasta que toda la información haya sido guardada y procesada.

Caso de uso	Ordenar exámenes
Actores	Profesor
Propósito	Ordenar los exámenes por nombre o por fecha
Resumen	El profesor, según su preferencia, ordena los exámenes.
Precondición	El usuario debe tener exámenes creados.
Postcondición	Los exámenes deben quedar ordenados de izquierda a derecha según la selección del profesor.

Caso de uso	Zoom en una página de examen
Actores	Profesor
Propósito	Ampliar una zona de la página del examen
Resumen	El profesor no tiene claro el nombre del alumno de un examen, por tanto, pone el cursor encima de la imagen para que le aparezca un zoom y poder leer el nombre cómodamente.
Precondición	Tener la sesión iniciada y estar dentro de un examen.
Postcondición	Mostrar un rectángulo con la imagen aumentada de la zona donde está el cursor.

Caso de uso	Asociar alumno con calificación y páginas
Actores	Profesor
Propósito	Asociar la información un alumno con su examen
Resumen	Dentro de un examen, el profesor quiere asignar tanto la nota como el rango de páginas del examen de un alumno. Aunque tenga la posibilidad de usar el ratón, el profesor es un usuario avanzado y deja las manos en el teclado, por tanto, con la página inicial del examen del alumno, escribe un par de letras que compone el nombre del alumno (o el DNI), con las flechas selecciona el alumno en una lista que se despliega, con “intro” selecciona el alumno, a continuación, tabula para introducir la nota del alumno que ve en la página inicial, vuelve a tabular y usando las flechas o “intro”, se sitúa en la página final del examen del alumno, finalmente tabula y pulsa “intro” para guardar la información.

Precondición	Tener la sesión iniciada y un examen con alumnos disponibles.
Postcondición	Guardar la información del alumno y mostrarla en una tabla junto con los demás alumnos con información asociada.

Caso de uso	Editar alumno con calificación y páginas
Actores	Profesor
Propósito	Editar la información de un alumno dentro de un examen.
Resumen	El profesor se da cuenta de que el alumno está mal clasificado, por tanto, busca este alumno ordenando la tabla y decide editarlo. Introduce sus datos correctamente y guarda los cambios.
Precondición	Tener la sesión iniciada y que el alumno esté en la tabla de alumnos con información asociada.
Postcondición	Guardar los cambios y posicionar el examen en el punto en que dejó el profesor la corrección.

Caso de uso	Borrar examen
Actores	Profesor
Propósito	Eliminar un acto de evaluación definitivamente
Resumen	El profesor, por el motivo que sea, decide borrar el examen y toda su información, por tanto, borra el examen después de confirmar en el diálogo de advertencia.
Precondición	Tener la sesión iniciada y estar dentro de una asignatura.
Postcondición	Borrar toda la información asociada del examen (alumnos con información, el documento y el propio examen).

Caso de uso	Descargar un examen
Actores	Profesor
Propósito	Descargar toda la información de un examen
Resumen	Una vez finalizada la clasificación de los exámenes el profesor quiere descargar la información, para, por un lado, subirla a padrino y por otro, compartir cada examen con el alumno correspondiente. Por tanto, solicita descargar el examen y descargará un archivo ZIP donde encontrará una carpeta con el nombre del examen, que dentro tendrá dos carpetas, una con los archivos CSV, para subirla las notas a padrino y toda la información del alumno asociada en la aplicación, y otra carpeta con un conjunto de archivos PDF con el nombre y la información necesaria de los exámenes de cada alumno y así poder subirla al espacio compartido de PoliformaT.
Precondición	Tener la sesión iniciada y un examen creado.
Postcondición	Crear y descargar un archivo ZIP con los requisitos de la información del examen preparada para subir, tanto las notas a PADRINO, como para subir los exámenes de los alumnos a su espacio compartido.

Caso de uso	Cerrar sesión
Actores	Profesor
Propósito	Cerrar sesión en un dispositivo
Resumen	Cierra la sesión del profesor invalidando el acceso a la aplicación, de manera que para volver a entrar y retomar el trabajo por donde el profesor lo dejó hace falta introducir las credenciales.
Precondición	Tener la sesión iniciada.
Postcondición	Guardar la petición de “cerrar sesión” y enviar al usuario a una pantalla que no necesite credenciales.

6. Diseño

En este capítulo se presenta el diseño que tiene la aplicación, tanto la arquitectura de la aplicación web, el modelo de datos que sigue y el diseño gráfico y lógico. Se explican las líneas de diseño que siguen las pantallas de la aplicación y porqué se toman estas decisiones. También se ve el conjunto de funciones que componen la aplicación y una usabilidad centrada en el usuario. Por último, se indican los métodos usados para mantener la seguridad de los datos en la aplicación.

6.1. Arquitectura

La aplicación se diseña mediante el uso de la arquitectura cliente servidor, específicamente el modelo de desarrollo en tres capas: presentación, negocio y persistencia [4]. Se usa un diseño web para facilitar el acceso y el despliegue al usuario. También, de esta manera se consigue un software multiplataforma. La decisión se toma pensando en el usuario y la disposición de usar tecnologías que permiten un acceso directo sin complicaciones.

La Figura 13 muestra la arquitectura de la aplicación web.

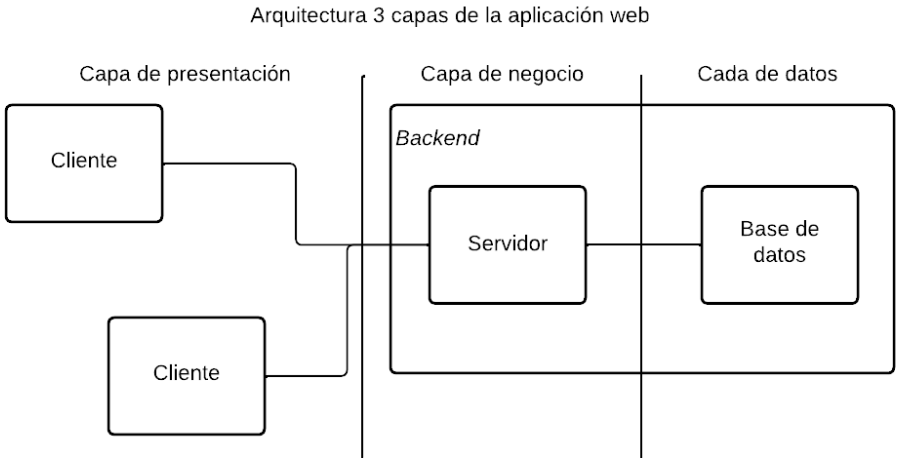


Figura 13 Arquitectura en tres capas de la aplicación web.

El cliente o *frontend* implementa la capa de presentación usando un lenguaje que puede ser interpretado por los navegadores, y su diseño sigue el patrón de desarrollo MVC (modelo, vista, controlador). Gracias a esto, el proyecto dispone de un *middleware* que puede interactuar tanto con el servidor como con las funciones que ofrecen los navegadores web como, por ejemplo, guardar datos de sesión. La Figura 14 muestra el diagrama del patrón MVC usado por el cliente o *frontend*.

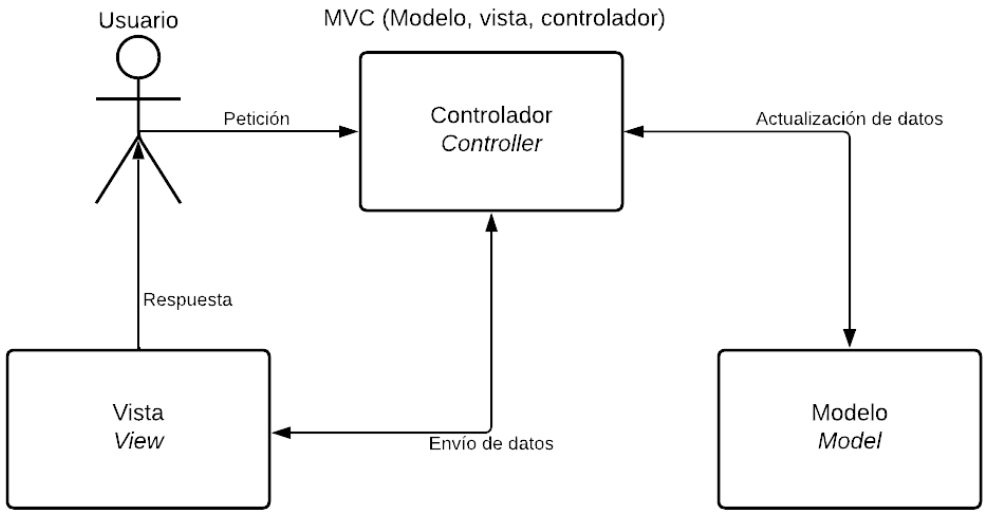


Figura 14 Patrón de desarrollo software MCV para el cliente.

En el servidor o *backend* se aloja tanto la capa de negocio como la capa de datos, además de un servidor web. Dispone, por tanto, de tres servicios.

- Un servicio web público para el cliente (Apache[3]) [5], es decir, el contenedor donde se aloja el *frontend* de la aplicación y que permite al cliente recuperarlo para ejecutarlo en local.
- Un servicio (Express[4]) [6] que corresponde a la función de la capa de negocio, el servicio al cual el cliente realiza peticiones. Este servicio ofrece un servidor API REST y otro servidor API Socket. Al ser una API REST, el servidor atiende a peticiones HTTP del cliente y devuelve las respuestas. La API Sockets establece un canal de comunicación bidireccional con el cliente, en la que tanto el servidor como el cliente pueden iniciar la comunicación. Es decir, el cliente no necesita enviar una petición HTTP para que haya intercambio de datos entre ambos. La ventaja es poder usar el mismo puerto de escucha para ambos servicios y estar en el mismo servidor, de manera que el procesado de la información se hace en el mismo proyecto.
- El tercer servicio corresponde a la capa de persistencia (MySql[5]) [7]. En este caso, la base de datos se desplegará en la misma máquina que el *backend*. Es un servicio privado que perfectamente podría ser separado y estar en otra máquina teniendo así las tres capas en tres máquinas diferentes, pero para el desarrollo de este proyecto no se considera necesario ni aportaría ninguna ventaja.

La Figura 15 muestra la estructura del servidor: un servidor web público para ofrecer la aplicación *frontend*, un servidor *backend* con comunicación por HTTP y TCP/IP [8], y una base de datos privada.

[3] https://httpd.apache.org/
[4] https://expressjs.com
[5] https://www.mysql.com/

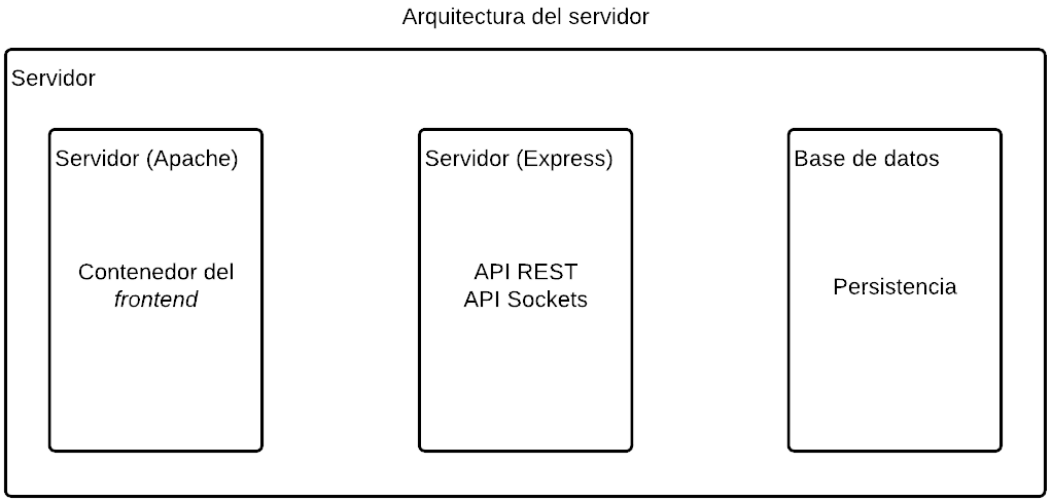


Figura 15 Arquitectura del servidor.

6.2. Negocio

En la capa de negocio (o lógica) se diseña el modelo de datos que tiene la aplicación, es decir, los conjuntos de datos que tiene y como los eventos generados por el cliente acceden y modifican los datos. La Figura 16 muestra las clases y las funciones que pueden acceder y modificar cada clase. Se puntualiza que hay atributos que realmente son prescindibles en el modelo de datos, sin embargo, están para facilitar ciertas funciones como el filtrado de exámenes o su ordenación.

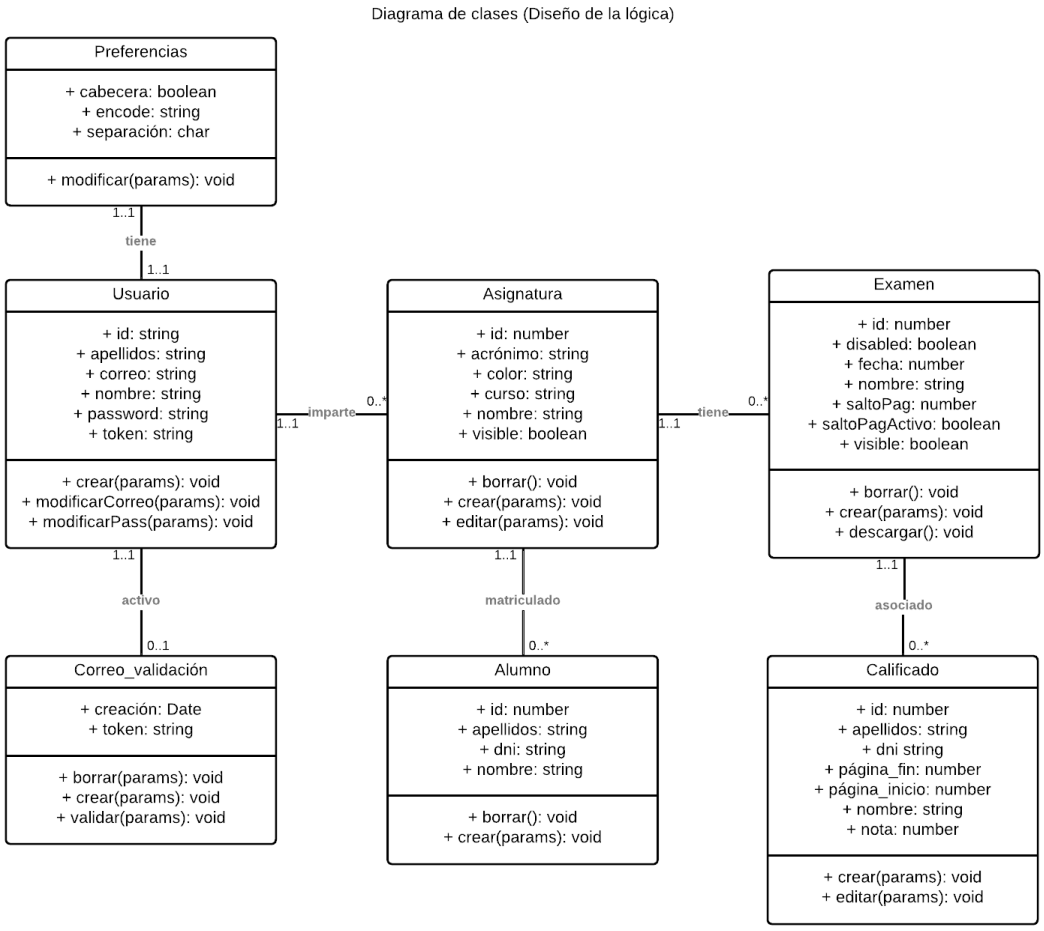
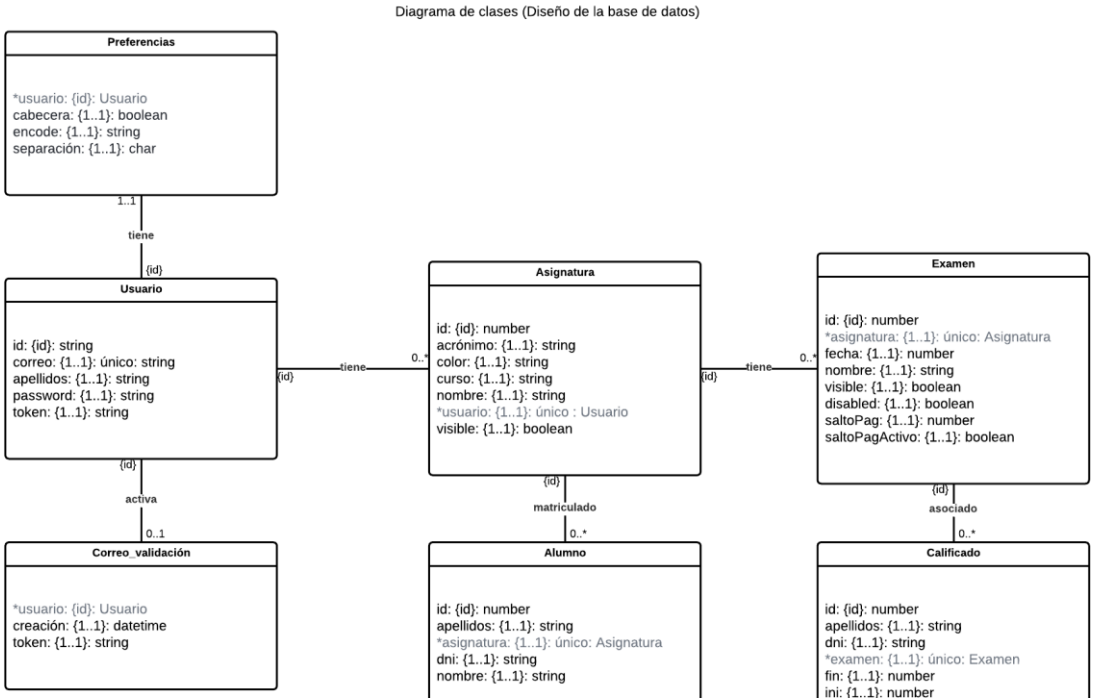


Figura 16 Diagrama de clases para el diseño de la lógica.

6.3. Persistencia

En el diseño de la base de datos aparecen todos los objetos que deben tener persistencia. Se incluyen también algunos de los atributos que permiten agilizar las consultas y/o ofrecer una buena experiencia al usuario. Además, hay que mencionar que tanto la clase de “Preferencias” como la de “Correo validación” son para mantener la persistencia en las opciones del usuario al importar los alumnos y para dar validación a los correos a la hora de cambiar la contraseña. El diagrama del diseño de la persistencia se puede ver en la Figura 17, donde se especifica las relaciones, el tipado y las restricciones.



*Indica el atributo que funciona como complemento agente para la relación entre tablas.

Figura 17 Diagrama de clases para el diseño de la base de datos.

Por último, en cuanto a los documentos PDF que contienen los exámenes de los alumnos, se ha decidido guardarlos en el servidor de manera local, organizados en carpetas, cada una con un nombre compuesto de los identificadores del usuario, asignatura y examen. Por tanto, no es necesario guardar los documentos en la base de datos evitando así ineficiencia en las consultas.

6.4. Presentación: Interfaz gráfica y experiencia de usuario (UI/UX)

El trabajo de clasificar los exámenes puede llegar a ser tedioso. Un profesor puede tener 200 exámenes que procesar e incluso más. Un punto fuerte y principal que debe tener la aplicación que se desarrolla en este trabajo es una interfaz enfocada al usuario, cómoda, simple, con una experiencia de uso satisfactoria y agradable. La usabilidad de la aplicación es una prioridad.

Todas las decisiones sobre el diseño gráfico de la aplicación están tomadas en torno al profesorado y sus especificaciones.

6.4.1. Líneas de diseño básicas

La aplicación cuenta con una barra superior que se mantiene en todas las páginas (Figura 18). Esta barra permite al usuario acceder a la ayuda, cambiar el correo y cerrar sesión, dando la oportunidad de continuar con su trabajo en otro momento. También es un identificativo para decirle al usuario la aplicación que está usando. Se aprovecha para añadir la funcionalidad de retroceder. La disposición de ambos botones ya sea el de retroceder o el de cerrar sesión siguen un estándar que se ve en la mayoría de las aplicaciones. El botón de retroceder normalmente se encuentra arriba a la izquierda, al igual que el botón de las opciones que se dispone arriba a la derecha, encerrados en una barra superior con el nombre de la aplicación. Los botones solo se muestran en caso de que se pueda realizar la acción.

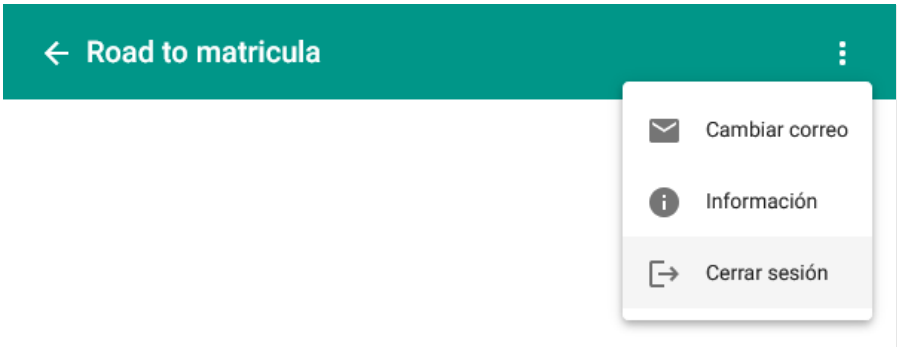


Figura 18 Barra superior.

Las pantallas de “Iniciar sesión”, “Registrar Usuario” o “Cambiar contraseña” tienen un diseño simple. En cada caso, el usuario solo tiene que rellenar un formulario, con un uso simple y directo. La Figura 19 muestra los tres tipos de formularios previos al iniciar sesión.

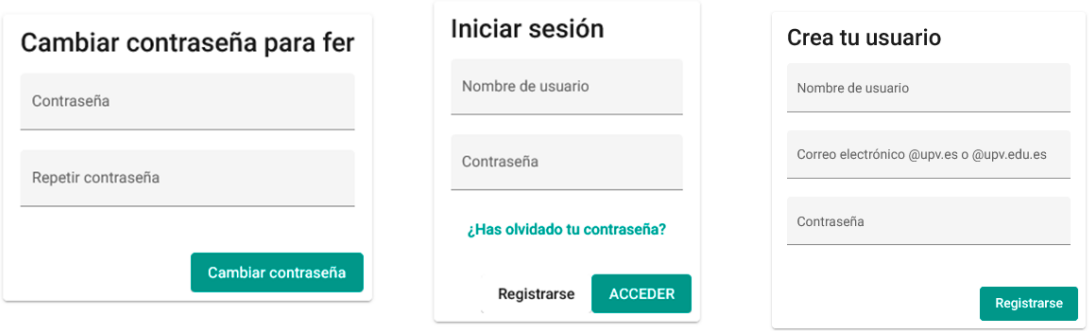


Figura 19 Formularios para: cambiar la contraseña, iniciar sesión y crear un usuario.

La aplicación usa ventanas emergentes para funciones bloqueantes. Así centra la atención del usuario en el procedimiento. Es decir, para aquellas operaciones que requieran únicamente que el usuario rellene un formulario, se muestra una ventana de diálogo con el formulario necesario. Un ejemplo es el formulario para cambiar el correo electrónico en la Figura 20.

Figura 20 Ejemplo de ventana emergente (Diálogo para cambiar el correo del usuario).

Por un lado, están los diálogos de confirmación. Simples ventanas con información advirtiendo de la acción que se desea realizar para confirmar la acción, seguido de sus respectivos botones de cancelar y confirmar. La Figura 21 muestra un ejemplo de este tipo de confirmaciones.

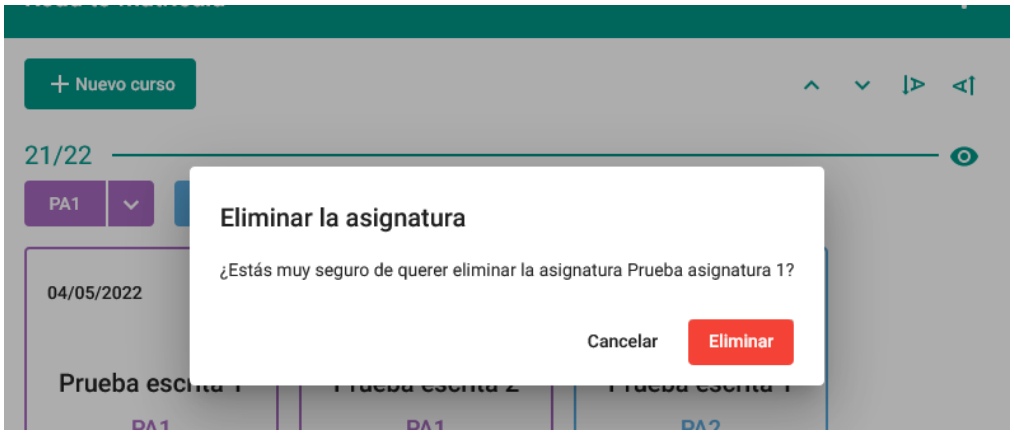


Figura 21 Ejemplo de ventana de confirmación emergente (Diálogo de advertencia al borrar una asignatura).

Hay que destacar que para todo el diseño se sigue la regla, siempre que se pueda, de posicionar la botonera a la derecha, siendo el botón más a la derecha el que haga la acción principal, dado que, en la cultura occidental, se lee de izquierda a derecha y, por tanto, la acción de continuar (que se entiende como la acción principal ya que permite avanzar) se posiciona más a la derecha.

6.4.2. Indicadores de progreso

Por último, para aquellas acciones, principalmente cargas o descargas de ficheros, que son críticas y pueden requerir un tiempo considerable, se muestra el progreso del proceso. Así se le da al usuario constancia de que la aplicación está trabajando y no que se ha quedado bloqueada. Una manera de evitar frustraciones y dar una buena experiencia de uso.

La visualización de este progreso siempre se hace mediante el uso de Sockets. Al establecer una comunicación bidireccional cuando el cliente inicia sesión, el servidor puede informar al cliente

del progreso de las operaciones con coste temporal elevado. Eso se puede ver, por ejemplo, en la subida de la lista de alumnos, aunque el proceso suele ser bastante rápido y en la subida de los archivos PDF de los exámenes escaneados, ya que su procesado sí que lleva tiempo.

6.5. Pantalla principal (asignaturas y exámenes)

A la pantalla principal se accede una vez se inicia sesión. Desde aquí se accede a toda la información que tiene almacenada el usuario. Además, incluye las funcionalidades de gestión, tanto del curso, de asignaturas, como de exámenes (Figura 22).

La meta es llegar a crear un examen, para ello se necesita una asignatura a la cual pertenece el examen y la asignatura debe ser de un curso, por tanto, la primera funcionalidad que se muestra es la de crear un nuevo curso. Los cursos se muestran en forma de lista, de esta manera hay un registro organizado de toda la información. La lista está ordenada de mayor a menor para que el curso más reciente sea el primero que se encuentre. De esta manera está la barra superior, constante para toda la aplicación anclada a la parte superior de la pantalla, y debajo de ella, una cabecera fija con un botón para crear un nuevo curso académico. Aprovechando el espacio de la cabecera de curso y que esta se mantiene en la pantalla, las funciones de ordenar los exámenes se incluyen ahí. El resultado es una cabecera fija con funciones claras y de acceso fácil, y seguido en forma de lista todos los cursos que crea el usuario.

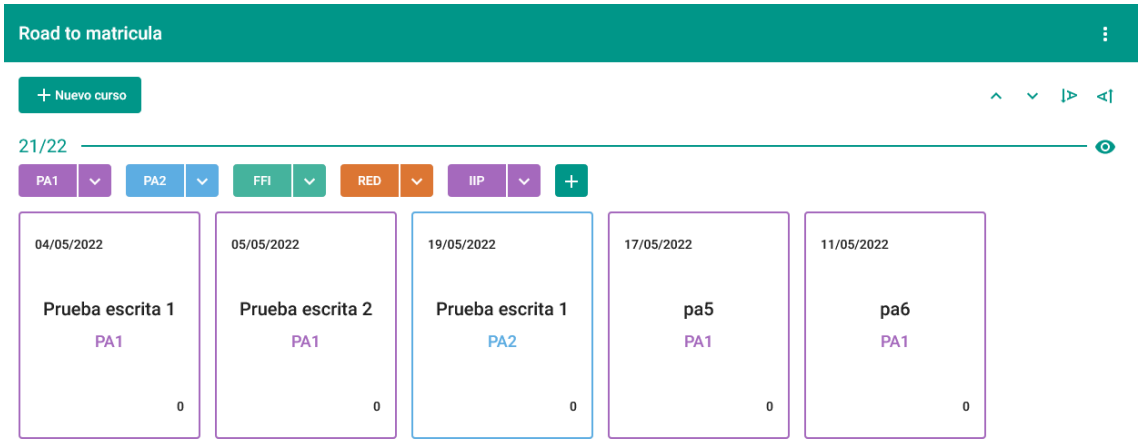


Figura 22 Pantalla principal con la gestión de asignaturas y exámenes.

Cada curso tiene dos filas. El uso de filas es porque aquí los componentes pueden crecer mucho y la aplicación está pensada para ser usada en ordenador, es decir, la visualización es en una pantalla en formato 16:9.

La primera de estas dos filas muestra la lista de asignaturas del curso, con un botón final para crear una nueva asignatura (Figura 23). Cada asignatura dispone de un desplegable que se muestra en forma de icono para dar acceso a sus funciones (crear examen, editar y borrar). La asignatura muestra únicamente su acrónimo para facilitar la identificación, además del uso de colores. Sin embargo, posicionando el puntero encima del botón salta un cuadro de texto con el nombre completo de la asignatura, una manera secundaria de identificar la asignatura, por ejemplo, en caso de que dos o más asignaturas tengan el mismo acrónimo. No llega a ser

necesario que esta lista esté ordenada, ocupa poco espacio en pantalla, con los acrónimos y los colores la identificación se hace muy rápida.

Figura 23 Lista de asignaturas de un curso.

Los exámenes también se muestran en fila por el mismo motivo que las asignaturas (Figura 24). El botón del examen está compuesto por: fecha, dado que es un campo por el cual se puede ordenar la lista, el nombre del examen centrado en el botón (también campo por el que ordenar la lista), junto al acrónimo de la asignatura en su propio color, y finalmente un conteo de alumnos con el examen ya asignado. El acrónimo y el borde del botón se muestran con el color de la asignatura asociada, una manera de identificación rápida. La fecha y el conteo de alumnos con examen se muestran con una disposición opuesta y con una fuente más pequeña que el nombre del examen. Así se destaca el nombre del examen y con los otros dos campos de texto separados, da la sensación de un botón más limpio y minimalista, pero permitiendo mostrar esa información en un segundo plano menos importante e intrusivo.

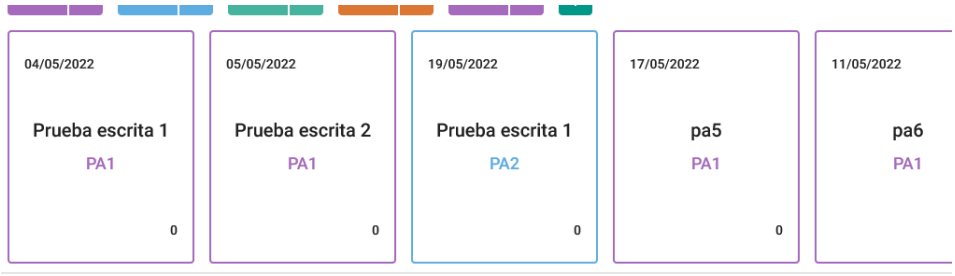


Figura 24 Lista de exámenes de un curso.

6.6. Asignatura y lista de alumnos

Para los formularios de crear y editar una signatura se cuenta con dos partes que se pueden ver en la Figura 25. La primera son campos de texto para introducir la información de la asignatura. La segunda es un lector de archivos CSV. Para esto, se carga el archivo y a continuación se debe seleccionar el formato que tiene (la codificación, el separador y si hay cabecera), todo esto en la misma línea dado que pertenece al mismo proceso. A continuación, se muestra en un campo de texto el resultado de procesar la información del archivo cargado en texto plano, así el usuario puede ver que se está leyendo bien. Le sigue la selección de columnas. La aplicación necesita almacenar el numero identificativo, los apellidos y el nombre del alumno, por tanto, hay cuatro selectores. Según la combinación que tenga el archivo, la aplicación procesa esta información de una manera u otra. De forma dinámica se actualiza la tabla donde se muestra el resultado de las acciones del profesor. Esta segunda parte se describe con detalle en el subapartado siguiente.

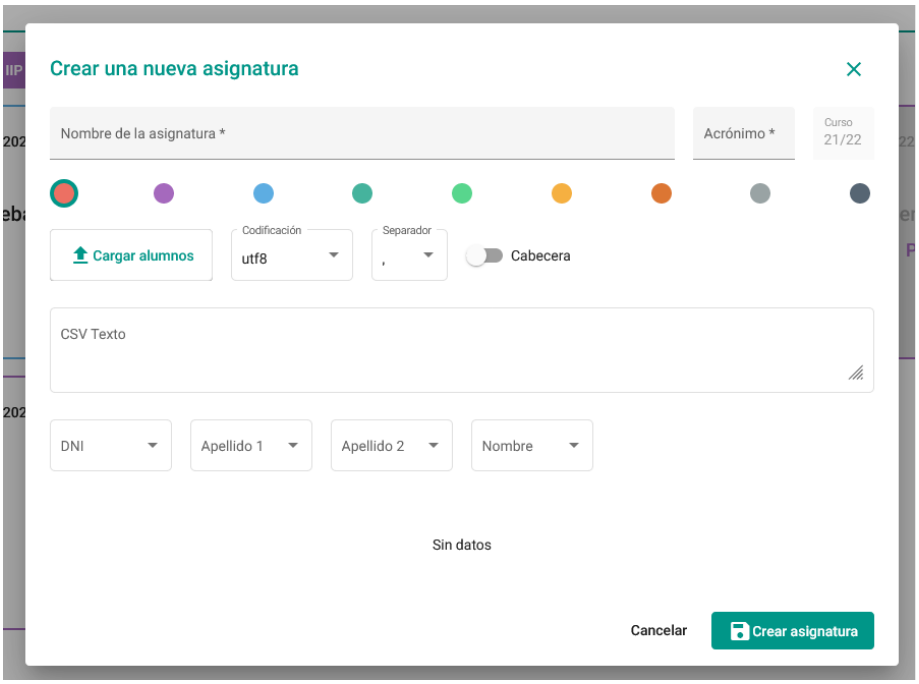


Figura 25 Ventana emergente/formulario para crear una asignatura.

Al guardar la información, aparece la asignatura con un indicador de progreso en porcentaje. Este indicador informa del porcentaje de alumnos guardado. Realmente su coste temporal no es tan elevado como para que sea perceptible fácilmente, sin embargo, se implementa para dar una mejor experiencia al usuario.

6.6.1. Importación de la lista de alumnos

Uno de los puntos críticos es tener la capacidad de subir la lista de alumnos de la asignatura. El inconveniente reside en los múltiples formatos de archivos CSV con que se pueden generar la lista de alumnos. Para ello se implemente un lector con opciones dando la capacidad de configurar la lectura de la lista con el formato que se desea.

La aplicación cuenta con la opción de escoger la codificación del archivo dando las codificaciones más generales y usadas en la UPV evitando problemas de acentos o caracteres extraños. También cuenta con la selección del carácter separador de las columnas y un indicador de si hay cabecera. Con esto se cubren las opciones básicas a la hora de leer un archivo tipo CSV. Para mejorar la experiencia de usuario se muestra en una caja el texto plano del archivo con la codificación seleccionada, de manera que el profesorado pueda ver que la codificación y los otros campos de selección de formato sean los correctos, comprobando manualmente que el archivo es bueno. Además, para agilizar el proceso, se mantiene el último formato introducido.

La selección de codificación es importante como se ve en la Figura 26, donde se muestra para un archivo con codificación Latin1 qué ocurre si no se marca correctamente la codificación. UTF-8 está siendo uno de los estándares de codificación actual, esto significa que aplicaciones antiguas no compatibilizan siempre con la codificación de herramientas más modernas. Para hacer una lectura ágil del archivo se hace directamente en texto plano indicando la codificación

que tiene el archivo, dado que realizar la conversión manualmente requeriría leer bytes del archivo y posteriormente recorrer toda esta información dividiendo la información en fragmentos correspondientes a su codificación y transformar los bytes en caracteres.

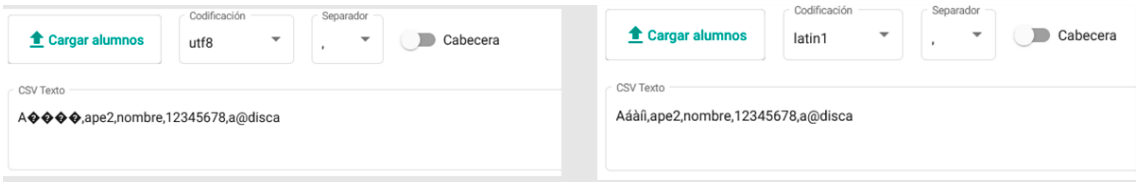


Figura 26 Ejemplo de codificación del texto incorrecta contra correcta.

Una vez con el formato correcto de la lista se debe seleccionar las columnas con información útil para la aplicación (número identificativo, nombre y apellidos). Como no es obligatoria la cabecera en el archivo, también se debe poder seleccionar para cada columna qué información contiene. El resultado son cuatro campos de selección correspondientes a los datos del alumno donde se selecciona, en cada campo, la columna que contiene dicha información, como muestra la Figura 27, donde cada columna es seleccionada en su correspondiente campo. Por ejemplo, en el campo de selección de DNI o número identificativo, se seleccionará la columna que contenga el DNI del alumno. La cantidad máxima de columnas útiles pueden llegar a ser 4 (DNI, primer apellido, segundo apellido y nombre) y la información requerida por la aplicación son de 3 campos (DNI, apellidos y nombre). Para poder dar la máxima versatilidad a la selección, la combinación de estos campos da resultados diferentes.

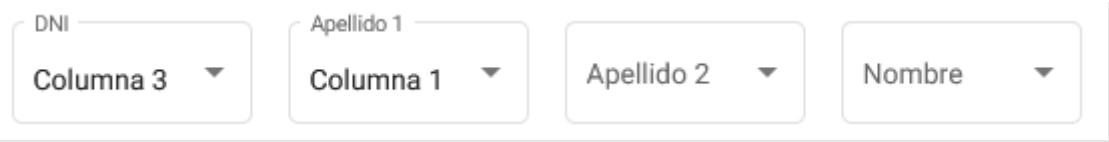


Figura 27 Selección de columnas en sus respectivos campos.

Las combinaciones posibles en un archivo completo:

- Columna con DNI, columna con el primer apellido, columna con el segundo apellido y columna con el nombre. Los apellidos se concatenan en orden dando como resultado un campo de apellidos.
- Columna con DNI, columna con el primer y segundo apellido y nombre. Los campos tienen el formato idóneo.
- Columna con DNI y columna con los apellidos y nombre. En este caso se divide los apellidos y el nombre teniendo en cuenta que están separados por una coma.

Estos sería los tres casos posibles en caso de que el alumno tenga toda su información, sin embargo, se tienen en cuenta y se tratan casos en los que a un alumno le falte algún campo como, por ejemplo, un segundo apellido.

La utilidad de tratar así estos campos es poder decirle al usuario que simplemente debe seleccionar la columna que contenga la información del campo y la aplicación ya se encargará de procesar y guardar lo que necesite.

Por último y pensando en el usuario, tras la selección de las columnas aparece una tabla con el resultado final del procesado de la lista de alumnos, quedando el apartado para leer archivos CSV como en la Figura 28.

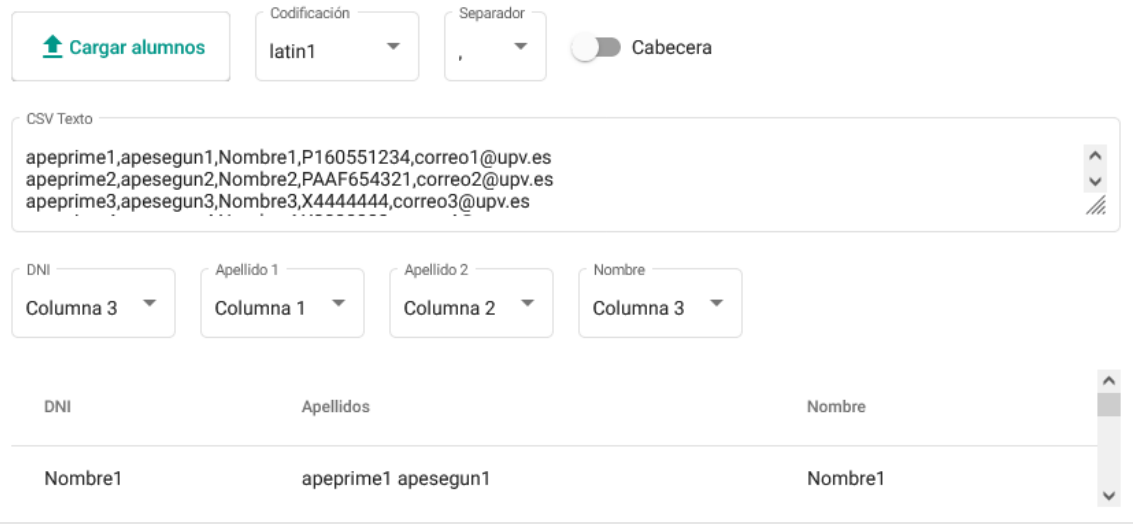


Figura 28 Ejemplo de una correcta lectura de un archivo CSV.

6.7. Exámenes y su proceso de asociación

La creación de un examen se realiza mediante un formulario simple (Figura 29). En la primera línea muestra un campo de texto para el nombre del examen y un selector minimalista de la fecha de realización, campos que permiten identificar un examen. Le sigue la función para cargar todos los archivos PDF con los exámenes escaneados. Una vez se cargan, los archivos aparecen en una lista con la capacidad de ser ordenados mediante arrastre. Aunque es habitual que los equipos de escaneo asignen nombres que permiten ordenar automáticamente los ficheros, esta función se pone para facilitar al usuario ordenar los archivos en caso de que no lo estén sin necesidad de cambiar el nombre de los ficheros manualmente.

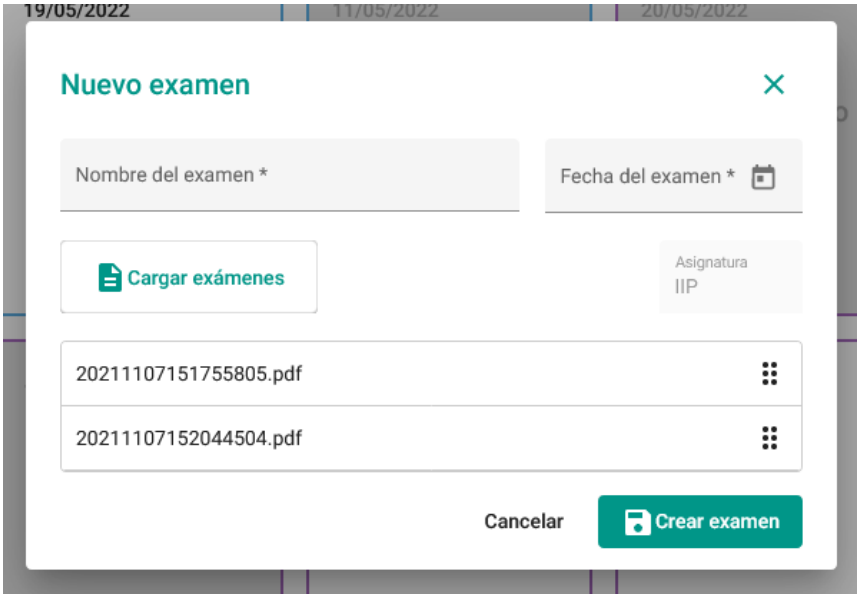


Figura 29 Ventana emergente/formulario para crear un nuevo examen.

La pantalla de exámenes, accesible una vez procesada la creación del examen, es donde el profesor pasará más tiempo trabajando. Aquí se hace la asociación de los alumnos con sus notas y hojas de examen. La ventana debe mostrar toda la información necesaria y correctamente organizada, facilitando el trabajo al profesor. La pantalla donde se enfoca el trabajo debe dar al usuario una experiencia agradable y con un uso sencillo.

Se mantiene la barra superior, y la parte inferior de la pantalla está dividida en dos partes, a la izquierda la visualización de los exámenes con una barra de navegación simple, y a la derecha una parte de control donde se asigna y visualiza los alumnos.

6.7.1. Subida y procesado de los exámenes escaneados

En la aplicación hay un par procesos que pueden presentar un coste temporal considerablemente elevado.

El principal y más costoso es el procesado de archivos PDF con los exámenes de los alumnos. Esto ocurre porque trabajar con este tipo de archivos necesita de bibliotecas y aplicaciones externas, además de que estos archivos suelen ser pesados en cuanto a su tamaño, por tanto, cuanta más información a procesar, más tiempo lleva.

Para enviar un archivo se lee su contenido y se crean paquetes de poco más de 1 MB, que son enviados al servidor mediante la conexión de Sockets. De esta forma se puede mostrar en el cliente el progreso del proceso. Además, se evita que el envío de un fichero de gran tamaño bloquee las peticiones al servidor.

Cuando se tienen todos los paquetes de un archivo, se genera el documento en el lado del servidor. Para evitar problemas, todos los archivos son procesados por una herramienta de consola que reduce las capas del documento, es decir, si el documento tiene insertadas imágenes, comentarios, o dibujos (entre otras funciones de los documentos PDF), estos se procesan y se incrustan a la hoja. De esta manera, cuando los diferentes documentos (en caso de que sean varios) son concatenados generando un único archivo y posteriormente convertido a imágenes, las anotaciones se mantendrán. Si el profesor ha corregido los exámenes sobre el documento PDF, las notas no se perderán. El documento es convertido a imagen para mejorar el control sobre el visor de exámenes.

De la mano de los costes temporales están los tiempos de espera. Para evitar frustraciones y malas experiencias en la aplicación, al crear un examen se muestra el porcentaje del progreso dividido en 6 fases:

- Leer. Lectura de los archivos en el navegador.
- Subir. Cantidad enviada de los paquetes resultado de fragmentar todos los archivos.
- Procesar. Captura y procesado de la información de los paquetes enviados.
- Crear. Reconstrucción de los archivos en el servidor.
- Generar. Creación de un único archivo concatenando los exámenes.
- Preparar. Transformar los exámenes a imágenes para poder visualizarlos en la aplicación.

La Figura 30 es una captura de la ventana que muestra el progreso de creación de un examen. Esta muestra el progreso del procesamiento de sus 6 fases.

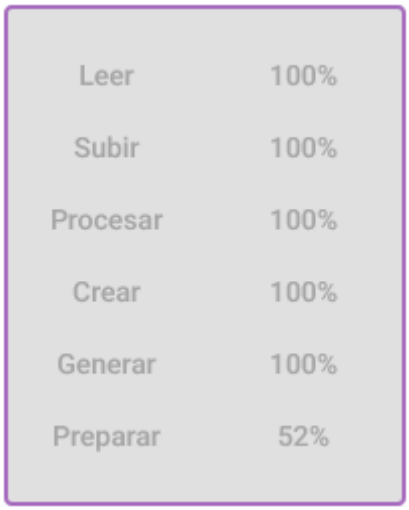


Figura 30 Examen mientras se procesa.

6.7.2. Visor de exámenes

El visor de los exámenes está a la izquierda de la pantalla por una razón concreta. En la cultura occidental leemos de izquierda a derecha, por tanto, resulta más cómodo tener la información principal o importante a la izquierda. La visualización no es un apoyo a la hora de asociar los exámenes, más bien es uno de los pasos que se deben seguir para este trabajo, por lo cual, no se puede tratar como información de apoyo sino como parte principal del recorrido a seguir, en concreto, el punto de partida.

La barra de navegación se sitúa a la derecha del visor de exámenes. Esta barra permite al usuario controlar el visor, además le da la oportunidad de asignar de forma manual el rango de páginas de un alumno. Es una barra de navegación donde el centro muestra la página que se visualiza y según la acción que se quiera, retroceder o avanzar, los controles se disponen arriba o abajo de este centro respectivamente. Además, muestra la selección de la página inicio y final del examen del alumno y dos botones para asignar manualmente la página inicial y final. La navegación está lo más arriba de la pantalla para acercarla lo máximo posible a los controles de asignación de alumno y nota de la zona derecha.

Una función imprescindible es el poder ampliar el examen. El objetivo de implementar esta capacidad viene dado por la necesidad del profesor de identificar al alumno del examen. Para ello tiene que leer el número de identidad o el nombre del alumno, textos escritos a mano por un alumno que puede tener mala caligrafía. Para ayudar al usuario se crea un rectángulo alargado dentro del visor del examen. El rectángulo funciona a modo de lupa y su forma está enfocada a aumentar la zona en la que el alumno pone su identificativo.

En la Figura 31 se localizan todas las características mencionadas para el visor de exámenes.

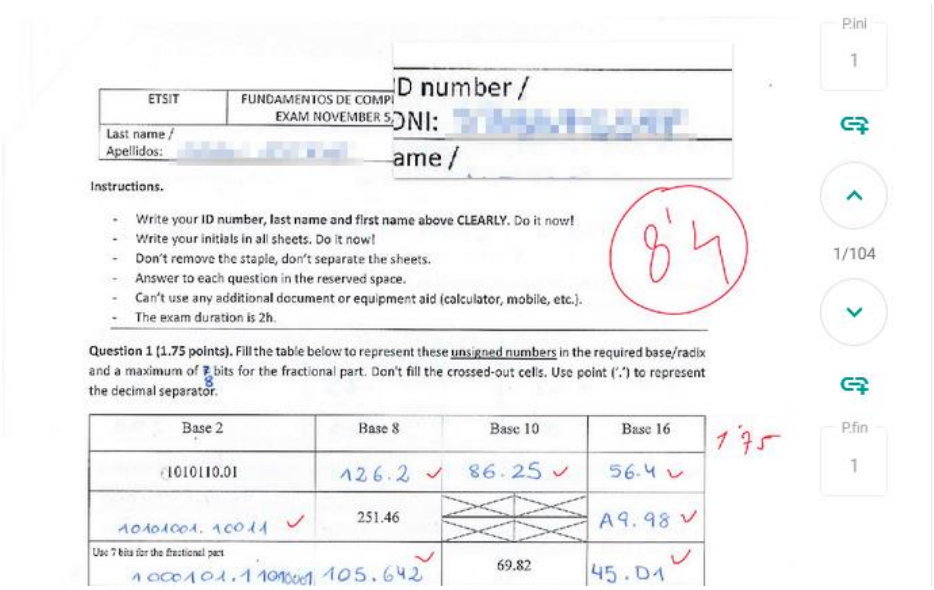


Figura 31 Visor del examen con la lupa activada.

La lupa amplia la zona del examen sobre la que está posicionada. La zona será ampliada por control del usuario, al igual que su posicionamiento, es decir, el usuario tiene el control para mostrar la lupa y posicionarla sobre la zona que desee. Por último, una característica muy cómoda que tiene la función es poder posicionarla en el examen y que se mantenga en todas las hojas, de modo que una vez configurada no hará falta volver a interactuar con el zoom.

6.7.3. Gestión de alumnos

En el lado derecho de la pantalla, en forma de columna, se disponen cuatro partes, una identificativa, otra de asignación, la de información y finalmente la de control.

Primero, en la identificación, se muestra el curso, acrónimo y nombre del examen para dejar claro qué prueba se está manipulando.

Seguido, está la parte de asignación dividida en dos filas. La primera fila contiene el número de páginas que se desea saltar automáticamente una vez se guarde un alumno y le sigue un botón para activar esta función. Se muestra esto primero en la parte de asignación, porque en caso de que los exámenes estén compuestos por el mismo número de páginas, debe ser el primer campo para rellenar, introduciendo el número de páginas del examen y acto seguido, la confirmación de que se tenga en cuenta esta información. En la segunda fila de la asociación están los campos para introducir el nombre del alumno, la nota, la asignación de la página final del examen y la confirmación para guardar. El proceso de asociación se explica más adelante, pero el diseño tiene este orden pensando en el usuario. Una vez se valida la asignación alumnos-examen, el alumno aparecerá en la tabla de información.

La tercera parte muestra una tabla con los alumnos y su información asociada. Esta tabla permite ver el progreso y los alumnos con examen, además, da la oportunidad de editar algún alumno que tenga mal asociado algún campo.

Finalmente, en la cuarta parte están los botones de control, un botón por si se quiere eliminar el examen y toda su información, y otro para descargar toda la información relacionada con el examen. El resultado se puede ver en la Figura 32.

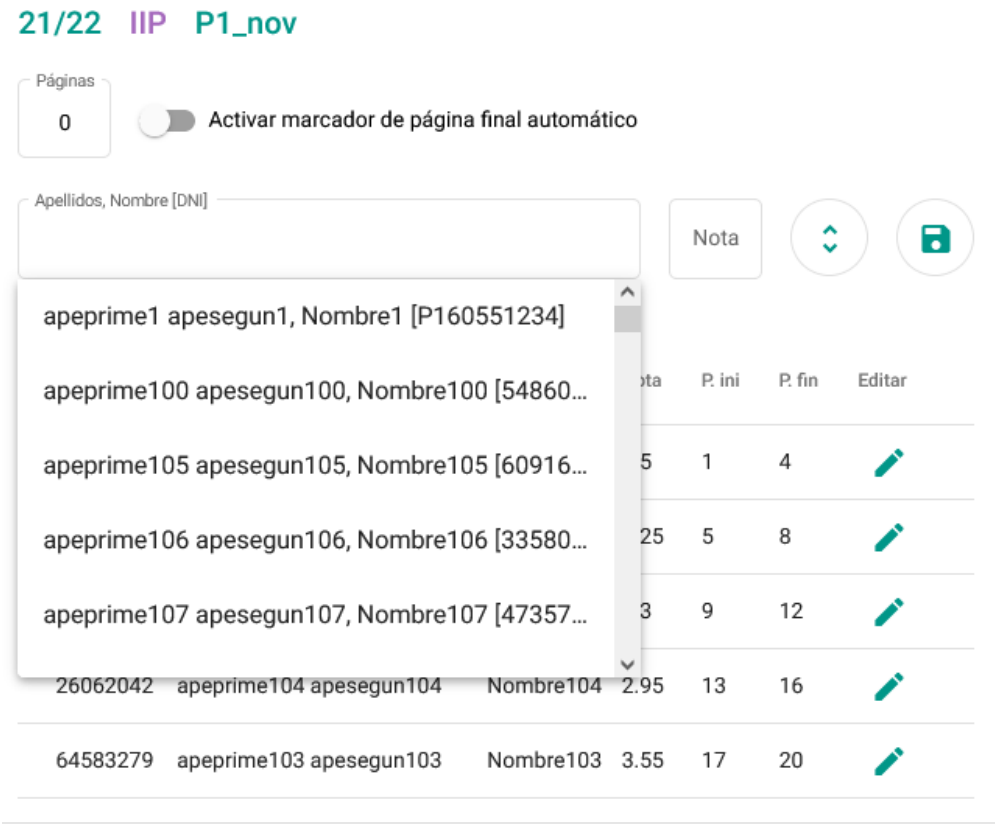


Figura 32 Sección de asociación de alumno y examen.

6.7.4. Proceso de asociación

Para asignar a un alumno su examen se diseña un proceso específico. Todas las acciones que se verán a continuación son posibles mediante el uso del ratón, pero el diseño está pensado y enfocado a que el usuario controle todo el proceso con el teclado para agilizar y facilitar la tarea.

Primero se identifica el nombre del alumno en la primera página del examen que se puede ver en el visor, usando, si es necesario, el zoom. La página inicial del examen se asigna automáticamente a la primera página del PDF subido por el profesor, o a la página siguiente de la página final del último examen asignado a un alumno. Así la página inicial siempre se asigna de forma automática, aunque puede ser modificada por el profesor. A continuación, el profesor busca el nombre del alumno en el cuadro de texto. El alumno se puede buscar por nombre, apellidos o número identificativo indiferentemente. Una vez se empiece a escribir, se despliega una lista con todos los alumnos que coincidan con el texto escrito. Con las teclas de flechas arriba y abajo, y la tecla “intro” se selecciona el alumno. Para pasar al siguiente campo, introducción de nota, se usa la tecla de tabulador. La nota acepta tanto puntos como comas para separar los decimales. Con la nota introducida se pasa a la asignación de la página final usando nuevamente la tecla de tabulador. Si la asignación de la página final está en automático, esta se

actualizará con el valor correspondiente. Si el número de páginas de los exámenes es variable, usando la tecla de “intro” o las teclas de flecha arriba y abajo se desplaza el visor hasta estar en la última página del examen. Cuando se tenga posicionado, se tabula para llegar al botón para guardar. Una vez se guarda, el alumno aparecerá en la tabla de abajo, además los campos de texto se borran (nombre y nota) y el visor avanza automáticamente a la siguiente página, seleccionando a su vez la página inicial del examen (y en caso de que esté activado el indicador que define un mismo número de páginas para todos los exámenes, la página final). Todo este proceso se puede ver indicado en la captura de la Figura 33.

Cuando se edite la información de un alumno, los campos se rellenarán con los datos del alumno y el visor pasará a la página inicial de su examen. Aquí se modifica lo necesario y una vez guardadas las modificaciones, la aplicación pone el visor en la siguiente página para asociar, retomando así el trabajo.

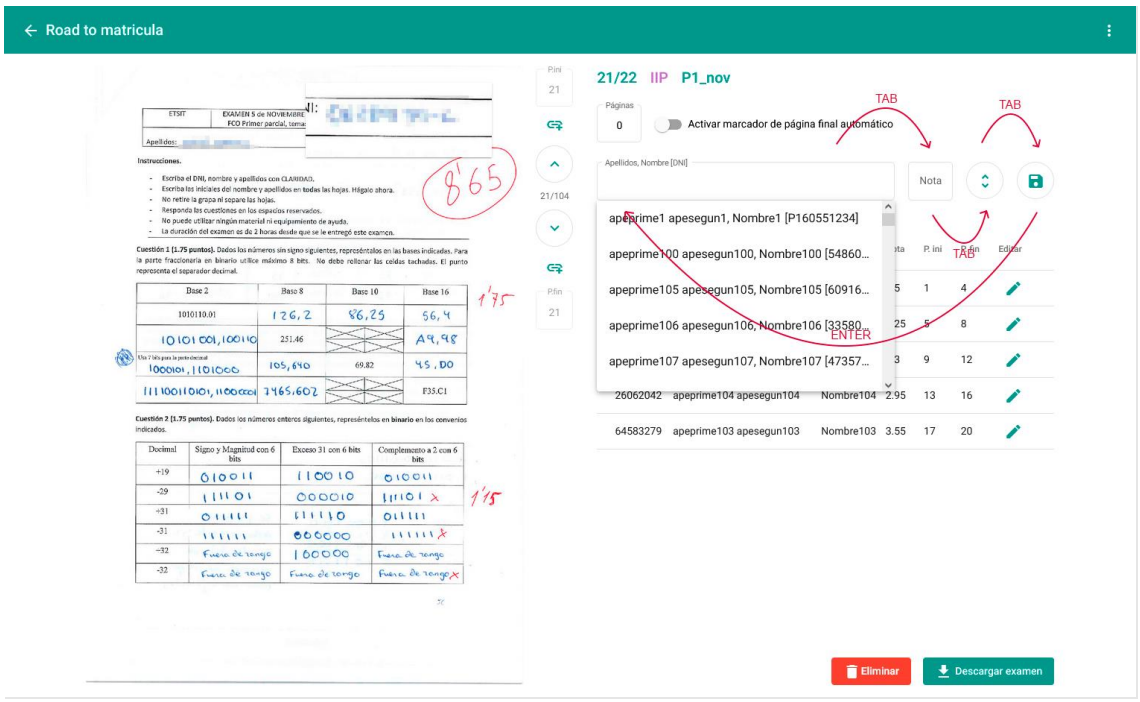


Figura 33 Proceso de asociación alumno-examen.

6.8. Exportación de información

La descarga del archivo comprimido con toda la información relacionada del examen debe realizarse de una sola vez, es decir, un único archivo que contenga todos los ficheros. Esta descarga debe estar disponible en todo momento, ya que, al mantener persistencia en cada asociación de alumno y examen, los datos son guardados y pueden ser exportados en cualquier momento. Para su exportación primero se hace la solicitud del archivo comprimido, dando así tiempo al servidor para que procese todos los datos y genere el archivo. Una vez generado el archivo comprimido, el navegador solicita al servidor el archivo y se encarga de gestionar la descarga.

Entre la solicitud de descarga y la descarga se muestra una única cifra en porcentaje indicando el progreso de la generación del archivo. Los tiempos no suelen ser muy elevados, pero como se trabaja con archivos pesados se muestra un porcentaje para dar *feedback* al usuario dado que el proceso no es instantáneo.

6.8.1. Archivo comprimido

El archivo comprimido que se genera contiene: una capeta con el nombre del examen. Dentro de esta carpeta se encuentran dos carpetas más, una para las listas y otra para los exámenes. Esta última carpeta tiene como nombre la fecha de realización del examen y su nombre. Para evitar errores con los caracteres en blanco, en todos los nombres, tanto de las carpetas como de los archivos, el espacio se representa con guion bajo.

6.8.2. Lista de notas y detalles

Esta carpeta contiene dos listas, una con las notas y otra con los detalles de la asociación examen-alumno. El archivo que se sube a PADRINO tiene de nombre el nombre del examen seguido de "_notas_padrino.csv". Este archivo CSV cumple con el siguiente formato:

1. Nombre del archivo: "nombre del examen" concatenado con "_notas_padrino.csv".
2. Tiene cabecera (ID estudiante y Nota).
3. Los datos que muestra son: el número de identidad del alumno y su nota. En el número de identidad de los estudiantes que contienen letras, estás están en minúsculas, aunque PADRINO acepta también mayúsculas.
4. El carácter separador de columnas es el punto y coma.
5. Los decimales están separados por un punto, aunque PADRINO acepta ambos formatos.

La función del segundo archivo es tener toda la información descargada. Este archivo funciona como una lista legible para buscar y consultar datos. Sigue el siguiente formato:

- Nombre del archivo: "detalles.csv".
- Tiene cabecera (ID estudiante, Apellidos, Nombre, Nota, Inicio y Fin).
- Los datos que muestra son: el número de identidad del alumno, sus apellidos, el nombre, la calificación del examen, la página inicio y final respecto al documento con todos los exámenes.
- El carácter separador de datos es el punto y coma.

Todo esto se hace mediante la escritura de un archivo con texto plano con la codificación UTF-8, indicando en la extensión del archivo su tipo.

6.8.3. Exámenes individuales

Para subir los exámenes al espacio compartido se hace uso de un programa Java especializado, Platero. Para que este programa pueda leer correctamente los exámenes, el archivo debe tener un nombre concreto. El nombre empieza por el número de identidad del alumno, un guion bajo y el nombre del examen. Aunque no es necesario, para ayudar a la identificación se añade el nombre del alumno. Se mantiene la sustitución de los espacios en blanco por el guion bajo, pero se añade la simplificación de caracteres, es decir, si por ejemplo un carácter tiene acento, se le elimina, al igual que si está en mayúscula, se pone en minúscula. Esto último se hace para simplificar el texto y que no haya problemas a la hora de leer el nombre.

En caso de que un alumno no tenga nombre (por ser ilegible o no encontrado), el examen aparecerá con el formato "XXXXXXXXX_Sin_nombre_0.pdf" donde el 0 es un contador de exámenes que se encuentran en la misma situación.

La carpeta contiene todos los exámenes de los alumnos de manera individual. El documento es generado a partir del documento padre, el archivo único que contiene todos los exámenes y que está subido al servidor.

6.9. Seguridad

El proyecto surge de la necesidad del profesorado de la UPV, por tanto, se entiende que su uso es principalmente para esta entidad y dentro de sus servidores. Las preocupaciones de seguridad podrían delegarse a estos servidores, sin embargo, como buen desarrollo, debe permitir versatilidad, así que, además de contar con la seguridad que brinda la universidad a sus redes, la propia aplicación cuenta con sus métodos para mayor confianza.

También se mencionan los derechos de la Ley Orgánica de Protección de Datos (LOPD). Dado que, aunque se guarden los datos sensibles de manera cifrada, hay que permitir al usuario informarse sobre esto. Se muestra en la pantalla de "Iniciar Sesión" al ser la primera mostrada en la aplicación web y la que siempre se debe visitar.

6.9.1. Cifrado de correo y contraseña

El primer punto que tratar sobre la seguridad en la aplicación son los datos sensibles del usuario, es decir, el correo electrónico y la contraseña. Estas se guardan en la base de datos de manera cifrada. Para esto, se usa la biblioteca CryptoJS[6] que contiene los estándares de cifrado [8]. El cifrado se hace con el algoritmo de *hashing* SHA-3. Se aplica el algoritmo a la concatenación de una clave privada con la información del usuario que se desea cifrar, como resultado se obtiene

[6] https://cryptojs.gitbook.io

una cadena de texto cifrada de 256 bytes, texto que se guarda en la base de datos en sus correspondientes campos de correo y contraseña.

6.9.2. Cambio de contraseña

En caso de que el profesor quiera cambiar la contraseña, este únicamente tendrá que introducir su correo. Su correo se comparará con el correo cifrado guardado y en caso de que este exista, se le enviará un correo para solicitar el cambio de contraseña.

Este correo contiene una dirección URL a la aplicación web. Esta dirección corresponde con la ruta de la página para cambiar la contraseña. La ruta tendrá dos parámetros en la URL, uno indica el ID del usuario y otro con una cadena de texto aleatoria. Cuando se solicita el correo, se inserta en la base de datos la solicitud con el usuario y esta cadena, por tanto, para cambiar la contraseña se necesitan dos códigos aleatorios que solo son mostrados en el correo. Cuando se entra a la página para cambiar la contraseña, se comprueban las credenciales de la URL y posteriormente permite hacer la operación. Cuando se confirma el cambio, la solicitud se borra caducando el correo.

6.9.3. Autenticación en las consultas

Con respecto al servidor web que aloja la API REST y la API Sockets, también requiere de cierta autenticación. La API (API REST) está dividida en dos tipos de rutas, las que están relacionadas con el inicio de sesión y las que permiten acciones y consultas. Las rutas relacionadas con el inicio de sesión no necesitan de autenticación. La consulta para el inicio de sesión es la única que asigna al usuario una cadena de texto aleatoria de 64 bytes, cadena de texto que valida el inicio de sesión del usuario y que es necesaria para todas las consultas de la lógica. Es decir, y siguiendo con el otro tipo de rutas, todos los métodos son POST porque en el cuerpo llevan el objeto del usuario que contiene la cadena de texto que se le asigna cuando inicia sesión (y que se borra cuando la cierra). Esta cadena de texto se comprueba para cada petición que se hace al servidor API

6.9.4. Descarga de archivos en servidor

Para evitar consultas ajenas, solo se procesan métodos POST a excepción de dos casos, dos tipos de petición GET. Esto ocurre cuando el usuario quiere descargar la información del examen. Es ahí y únicamente ahí, cuando el archivo se aloja en una carpeta publica (pero con un nombre compuesto por identificadores aleatorios) y el cliente hace una petición GET para descargar el archivo ZIP, un archivo que estará disponible solo cuando el cliente haga la petición. El otro caso, es una carpeta pública que aloja una página HTML simple y se permite hacer una petición GET a esta página que contiene una guía para usar la aplicación.

Como las demás peticiones GET se ignoran, para mostrar las páginas de los exámenes, se usan los Sockets. Estos leen las imágenes, extraen los bytes, los convierten a texto y se fragmentan las cadenas de texto para enviárselas al cliente. De esta manera, el cliente recrea las imágenes de forma segura para poder mostrarlas en el visor de exámenes. Así, en ningún momento, las imágenes quedan expuestas para cualquier otro usuario.

7. Implementación y tecnologías

Como se ha dicho antes, se usa una arquitectura cliente servidor. En este TFG se ha realizado un desarrollo desacoplado, es decir, como si se tratara de dos aplicaciones independientes. En realidad lo son, pues podría sustituirse el cliente o el servidor por otra aplicación diferente.

En este capítulo se describen, por separado, las tecnologías y estructura del cliente y del servidor.

7.1. Cliente (*frontend*)

Actualmente el mercado de *frameworks* para desarrollo *frontend* lo lideran Angular[7] [9] y React[8] [10].

React mantiene las mismas características que Angular, pero tiene un enfoque de desarrollo diferente. Ambas son webs dinámicas, con el patrón MVC y multiplataforma. Esta biblioteca de JavaScript [11] está diseñada para enfocarse en la interfaz gráfica. El objetivo de este *framework* son aplicaciones para mostrar datos con la posibilidad de mostrar un diseño gráfico e interactivo muy trabajado.

Angular, sin embargo, tiene un objetivo más orientado a la funcionalidad y tratamiento de datos. Se divide en componentes, usa TypeScript [12] danto robustez a la aplicación y permite el enlace de variables entre el HTML y el código. Además, mantiene la esencia y las funcionalidades de JavaScript a pesar de incluir el tipado fuerte.

Tanto React como Angular son opciones válidas, pero se ha escogido Angular, el *framework* mantenido por Google. Se usa Angular con su herramienta Angular CLI para desarrollar la aplicación con sus respectivas funciones de crear, depurar y publicar de manera sencilla. Angular está preparado para ser multiplataforma, pero de manera genérica, es decir, se puede desarrollar tanto clientes web, aplicaciones móviles nativas o incluso aplicaciones de escritorio con la ayuda de otros *frameworks* como Electron[9] [13]. En este caso, el uso corresponde a un cliente web. Los navegadores web interpretan JavaScript de manera nativa, TypeScript es un lenguaje que tiene de base JavaScript. Para sacar una versión de producción de Angular, hay que compilar el proyecto y en este paso TypeScript es compilado para poder ser interpretado como código JavaScript permitiendo que cualquier navegador ejecute la aplicación.

[7] https://angular.io/
[8] https://reactjs.org/
[9] https://www.electronjs.org/

7.2. Servidor (*Backend*)

Se estudió la posibilidad de implementar un servidor en PHP[10] [14] usando el *framework* de Laravel[11] [15] para conseguir la misma facilidad a la hora de declarar las rutas de consulta. PHP es una de las tecnologías más usadas para desarrollar web, y una de las que cuenta con mayor antigüedad. Las desventajas que presenta son, por un lado, que no se le puede implementar Sockets, es decir, existen métodos para llegar a ese punto, pero lo único que hacen es simular esa conexión, PHP es un lenguaje estático que ejecuta scripts, por tanto, ese tipo de conexiones bidireccionales no son sencillas de implementar. Por otro lado, el lenguaje no es compatible de forma directa con JavaScript, existe el cambio de formato y otros métodos para interactuar con los datos, pero no es lo más cómo teniendo otras alternativas. El resumen rápido es que PHP no es una mala decisión, pero no ofrece ninguna ventaja y, por el contrario, si desventajas.

Lo más lógico es usar un *backend* con la misma tecnología que el cliente, es decir, JavaScript. Para ser más exactos, un lenguaje basado en TypeScript, pero la biblioteca para desarrollar un servidor en este lenguaje es relativamente nueva a la fecha de crear este trabajo, así que se prefiere algo más consistente. Un servidor basado en JavaScript da lugar al uso del *framework* Express. Existen alternativas como Meteor[12], pero su uso no se considera tan orientado ni tan ágil como Express a la hora de crear una API.

El servidor dispone de dos puertos de escucha. Por un lado, queda abierto el puerto 80 obligatoriamente, mediante el servidor web HTTP Apache (versión 2), donde se deposita el proyecto Angular compilado para que el servidor pueda proveer al cliente de la aplicación en todo momento. Por otro lado, se usa el *framework* Express. Este *framework* permite crear un servidor web usando JavaScript, ya que cuenta con el entorno de NodeJS[13] [16] para poder ejecutar el código fuera de los navegadores de manera local, es decir, como si fuera un programa de escritorio con todas sus características. Express pone a la escucha otro servidor con dos características: una API REST (no estándar debido a su definición particular de las URI[14]y métodos de petición) y una API Sockets para el alojamiento de conexiones bidireccionales TCP/IP. El resultado es poder programar una API sólida, rápida y sencilla donde se facilita el uso de las URI para las consultas y métodos de lógica, persistencia y de verificación del usuario. Los Sockets permiten un sistema de comunicación dinámico entre cliente y servidor donde cada uno puede enviar y recibir información sin necesidad de que el cliente envíe una petición al servidor y que el servidor le responda, como ocurre con los métodos HTTP de la API REST. El uso de esta tecnología viene motivado por las esperas y por el trabajo con archivos pesados de la aplicación. De esta manera el servidor puede estar informando al cliente de cómo va un proceso o, al contrario, el cliente puede enviar al servidor información de manera constante y dar en tiempo real constancia de cómo va el proceso. Otro punto positivo por el cual se usan los Sockets es poder usar la misma conexión para la API REST y Sockets. Express y la biblioteca de Sockets están preparadas para trabajar juntas, es decir, permite usar el mismo puerto de

[10] https://www.php.net/
[11] https://laravel.com/
[12] https://www.meteor.com/
[13] https://nodejs.org
[14] *Uniform Resource Identifier*. Localizador de recursos uniforme que puede ser una URL, URN o ambos.

escucha para establecer la conexión con la API de Sockets y a la vez, responder las peticiones de la API REST, por tanto, su implementación solo ofrece ventajas.

7.3. Estructura de la aplicación web

Como se ha mencionado anteriormente, la implementación se hace mediante dos proyectos independientes, uno para el cliente y otro para el servidor, cada uno con su estructura propia del proyecto. Además, se explica la manera de preparar estos proyectos para ejecutar en un contexto de uso fuera del entorno de desarrollo.

7.3.1. Estructura del cliente

Angular divide su estructura en tres archivos, el maquetado, los estilos y el código. El maquetado es definido mediante etiquetas con la estructura de HTML, pero permitiendo declarar *templates* y directivas que usa Angular para enlazar funciones y datos de la vista con el código. Para los estilos se usa el lenguaje CSS y sus derivados (SCSS y SASS), aunque cabe mencionar que los estilos predeterminados de Angular y Angular Material usan SASS al ser la manera de declarar estilos más complejos y versátiles. Finalmente, el código está escrito en TypeScript, un lenguaje con base de JavaScript, pero que incluye tipado fuerte, además usa los módulos de NodeJS para ampliar las funciones del lenguaje.

La estructura de carpetas pretende seguir el estándar que da Angular es su documentación y en sus ejemplos. La creación de un proyecto Angular define ya una estructura de carpetas y archivos.

Una vez se crea, en la raíz se encuentran varios archivos de Angular y dos carpetas, la correspondiente a los módulos de NodeJS y la carpeta donde se encuentra los archivos a escribir del proyecto. Esta raíz contine más bien la configuración, los paquetes de NodeJS, la versión del proyecto, etc.

En la carpeta “src” (donde se desarrolla la aplicación) se encuentran los archivos correspondientes a la base del proyecto y además otras carpetas. Una primera carpeta llamada “app” que contiene todos los componentes de la aplicación, desde el padre de todos, hasta el componente que controla las rutas y los que se crean para desarrollar la aplicación. Luego está la carpeta propia para archivos estáticos, una carpeta que pertenece a variables globales a las que se tiene acceso desde cualquier componente de la aplicación, una con los modelos de los objetos de la aplicación y finalmente con los servicios.

Los modelos son de especial importancia. Aquí es donde se declaran todas las clases de los objetos/modelos de la aplicación. Al ser un proyecto que usa TypeScript como lenguaje de programación, definir el tipo de las variables es de especial importancia. La variable se define o por tipo básico como un entero o un carácter, o por clases. Estas clases son definidas como los modelos de datos que tiene la aplicación, es decir, para crear un usuario con nombre y apellidos,

la manera correcta de hacerlo sería definir un modelo de datos que sea una clase usuario con sus dos atributos. Por consistencia, todas estas declaraciones se alojan en el mismo sitio.

Los servicios son clases que funcionan como *middleware*. Esto quiere decir que, por un lado, se tiene acceso a los servicios que ofrece tanto el *framework* de Angular y el navegador, como los servicios que ofrece el *backend* construido para la aplicación. En el proyecto, son solo dos clases que usan los servicios del navegador, únicamente los servicios de persistencia, es decir, servicios para guardar información en la memoria del navegador y a la cual se accede para autenticar la sesión del usuario. Los servicios de servidor están divididos en tres, el que accede a las rutas de lógica, el que accede a las consultas referentes a la sesión del usuario y el que establece la comunicación mediante Sockets.

7.3.2. Estructura del servidor

Express es un *framework* de NodeJS que no necesita de una estructura de archivos, esto significa que con instalar la biblioteca de Express en un proyecto NodeJS es suficiente. Sin embargo, tiene sus herramientas para ayudar al programador. El proyecto Express de la aplicación tiene una estructura base creada con el generador que tiene el *framework* y se mantiene su estructura principal, a pesar de ciertas modificaciones.

La estructura de carpetas y archivos comienza por una carpeta que contiene el script principal el cual crea y pone a la escucha tanto el servidor como los sockets. Este script importa la declaración tanto del servidor como de los sockets de otros archivos que están alojados en el proyecto. A continuación, hay dos carpetas, una pública que contiene una página básica que funciona como guía para la aplicación y otra que aloja el HTML que se usa al enviar el correo para cambiar la contraseña. Le sigue la carpeta de los módulos de NodeJS. Seguido está la carpeta de rutas, contiene los archivos que se encargan de procesar las consultas de API y los eventos de los Sockets. Hay una carpeta que pertenece al renderizado de información en caso de consultas a rutas sin definir, o fallos en la aplicación. Estos archivos realmente no son necesarios, el proyecto está preparado para actuar dentro de un rango de rutas e ignorar las demás, sin embargo, estas declaraciones ante fallos no ocupan ningún coste ni carga adicional, por tanto, se dejan ya que fueron generados por Express y no tuvo coste de desarrollo. En la carpeta raíz, solo hay dos scripts de código enfocado a NodeJS, uno que contiene la declaración del servidor, es decir, este archivo crea y configura el servidor, y define las rutas y la comunicación de los Sockets que necesita el archivo principal para lanzar el servidor y un segundo script que pertenece a las credenciales de la base de datos y sus métodos para acceder tanto de forma síncrona como asíncrona. Finalmente, el proyecto crea en tiempo de ejecución dos carpetas, una pública, donde se alojan los archivos comprimidos de los exámenes en caso de que un usuario haya solicitado su descarga, de manera que el navegador hace una petición GET y se deja que el propio navegador gestione la descarga, y otra carpeta privada donde se guardan los archivos de los exámenes.

Las URI o rutas están diferenciadas en dos tipos, las de lógica (API REST) y las de sesión, sin embargo, en la carpeta de rutas se encuentran cuatro archivos. El servidor define cuatro tipos de rutas. Dos rutas que corresponden a las carpetas públicas, una para la guía de la aplicación y otra para la descarga de los archivos comprimidos. Otras dos que pertenecen a los dos archivos

con las rutas de la lógica y las rutas relacionadas con la sesión del usuario. También se define un intermediario para las rutas de la lógica. Antes de definir la API REST se define una ruta intermediaria que se ejecuta antes de acceder a una ruta de la lógica, esta ruta no tiene nombre y es genérica, quiere decir que todas las consultas que vayan a la lógica primero tendrán que pasar por esta ruta genérica que valida la sesión de usuario y en caso de que sea válida, permite continuar la consulta. Finalmente, en la carpeta de rutas está la declaración de los eventos que escucha el Sockets por el mismo puerto que el servidor web de la API REST.

Cabe mencionar que NodeJS ejecuta código JavaScript, un lenguaje que tiene como característica las funciones asíncronas que devuelven *callbacks* (función que se pasan como argumento). Por ejemplo, una función que lee una carpeta es llamada con la ruta de la carpeta y un *callback*, cuando el programa haya accedido al directorio y leído la información, ejecuta el *callback*. Esto afecta a la estructura interna del código ya que no se puede basar enteramente en la asincronía, las funciones del servidor siempre requieren de consultas a base de datos y estas consultas funcionan con *callbacks*. El problema es que la asincronía no siempre es una ventaja, a veces una consulta depende de la información extraída de otra. Para solucionar esto se usa la estrategia de "async/await" que proporciona JavaScript. Combinándola con las bibliotecas usadas para crear el servidor se obtiene el control para decidir ejecutar un código de manera secuencial o asíncrona.

Por último, y en relación con la asincrónica de JavaScript se menciona cómo se ha procesado los documentos PDF. Estos documentos son pesados y por tanto su procesado es lento. Para poder optimizar su procesado se usan comandos de *Shell* [17] [18]. La lista completa de comandos, *scripts* y herramientas utilizadas se muestra en una sección posterior de esta memoria.

NodeJS tiene módulos para lanzar un proceso de terminal en Linux mediante un proceso hijo. El problema es que se necesita que esta función sea síncrona. Dentro del módulo que permite esta función, tiene la opción síncrona, pero esta pausa el proceso padre hasta que termina el proceso hijo, el resultado es el esperado, sin embargo, al pausar el proceso padre las conexiones con el cliente se pierden y se deja de informar sobre el progreso. Por tanto, la función asíncrona se declara dentro de una *Promise*, el objeto que permite la computación asíncrona, pero devolviendo el resultado cuando se indica. La estrategia para convertir la función asíncrona de *Shell* en síncrona es declarar esta *Promise* como objeto resultado de una función con la etiqueta *async*, objeto que lanza la orden de terminal y notifica cuando esta termina, de manera que esta función pasa a ser síncrona ya que ahora permite que el proceso espere a que la función termine a pesar de ser asíncrona al tener la etiqueta *async*.

7.3.3. Despliegue del cliente

Para desplegar un proyecto Angular, primero hay que compilarlo. Angular tiene una gran estructura de carpetas y archivos, además de ser programado en TypeScript, esto se traduce en que el navegador no es capaz de ejecutar sus archivos de manera nativa sin ayuda de otras herramientas. Para lanzarlo a producción, el proyecto se compila y se comprime todo en apenas tres archivos HTML, CSS y JS, usando así las tecnologías estándar en los navegadores y dándole la máxima compatibilidad al proyecto. Una vez compilado, la aplicación está lista para

usar. Para desplegarla se usa un servidor web donde se depositan estos archivos, de manera que el cliente cuando accede a la URL de la aplicación lanza una petición GET del index.html de la página (archivo que genera Angular) y posteriormente a los otros objetos, mostrando así una aplicación web lista para usar.

7.3.4. Despliegue del servidor

El despliegue del servidor es muy simple, hay que instalar los paquetes de NodeJS y ejecutar el programa en un sistema Linux. Para conectar con la base de datos se debe modificar las credenciales de acceso en el archivo "connection.js" poniendo las correctas (usuario, puerto, contraseña y nombre de la base de datos). Hay que tener en cuenta dos cosas, por un lado, el servidor se cerrará en caso de que no consiga conectarse con la base de datos, ya que en tal caso no podría realizarse ninguna de las funciones incluidas en la aplicación.

7.4. Base de datos

La persistencia queda a cargo de un sistema de gestión de bases de datos relacional, MySQL. Se usa una base de datos relacional porque la información de la aplicación depende únicamente del usuario y de este descienden todas las relaciones. MySQL tiene licencia pública y su uso es muy habitual, contando esto y que la aplicación tiene un número muy pequeño de tipos de objetos a guardar, es el que se considera más conveniente.

Se ha instalado la versión MySQL 8.0, pero podría usarse cualquiera de sus versiones compatibles. Se necesita un usuario el cual usa el servidor para conectarse y hacer las consultas, también hay que hacer la definición de las tablas.

7.5. Servicio de correo (GNU Mailutils)

A la hora de cambiar la contraseña, el modo más seguro al que se tiene acceso es mediante el uso del correo electrónico. Para enviar correos se necesita de un servicio externo. Aprovechando el uso del sistema Linux se usan las herramientas ofrecidas por el paquete GNU Mailutils[15]. Este paquete dispone de un conjunto de funciones que permiten gestionar y enviar correos electrónicos.

NodeJS permite ejecutar órdenes *Shell* como si fueran lanzadas desde la terminal de Linux como se ha dicho en el procesado de los documentos PDF, pero en este caso se simplifica y se hace uso de su función asíncrona. El comando que se usa es "mail". Este comando es entubado

[15] https://mailutils.org/

con el texto HTML que se mostrará en el correo, además mediante las opciones se indica que el contenido está en formato HTML, el asunto del correo, el correo que se usa para enviarlo y el correo del destinatario. Se prepara el comando como cadena de texto y se lanza una función de NodeJS con ese texto, esto crea un proceso hijo que ejecuta la cadena de texto como un comando en la terminal, de manera que ejecuta la orden *Shell* enviando el correo.

7.6. Lanzar a producción, despliegue

La aplicación está basada en la arquitectura cliente-servidor, por tanto, hace falta un servidor. El servidor debe estar basado en Linux y tener un acceso público, ya sea por IP o por dominio. Para esto, la universidad, en concreto el departamento DISCA, puso a disposición una máquina virtual "corex-tfm" con el dominio público (dentro de la red universitaria) "corex-tfm.disca.upv.es".

A está máquina se le instaló las dependencias necesarias para el proyecto: Apache, GNU Mailutils, GraphicsMagick[16], MySQL y NodeJS. Con Apache instalado, se depositan los archivos compilados del proyecto Angular en su carpeta visible, en la ruta "/var/www/html/". Se debe tener en cuenta que para esto se usa la configuración predeterminada de la instalación de Apache, es decir, la escucha por el puerto 80 y la carpeta predeterminada que Apache hace visible.

El proyecto Express (servidor) no es necesario compilarlo o ejecutar algún procedimiento específico para lanzarlo. Simplemente se ejecuta el proyecto con su orden "npm start" dentro de la carpeta, sin importar donde esté ubicada esta. Pero antes, hay que tener la base de datos montada. Para preparar la base de datos, hay que crear las tablas necesarias y tener preparado un usuario por el cual conectarse. Una vez preparado, se indican las credenciales en el proyecto Express del *backend*, en un archivo llamado "connection.js" donde en las primeras líneas se ven los datos básicos que hay que cambiar para conectarse a la base de datos (*host*, *user*, *port*, *password*, *database*). Además, para que el servicio del correo funcione, hay que indicar el dominio donde se encuentre la máquina. En el proyecto Express, dentro de la carpeta "mail", está el archivo "config.json", en este se debe indicar el dominio público del servidor y el asunto que tendrán los correos que se envían.

Finalmente, a modo de resumen, hay que seguir los siguientes pasos: crear la base de datos con sus tablas y activarla, modificar las credenciales e indicar el dominio del servidor en el proyecto Express y ejecutarlo para poner a escuchar al servidor, depositar los archivos resultado de la compilación del proyecto Angular en un servidor web configurado en el puerto 80 para que el cliente tenga acceso al *frontend*.

[16] http://www.graphicsmagick.org/

7.7. Bibliotecas, *frameworks* y herramientas

A continuación, se muestra la lista de tecnologías, bibliotecas y otro tipo de herramientas usadas para el desarrollo de la aplicación con una breve explicación de su uso.

Cliente:

- Angular CLI[17] (v13.1.3 con NodeJS v12.20.0). [*Framework*] Herramientas para crear un proyecto Angular.
- Angular Material UI[18]. [Biblioteca] Es una biblioteca de estilos que facilita tanto funciones como estilos para los componentes. Tiene como línea de diseño los estilos de Material, es decir, el diseño gráfico de Google.
- File-saver[19]. [Biblioteca] Permite descargar archivos desde un servidor. Con esto se puede descargar un archivo desde un servidor sin necesidad que el usuario tenga que darle a un link para descargar. Además, Angular no permite esto de forma sencilla por seguridad.
- Ngx-socket-io[20]. [Biblioteca] Permite el uso de sockets en Angular.

Servidor:

- Apache (v2). [Servidor web] Servidor web que mantiene abierto las peticiones al puerto 80, es decir, peticiones desde un navegador al servidor.
- Archiver[21]. [Biblioteca] Permite generar un archivo ZIP con carpetas y archivos. Uso con NodeJS.
- Crypto-js[22]. [Biblioteca] Permite una gran cantidad de cifrados. Uso con NodeJS.
- Express (con NodeJS v12.20.0). [*Framework*] Entorno para la implementación de un servidor HTTP con API. Uso con NodeJS.
- GNU Mailutils. [Herramienta] Permite enviar correos electrónicos mediante el Shell de Linux. Uso en Linux.
- GraphicsMagick. [Herramienta] Permite trabajar con archivos PDF e imágenes. Uso en Unix.
- Mysql2[23]. [Biblioteca] Permite conexión y consultas a base de datos. Uso con NodeJS.
- Ps2pdf[24]. [Herramienta] Permite combinar las capas del documento, como los dibujos o comentarios que están en diferentes capas y dando como resultado una única capa. Uso en Linux.
- Pdfunite[25]. [Herramienta] Permite concatenar una lista de ficheros PDF para generar un único fichero. Uso con Linux.

[17] https://angular.io/cli
[18] https://material.angular.io/
[19] https://www.npmjs.com/package/file-saver
[20] https://www.npmjs.com/package/ngx-socket-io
[21] https://www.npmjs.com/package/archiver
[22] https://www.npmjs.com/package/crypto-js
[23] https://www.npmjs.com/package/mysql2
[24] https://linux.die.net/man/1/ps2pdf
[25] https://manpages.ubuntu.com/manpages/impish/man1/pdfunite.1.html

- Pdf-merger-js[26]. [Biblioteca] Permite concatenar documentos PDF, así como extraer páginas. Uso con NodeJS.
- Pdfjs[27]. [Biblioteca] Permite trabajar con PDF y es dependencia de pdf-merger-js. Uso con NodeJS.
- Pdftoppm[28]. [Herramienta]. Permita generar imágenes de las páginas de un archivo PDF. Uso en Linux.

Base de datos:

- MySQL. [Base de datos] Sistema gestión de base de datos relacional para la persistencia de datos.

[26] https://www.npmjs.com/package/pdf-merger-js
[27] https://www.npmjs.com/package/pdfjs
[28] https://linux.die.net/man/1/pdftoppm

8. Pruebas

Para confirmar el correcto funcionamiento de la aplicación se realizan las pruebas necesarias con los diferentes datos que pueden interactuar con las funciones de la aplicación. Los resultados de todas las pruebas son satisfactorios de manera que se asegura cumplir el objetivo de la aplicación.

8.1. Plataforma

A la aplicación se accede desde diferentes navegadores, Firefox, Chrome, Safari, etc. Al ser una aplicación web donde el tratamiento de datos se realiza en el servidor, el navegador solo necesita poder interpretar código JavaScript para que funcione. El cliente, al ser compilado a este lenguaje permite ser interpretado por cualquier navegador. Sí es verdad que el diseño está pensado para el uso en un monitor o una proporción de pantalla de 16:9, sin embargo, la interfaz gráfica es correcta hasta un mínimo de 1400 píxeles de ancho.

8.2. Acciones

Se comprueba que las acciones de la aplicación realicen su función de manera correcta. Para realizar estas pruebas simplemente se observa si al pulsar en el botón se realiza la acción enlazada. La comprobación de estas funciones se reduce a los botones no relacionados con los formularios, dado que sus acciones no son directas, es decir, se realizan comprobaciones previas a los datos y las acciones relacionadas con la asociación de alumno y examen. Se lista las acciones probadas para este apartado:

- Retroceder página (Botón de retroceso de la barra de navegación).
- ¿Has olvidado tu contraseña?
- Menú: Cambiar correo.
- Menú: Información.
- Menú: Cerrar sesión.
- Nuevo curso.
- Ordenar por fecha: Ascendente y descendente.
- Ordenar por nombre: Ascendente y descendente.
- Ocultar y visualizar curso.
- Crear asignatura.
- Menú: Crear examen.
- Menú: Editar asignatura.
- Menú: Eliminar asignatura.
- Visualizar exámenes de una asignatura.

- Descargar examen.

Todos estos botones realizan su función correctamente.

8.3. Entrada de datos (formularios)

A continuación, se listan los formularios de aplicación. En este caso, la lista de formularios se centra en la entrada de datos básicas, es decir, una palabra o texto. Más adelante se verán las entradas de datos más complejas.

- Iniciar sesión: Nombre de usuario y contraseña.
- Crear tu usuario: Nombre de usuario, correo electrónico y contraseña.
- Solicitar cambio de contraseña: correo electrónico.
- Cambiar contraseña: contraseña y repetir contraseña.
- Cambiar correo: correo electrónico.
- Crear una nueva asignatura: Nombre de la asignatura y acrónimo (la entrada del CSV se verá más adelante).
- Editar la asignatura: Nombre de la asignatura y acrónimo.
- Nuevo examen: Nombre del examen, fecha del examen y documentos PDF con los exámenes.
- Eliminar asignatura.
- Eliminar examen.

Para estas entradas de datos, se comprueba siempre el mínimo y máximo de caracteres permitidos por la base de datos. Para cada tipo de dato se comprueba sus restricciones, es decir, para el nombre de usuario se comprueba que no esté repetido, como para el correo electrónico que se comprueba que pertenezca al dominio indicado. En caso de que el dato no sea válido la acción es rechazada o no lanzada. Si son válidos los datos introducidos, se realiza la acción correspondiente. En el caso de eliminar un dato, las comprobaciones se hacen desde la base de datos, es decir, se comprueba la posibilidad de eliminar. En caso de que haya datos sensibles asociados, no lo permite.

En el caso del envío del correo electrónico para el cambio de contraseña se comprueba, por un lado, el envío del propio correo y, por el otro lado que, al acceder a la página del correo, este sea aún válido.

8.4. Lectura de un archivo CSV

Al subir una lista de alumnos, se prueban los dos formatos de CSV ya vistos, pero para dar mayor versatilidad se crean archivos CSV con otro tipo de formato, cambiando el carácter separador o combinando las columnas de diferentes maneras, así se asegura que la aplicación es capaz de leer una amplia variedad de formatos.

Se generan archivos CSV con los diferentes caracteres de separación que acepta la aplicación. También archivos con y sin cabecera. Se cambian las codificaciones en los permitidos por el lector. Finalmente se comprueba que la lectura del archivo corresponde con el formato introducido.

Por otro lado, se hacen pruebas con las combinaciones posibles de los datos dentro de las columnas, ya sea juntando nombre y apellidos en una misma columna, dejando todos los datos separados u otros tipos de combinaciones permitidas.

Para todas las pruebas realizadas la aplicación responde de la manera esperada.

8.5. Procesado de exámenes

Para el procesado de los documentos se realizan pruebas con exámenes reales que se documentan en la siguiente tabla. Entre estos documentos se encuentran diferentes tamaños de archivo, diferentes números de páginas, varios documentos para subir y documentos con y sin comentarios del profesor.

Nombre	**Archivos**	**MB**	**Páginas**	**Tiempo**
Prueba 1	1	1	38	3s
Prueba 2	1	29	80	23s
Prueba 3	2	67	184	51s
Prueba 4	5	12	40	17s
Prueba 5	10	23	80	24s
Prueba 6	47	119	376	1m 34s
Prueba 7	1	120	376	57s
Prueba 8	1	67	184	49s

El procesado de exámenes es correcto. Sin embargo, para poder optimizar el trabajo sobre los documentos se hace uso de hilos y varios procesos. Este tipo de programación es delicada y crítica, por tanto, aunque las pruebas sean satisfactorias, la aplicación cuenta con un sistema de gestión de errores. El sistema comprueba que cada fase del procesado se haya realizado y en caso de que hubiese un fallo la creación del examen quedaría descartada. Es decir, el procesado de exámenes se puede considerar como una operación atómica.

8.6. Proceso de asociación examen-alumno

Las pruebas para este apartado no se reducen únicamente al proceso directo de asociación, sino que también engloba las funciones de toda la pantalla del proceso. Esta pantalla no tiene complejidad a nivel de lógica, se basa en dar una buena experiencia de usuario.

Por un lado, se comprueba que todos los botones hagan su función asignada. También se observa que funcione tanto con el control del ratón, como con el uso del teclado. Decidir si su

función es correcta es tan fácil como comparar la acción con las tablas de requisitos, dado que todas las funcionalidades o maneras de navegar de esta pantalla son requisitos explícitos del profesorado.

Por otro lado, la experiencia de usuario solo se puede comprobar dejando que el profesor use la aplicación. Durante la fase de verificación del desarrollo, varios profesores estuvieron usando la aplicación para la asociación de alumno y examen en pruebas de evaluación reales. El resultado fue satisfactorio, ya que se dio el visto bueno del procedimiento y no hubo malas experiencias durante las asociaciones.

8.7. Descarga de un examen

La descarga está especificada al detalle, tanto el modo de descarga, la estructura del fichero comprimido e incluso el nombre de los archivos. Para realizar esta prueba, se descarga un examen en el que se ha completado la asociación alumno - examen. El archivo comprimido, su estructura de carpetas y los nombres de todos los ficheros, tanto carpetas como archivos son los especificados en los requisitos, por tanto, se da por valida la prueba de descarga.

9. Manuales de instalación y de usuario

Para instalar la aplicación se deben tener las siguientes consideraciones:

1. Instalar MySQL, se recomienda la misma máquina donde estará alojado el *backend*. En caso de fallo de versión, la aplicación está desarrollada con la versión 8.0. Se debe crear una nueva base de datos y ejecutar para esta un fichero SQL que se facilita con las instrucciones para crear las tablas. Se debe habilitar un usuario para conectarse desde *backend*.
2. Para la máquina que alojará el servidor, se debe comprobar que tenga instaladas las siguientes herramientas en un sistema Linux: GNU Mailutils, GraphicsMagick, Pdfunite, Pdf2ps, Pdftoppm. Estas herramientas suelen venir instaladas con las distribuciones de Linux, pero se debe verificar que están e instalarlas en caso contrario.
3. Instalar NodeJS en su versión 12 o compatibles. El proyecto del servidor es irrelevante donde se deposite. En el proyecto hay que modificar dos archivos. Para tener conexión con la base de datos hay que modificar el archivo "connection.js" a partir de la línea 3 e introducir las credenciales donde corresponda (*host*, *user*, *port*, *password* y *database*). Para permitir el envío de correos electrónicos se debe modificar el archivo "mail/config.json" e introducir el dominio del servidor, también se puede modificar el asunto del correo si se desea.
4. La ejecución del servidor se realiza mediante el comando en consola "npm start" dentro de la carpeta del proyecto *backend* o ejecutando el archivo "bin/www" con la orden "node".
5. Se instala Apache y se deja su configuración por defecto a excepción de indicar el nombre del dominio. Se vacía la carpeta "/var/www/html/" y se deposita los archivos del proyecto *frontend*.
6. A la aplicación se accede mediante un navegador con la IP o el dominio del servidor.

El uso correcto de la aplicación se explica en los siguientes pasos, teniendo en cuenta que el servidor está en un dominio de la Universitat Politècnica de València.

1. Acceder a corex-tfm.disca.upv.es (nombre del domino). Requiere VPN fuera de la red de la universidad.
2. Registrarse con un correo del dominio "upv.es" o "upv.edu.es". La contraseña no tiene restricciones.
3. Iniciar sesión.
4. Crear un nuevo corso con el botón "Nuevo Curso".
5. Añadir una asignatura con el botón "+".
   5.1. Se indica: nombre completo de la asignatura, acrónimo y un color asociado.
   5.2. Cargar lista de alumnos.
      5.2.1. Escoger un fichero de tipo CSV, generado por PADRINO o la Intranet. Como mínimo debe tener el DNI y el nombre completo. No importa ni el orden ni si hay más datos.
      5.2.2. El cuadro "CSV Texto" muestra el texto plano del fichero tras establecer su formato. Es importante buscar alumnos con acentos para escoger la codificación, también se debe indicar el separador de campos y si hay cabecera.

5.2.3. Los siguientes selectores permiten decir en qué columna está el DNI, primer apellido, segundo apellido y el nombre. Se debe seleccionar para cada campo la columna donde se encuentra su información sin importar si ya está seleccionada. El resultado se puede comprobar con la tabla inferior.

6. Añadir examen a una asignatura.
   6.1. Con la flecha hacia abajo del botón de asignatura se despliegan las opciones y se pincha en nuevo examen.
   6.2. Se da nombre al examen (se usará para generar los documentos PDF para los alumnos) y se indica la fecha de realización.
   6.3. Con el botón de cargar exámenes se sube uno o varios ficheros PDF con los exámenes escaneados. Se da la opción de reordenarlos, aunque no debería ser necesario.
   6.4. Tras crear el examen se debe tener en cuenta que el proceso puede tardar hasta 10 minutos. Si se cierra el navegador, el proceso no se interrumpe.
7. Clasificación de exámenes.
   7.1. Abre el examen mediante su botón.
   7.2. Hay un marco rectangular a modo de lupa que se activa, se fija y se desactiva pinchando dentro del visor. La función se mantiene, aunque se cambie de página.
   7.3. La barra vertical muestra (de arriba abajo), la página inicial del alumno, el botón para asignar esta página, un botón para retroceder de página, el indicador de página, un botón para avanzar la página, otro para marcar la página final y el indicador de la página final.
   7.4. El proceso eficiente para la clasificación está diseñado para hacerse mediante teclado:
      7.4.1. Se introducen las iniciales del apellido del alumno o DNI, con las flechas se selecciona el alumno de la lista desplegada y con el “intro” se selecciona. Con el tabulador se pasa al siguiente campo.
      7.4.2. Se introduce la nota usando indistintamente en punto o coma para separar los decimales. Con el tabulador se pasa al siguiente campo.
      7.4.3. Se busca a página final del examen del alumno usando las teclas “w”, “s”, “flecha arriba” o “flecha abajo”. Una vez localizada la última página, se usa el tabulador para pasar al siguiente campo.
      7.4.4. Pulsando “intro” se finaliza la asociación. Se añade al alumno en la tabla inferior.
      7.4.5. El foco pasa automáticamente al campo para introducir el nombre del alumno y se muestra la página siguiente del examen.
   7.5. Se puede cerrar la sesión y continuar más tarde.
   7.6. Se puede corregir cualquier asociación.
   7.7. Las asociaciones se descargan en un archivo ZIP. Contiene los documentos PDF con los exámenes de manera individual para cada alumno, un CSV con las notas para subir a padrino y otro CSV con toda la información de la asociación (DNI, nombre, apellidos, nota, primera y última página).

10. Conclusiones

Finalmente se consigue desarrollar la aplicación web que cumple con el objetivo por el cual se inicia el proyecto. La aplicación web ofrece una manera ágil y cómoda para cumplir los requisitos especificados. Requisitos que surgieron de la experiencia de los profesores tras haber usado otro tipo de aplicaciones o herramientas.

Todas las funciones de la aplicación son ágiles y ofrecen una experiencia de uso satisfactoria, desde crear una asignatura con su lista de alumnos hasta descargar los exámenes con toda su información asociada.

La asociación de examen y alumno permite al profesor realizar esta tarea de manera rápida, con grandes facilidades como la introducción del alumno o la opción de lupa del visor. Se consigue implementar una interfaz gráfica junto a un control de teclado capaz de hacer estas asociaciones únicamente con las teclas, evitando pérdidas de tiempo desplazando la mano entre el teclado y el ratón. También es posible dejar el proceso a medias y retomarlo más tarde, por lo que no se obliga al usuario a empezar y acabar con el trabajo en una única sesión de trabajo.

Las grandes dificultades que se encontraron en el desarrollo fueron dos: el diseño gráfico para la lectura de los archivos CSV y el procesado de documentos PDF. Por un lado, está el reto de diseño, crear una interfaz gráfica simple, fácil de usar. que permita al usuario leer una lista de alumnos sin la necesidad que esta lista tenga un formato especifico, es decir, lo suficientemente flexible como para que admita los diferentes formatos de archivos CSV que puede llegar a cargar el usuario. Por otro lado, está el reto que ha supuesto el procesado de documentos PDF. Se tuvo que estudiar en profundidad la mejor manera de manipularlos. Gracias al uso de NodeJS se podrían haber usado bibliotecas capaces de realizar todos los procesos requeridos, pero el uso de estas bibliotecas puede llegar a ser muy poco eficiente. La decisión de usar comandos de Linux lanzados en procesos hijos permite las velocidades a las que la aplicación trabaja, sin embargo, se dedicó muchas horas al estudio de cómo lanzar los procesos hijos, cómo controlar el estado y resultado del proceso, cómo afectan estos hijos a la pila de memoria y qué problemas pueden aparecer con la programación de multiprocesos, así como, el control de fallos que puede llegar a haber.

Ha sido una gran experiencia montar un proyecto desde cero, con su *frontend*, *backend* y su base de datos. Gracias a este trabajo se han adquirido grandes conocimientos de eficiencia a la hora de procesar archivos pesados, los diferentes métodos de comunicación que hay con el servidor y cuál es su mejor contexto de uso, la potencia de combinar programación síncrona con asíncrona, analizar al cliente de manera precisa para conseguir traducir sus requisitos en un *software* dedicado para el cliente, el uso de métodos de cifrado y de protección de datos y construir una base de datos relacional a partir del estudio de la estructura de datos de la aplicación, entre otras cosas.

10.1. Relación del trabajo desarrollado con los estudios cursados

Gracias a los estudios realizados se consiguieron las bases de todos los conocimientos utilizados en este trabajo, los cuales se han ido ampliando durante el transcurso de su desarrollo. Son significativas las asignaturas siguientes. Fundamentos de sistemas operativos que introduce el sistema Linux y sus comandos en terminal, Concurrencia y sistemas distribuidos habla de la jerarquía de los procesos, Computación paralela que permite entender los problemas de lanzar varios procesos hijos, Tecnologías de base de datos que ayuda a la creación de consultas complejas y eficientes, Redes de computadores para aprender cómo se comunican los clientes y servidores, Tecnologías de sistemas de información en red del cual se sacan las bases para implementar la comunicación bidireccional entre cliente y servidor, Estructura de datos, Interfaces persona computador, Desarrollo web y Desarrollo centrado en el usuario entre otras asignaturas útiles a la hora de entender lo que el usuario necesita e implementar una aplicación con una usabilidad satisfactoria.

11. Referencias

[1] J. Ferrandis Jorge, Desarrollo back-end en .NET de una aplicación web para el reconocimiento de DNIs en exámenes manuscritos y su clasificación para su posterior distribución personalizada, Universitat Politècnica de València, 2021.

[2] P. Kruchten, The rational unified process: an introduction, Addison-Wesley, 2003.

[3] G. Booch, J. Rumbaugh y I. Jacobson, El lenguaje unificado de modelado, Addison-Wesley, 1999.

[4] I. Sommerville, Ingeniería del software, Pearson, 2005.

[5] E. Mifsuf Talón, Apache, MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE - ÁREA DE EDUCACIÓN, 2015.

[6] E. M. Hahn, Express in Action: Writing, building, and testing Node.js applications, Manning, 2016.

[7] L. E. A. Ullman, MySQL, Peachpit Press, 2006.

[8] J. F. Kurose y K. W. Ross, Redes de computadores: Un enfoque desdendente, Pearson, 2017.

[9] A. Freeman, Pro Angular 6, Apress, 2018.

[10] K. Chinnathambi, Learning React, Addison-Wesley, 2016.

[11] R. Toal y J. D. N. Dionisio, The JavaScript programming language, Jones and Bartlett Publishers, 2010.

[12] B. Cherny, Programming TypeScript: Making Your JavaScript Applications Scale, O'Reilly, 2019.

[13] S. Kinney, Electron in action, Manning, 2019.

[14] D. Reiersøl, C. Shiflett y M. Baker, PHP in Action: objects, design, agility, Manning, 2007.

[15] M. Stauffer, Laravel: up & running: a framework for building modern PHP apps, O'Reilly, 2019.

[16] E. Brown, Web development with Node and Express: leveraging the JavaScript stack, O'Reilly, 2020.

[17] B. Ward, How Linux works: what every superuser should know, No Starch Press, 2015.

[18] R. Blum y C. Bresnahan, Linux Command Line and Shell Scripting Bible, Wiley, 2021.

12. Anexo

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

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**	**X**			
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**		**X**		
ODS 17. **Alianzas para lograr objetivos.**				X

En este trabajo se ha desarrollado una herramienta que permite devolver los exámenes corregidos a todos los estudiantes y en un tiempo reducido. Esta corrección incluye, como mínimo, la nota. Pero el profesor, al tener la seguridad de que el estudiante podrá ver esta corrección, puede decidir añadir comentarios sobre los errores cometidos, las carencias detectadas e indicaciones para mejorar. También comentarios de ánimo y en algunos casos de elogio.

Claramente este trabajo está altamente relacionado con el Objetivo de Desarrollo Sostenible 4, educación de calidad. Una calificación aporta información sobre los conocimientos que el estudiante tiene, pero no le ayuda a corregir sus carencias. Con este trabajo, se le facilita al estudiante subsanar sus carencias, sabiendo donde ha errado, al mismo tiempo que incrementa su motivación, lo que le ayudará a alcanzar las competencias necesarias para acceder a un empleo o al emprendimiento, tal como se indica en la meta 4.4.

Por otro lado, al recibir su corrección, el alumno es testigo y partícipe de un acto de transparencia de gran envergadura. La corrección de un examen es el juicio que emite un

profesor sobre los conocimientos y habilidades del estudiante, y con esta herramienta no solo se le proporciona al estudiante el detalle de este juicio, sino también la posibilidad de comparar la forma en que se le ha corregido frente a sus compañeros y si lo cree conveniente, solicitar mayor detalle sobre las razones de su calificación.

Por tanto, el estudiante vive un ejemplo de transparencia que le motivará a practicarla en su vida profesional, lo que se relaciona con el ODS 16, Paz, justicia e instituciones sólidas, y en concreto con la meta 16.6, que persigue crear instituciones, y en su sentido más amplio, empresas y profesionales transparentes y que estén dispuestos a rendir cuentas, no solo cuando se les exija, como cuando un alumno solicita revisión de examen, si no de forma constante y habitual.