ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE APLICACIÓN WEB PARA LA REALIZACIÓN DE CUESTIONARIOS ON-LINE EN EL AULA

DEVELOPMENT OF WEB APPLICATION FOR CONDUCTING ON-LINE QUESTIONNAIRES IN THE CLASSROOM

Realizado por

JOSÉ FRANCISCO FERNÁNDEZ FUENTES

Tutorizado por

SIXTO SÁNCHEZ MERINO

Departamento

MATEMÁTICA APLICADA

UNIVERSIDAD DE MÁLAGA,
MÁLAGA, SEPTIEMBRE DE 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: La aplicación desarrollada en este trabajo tiene como objetivo apoyar la labor docente en el aula, facilitando principalmente la realización de cuestionarios.

Permite al profesor gestionar clases virtuales en la que otros usuarios se inscriben como alumnos y almacenar en ellas preguntas y cuestionarios. El profesor puede posteriormente programar dichos cuestionarios, para que los alumnos los realicen de forma autónoma on-line, o llevarlos a cabo presencialmente en el aula. Tras la realización de cuestionarios, la aplicación ofrece al profesor los resultados obtenidos por los alumnos así como algunos datos estadísticos, además de la posibilidad de exportarlos. Además, la aplicación también posibilita mantener un listado de alumnos de la clase y conocer la participación de los mismos mediante la realización de controles de asistencia.

Se trata de una aplicación web, en cuyo desarrollo se han empleado los lenguajes PHP, JavaScript, HTML y CSS; se ejecuta en un servidor HTTP Apache y almacena la información en una base de datos MySQL. La aplicación ofrece una interfaz gráfica que se adapta a distintos dispositivos, empleando para ello el framework Bootstrap.

Palabras clave: Docencia, cuestionarios, asistencia, desarrollo web, diseño adaptativo, PHP, MySQL, JavaScript, HTML, CSS, Bootstrap.

Abstract: The application developed in this paper aims to support teaching in the classroom by simplifying the conducting of questionnaires.

It allows the teacher to manage virtual classrooms in which make questions and questionnaires for the users enrolled as students in the classrooms. The teacher can then schedule questionnaires, for students to perform them autonomously on-line, or carry them out in person in the classroom. After the completion of questionnaires, the application provides the results obtained by the students to the teacher as well as some statistical data and the ability to export it. In addition, the application also allows to maintain a list of students in the class and learn about the participation of the same by managing attendance registers.

Languages as PHP, JavaScript, HTML and CSS had been used in the development of this web application. It runs on an Apache HTTP server and stores the information in a MySQL database. The application provides a responsive graphical user interface which adapts to different devices, using the Bootstrap framework.

Keywords: Teaching, questionnaires, attendance, web development, responsive design, PHP, MySQL, JavaScript, HTML, CSS, Bootstrap.

Índice

1. Introducción

1.1. Objetivos

El objetivo fundamental de este trabajo es el desarrollo de una aplicación web para la realización de cuestionarios on-line que tengan un uso directo en la actividad docente en el aula.

En general, el propósito del software desarrollado es facilitar que el profesor envíe preguntas de respuesta instantánea para que sus alumnos puedan responder e, inmediatamente, se obtengan los resultados a modo estadístico.

Además, se buscaba implementar un módulo específico que permitiese aplicar dichos cuestionarios a la gestión de listados de asistencia de alumnos a clase, de forma que el profesor pueda activar controles de asistencia puntuales o programarlos, con los que los alumnos puedan confirmar su presencia física en el aula.

Adicionalmente, se establecieron una serie de requisitos que dicha aplicación debía tener:

- El profesor podrá crear un banco de preguntas.
- El profesor podrá crear cuestionarios programados o que las preguntas se lancen al azar.
- El profesor podrá controlar los tiempos de participación.
- El profesor podrá configurar la salida para los alumnos.
- La aplicación guardará todos los datos en formatos flexibles.
- La aplicación contará con una interfaz con diseño adaptable a distintos dispositivos.
- Se espera que la interfaz sea sencilla y cómoda de usar.

1.2. Motivación

Kahoot! es una aplicación web que permite crear y realizar juegos educativos basados en preguntas multirespuesta. En cada uno de estos juegos, llamados *kahoots*, otros usuarios responden a las distintas preguntas empleando sus propios dispositivos, mientras que las preguntas en sí aparecen en una pantallas compartida.

Kahoot! es una herramienta que se encuentra on-line, a la que los usuarios

acceden mediante un navegador web. No es necesario para su uso contar, por tanto, con servidores propios en los que deba ser instalada ni infraestructura o soporte de una institución educativa. Parece, además, sencilla de usar; una vez que se ha iniciado el cuestionario, el alumno accede al mismo mediante un código y la actividad comienza a realizarse.

El sistema en el que una pantalla sirve de guía mientras que otros dispositivos sirven a los usuarios para responder nos parece interesante. Entre otros aspectos, obliga a que los participantes se encuentren físicamente en el lugar en el que el cuestionario se está realizando, por lo que se puede aplicar a la actividad docente de forma presencial, para realizar actividades en el propio aula.

Sin embargo, la participación en los cuestionarios realizados con *Kahoot!* no requiere registro previo. En su lugar, los participantes eligen un nombre en el momento en el que la actividad se realiza, que sirve para identificarlos únicamente en dicha actividad.

Esto impide a un profesor poder llevar un registro de las actividades que pudiera realizar en un aula, donde el conjunto de participantes suele ser fijo a lo largo del curso académico. Para llevar un control de los resultados obtenidos en cada una de estas actividades, así como recolectar los resultados obtenidos por cada uno de los alumnos a lo largo del curso, hace falta que los participantes estén identificados de forma única.

Si los alumnos estuviesen agrupados en función de sus clases y los resultados obtenidos estuviesen siempre relacionados con cada uno de ellos, el profesor podría llevar un mayor registro del progreso de los alumnos y de los resultados obtenidos en la clase.

Esta es la motivación principal de este trabajo. Basándonos en una forma de realizar los cuestionarios que parece interesante, se busca una forma de adaptarla aún más al ámbito docente habitual. La aplicación incluirá otras características que la completen para su uso en el aula, como la realización y corrección de cuestionarios o el control de asistencia. Estas funciones ya están cubiertas por aplicaciones y sistemas de terceros con amplia aceptación y con años de desarrollo a sus espaldas,[1] por lo que en este trabajo no han sido una cuestión prioritaria. No se pretende sustituir estas herramientas, sino dar otra a medio camino entre ellas y *Kahoot!*, que combine la facilidad de uso de este último con un uso docente prolongado y no tan esporádico.

1 Pensamos, por ejemplo, en plataformas de aprendizaje como Moodle o la herramienta de evaluación de cuestionarios SIETTE.

1.3. Estructura de esta memoria

En primer lugar, se describirán aquellos elementos que se decidieron de forma previa al desarrollo en sí de la aplicación, como la metodología empleada o las herramientas que se decidieron utilizar. Seguidamente, se analizará el desarrollo de la aplicación a lo largo del tiempo, con cada una de las fases por las que ha pasado. Después se procederá a describir de forma más amplia las funcionalidades con las que cuenta la aplicación tras finalizar su desarrollo.

Por último, se expondrán una serie de conclusiones y se sugerirá distintas líneas de mejora de la aplicación.

1.4. Convenciones

En este documento se emplean las siguientes convenciones tipográficas:

Cursiva

Indica direcciones URL, archivos, directorios, rutas o extensiones. También puede indicar extranjerismos o términos que se desean resaltar.

`Monoespaciado`

Indica fragmentos de código fuente, clases, funciones, variables, parámetros, nombres de tablas o columnas de la base de datos.

2. Metodología

Para la realización del trabajo se ha seguido un método de desarrollo iterativo incremental, en el que el software desarrollado pasa por distintas etapas intermedias en las que se va completando incrementalmente la aplicación, hasta cumplir con los objetivos propuestos.

La especificación de la que se partió de base únicamente requería una serie de funcionalidades descritas a grandes rasgos, por lo que se disponía de cierta libertad siempre y cuando se cumpliese con ellas.

El margen de libertad con el que se contaba y la no imposición de detallados requisitos hacía prever que, a medida que se fuesen abordando cada una de estas funcionalidades, se descubrirían nuevos requisitos o se vería necesario incorporar nuevas funcionalidades con objeto de cumpir los objetivos propuestos. Por tanto, se descartó un desarrollo de tipo secuencial en los que los requisitos tuviesen que especificarse en un primer momento.

Se pensó por tanto en ir desarrollando la aplicación en etapas intermedias o iteraciones, en las que cada una de ellas se hiciese un pequeño análisis, se añadiesen funcionalidades nuevas y se fuesen refinando y completando las ya existentes en las etapas anteriores. Estas iteraciones se agruparon en fases más genéricas en las que, en base a los objetivos propuestos, se dividió el proyecto con objeto de realizarlo en orden, en base a las previsibles dependencias.

3. Tecnologías y herramientas empleadas

A continuación se citan las principales tecnologías y herramientas empleadas para la realización de este trabajo, junto con una breve descripción de cada una.

Servidor HTTP Apache

Apache es un servidor HTTP, de código abierto, desarrollado para los principales sistemas operativos modernos, entre los que se encuentran Windows y aquellos basados en UNIX como Linux o Mac OS.[Ref. 1] Desde 1996 hasta 2004 se trató del servidor web más empleado; desde entonces fluctúa entre la primera y la segunda posición junto a Microsoft.[Ref. 2]

MySQL

MySQL es un sistema de gestión de bases de datos relacionales, de código abierto, propiedad en la actualidad de Oracle Corporation. Se le considera el sistema de base de datos de código abierto más utilizado, y es uno de los más empleados a nivel mundial junto a Oracle Database y Microsoft SQL Server.

PHP

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje programación de scripting de propósito general y de código abierto[Ref. 3] originalmente diseñado para crear contenido web dinámico,[Ref. 4] es decir, contenido web generado automáticamente tras una consulta y que depende de distintas variables.

PHP es el séptimo lenguaje de programación más popular[Ref. 5] y se utiliza en el 81,8% de los sitios web que emplean lenguajes de programación del lado del servidor.[Ref. 6]

La versión de PHP empleada en este trabajo es la 5.6.8. Tendrá soporte activo hasta finales de 2016 y de seguridad hasta finales de 2018.[Ref. 7]

XAMPP

Es un paquete de soluciones para desarrollo web del lado del servidor, desarrollado por Apache Friends, que consiste principalmente en una distribución del

servidor HTTP Apache que contiene MySQL,[2] Perl[3] y PHP.[Ref. 8]

phpMyAdmin

Se trata de una herramienta de software libre desarrollada en PHP para facilitar la administración de bases de datos MySQL. Las operaciones más frecuentes pueden realizarse mediante una interfaz web a la que se accede con un navegador. [Ref. 9]

Con phpMyAdmin se ha creado la totalidad de la base de datos empleada en este trabajo.

HTML

HyperText Markup Language (lenguaje de marcado de hipertexto), abreviado como HTML, es un lenguaje de marcado empleado para la creación de páginas web. Describe la estructura semántica de una página web.[4] Es un estándar fijado por el World Wide Web Consortium (W3C). La última revisión recomendada es HTML5, a la que se ha intentado ajustar el código HTML desarrollado para este trabajo.

CSS

Cascading Style Sheets (hojas de estilo en cascada), abreviado como CSS, es un lenguaje de estilo empleado para representar la presentación visual de un documento escrito en un lenguaje de marcado. Es un estándar fijado por el World Wide Web Consortium (W3C).

JavaScript

JavaScript es un lenguaje de alto nivel interpretado, dinámico y no tipado. Se emplea en la mayoría de los sitios web modernos, y en la actualidad los principales navegadores web incluyen intérpretes de JavaScript.[Ref. 10] Permite sobrealimentar el lenguaje HTML con animación, interactividad y efectos visuales dinámicos en el lado del cliente.

Se ha empleado junto a jQuery, una librería de JavaScript diseñada para

2 A partir de las versiones 5.5.30 y 5.6.14 XAMPP emplea MariaDB, un *fork* de MySQL, en lugar de este último. No obstante, para la realización de este trabajo se ha optado por una versión anterior de XAMPP que sí incluye MySQL.

3 El lenguaje de programación Perl, aún incluido en XAMPP, no ha sido empleado en este trabajo.

4 HTML también incluía originalmente atributos para representar la visualización del documento. La mayoría de estos atributos se encuentran hoy obsoletos y el W3C recomienda alternativas como CSS en su lugar.

facilitar el uso de dicho lenguaje.[Ref. 11]

Bootstrap

Se trata de un *framework front-end* para el diseño de páginas y aplicaciones webs, que incluye plantillas con código HTML y CSS para realizar botones, barras de navegación, formularios u otros componentes de una interfaz gráfica.

Bootstrap ha facilitado la creación de la interfaz, especialmente su adaptación a distintos dispositivos. No obstante, algunos elementos de Boostrap se han adaptado empleando hojas de estilos personalizadas.

Git

Se trata de un software de control de versiones. El control de versiones es un sistema que almacena los cambios realizados en ciertos archivos a lo largo del tiempo, de modo que se puedan recuperar a un estado anterior.[Ref. 12]

4. Desarrollo temporal

4.1. Temporización inicial

En base a los objetivos propuestos, inicialmente se previó la existencia de una serie de fases generales ordenadas según las dependencias que previsiblemente existían entre ellas. Para cada fase, además, se estimó su duración aproximada, resultando la duración total del trabajo en 12 semanas.

- Gestión de usuarios (1 semana)
- Cuestionarios (2 semanas)
- Control de asistencia (2 semanas)
- Generación de estadísticas (3 semanas)
- Interfaz de usuario (3 semanas)
- Documentación y memoria (2 semanas)

Sin embargo, durante la fase de gestión de usuarios se vio necesaria la realización de una nueva fase tras la generación de estadísticas, en la que se desarrollase todo lo relacionado con el borrado de usuarios y las distintas actividades llevadas a cabo por ellos.

4.2. Iteraciones

El software desarrollado va pasando por una serie de distintas etapas intermedias, llamadas iteraciones, en las que se va completando incrementalmente la aplicación, añadiendo nuevas funcionalidades y refinando las existentes hasta cumplir con los objetivos propuestos.

Las iteraciones son las etapas más pequeñas en las que se ha dividido la realización del trabajo. Cada una de ellas forma parte de una de las fases generales, cuyo orden de precedencias se ha mantenido.

4.2.1. Iteración 1: Gestión de usuarios

Una vez instalado y puesto a punto el software necesario para la realización del trabajo, el primer paso es permitir que los usuarios puedan registrarse en la aplicación, acceder a las áreas restringidas para ellos y que se les impida el acceso

cuando traten de entrar sin haber iniciado sesión o a zonas en las que no tienen permisos.

En esta etapa se realiza la conexión a la base de datos que luego servirá para el desarrollo del resto de la aplicación (clase BD), así como el código necesario para la creación de cuentas, el control de sesiones, el inicio (login) y el cierre (logout) de sesión y la validación de los correspondientes formularios.

Las contraseñas se almacenan en la base de datos empleando un *hash*, por motivos de seguridad.

También se crea una página a modo de perfil de usuario, en la que el usuario puede acceder a su información básica y a opciones para modificar sus datos, correo electrónico, contraseña; así como borrar su cuenta. La eliminación de la cuenta aún no tiene en cuenta que habrá que borrar con posterioridad la información resultante de su actividad en la aplicación, por lo que aún tiene una utilidad limitada.

4.2.2. Iteración 2: Creación de clases

Como paso previo para realizar la parte relativa a los cuestionarios, en esta iteración se trabaja en permitir una organización en las clases, dado que los cuestionarios y las distintas actividades se realizarán no por la totalidad de los usuarios de la aplicación, sino por un subgrupo que se corresponderá a los alumnos de una clase real.

En primer lugar se permite a un usuario crear clases nuevas, en las que será considerado profesor de la misma, y que en la página principal del sitio web aparezcan todas aquellas clases que ha creado. También se permite a los usuarios su inscripción como alumnos en las clases.

Nótese que no hay “usuarios alumnos”, ni “usuarios profesores”, sino que un mismo usuario puede ser tanto profesor como alumno, según se encuentre en una clase creada por él o por otro usuario. Así, la consideración que se tiene de un mismo usuario depende del contexto en el que se encuentre y, consecuentemente, son distintas las funciones que se pueden realizar.

Se opta por esta ausencia de división de los usuarios en tipos al considerar que facilita el registro de nuevos usuarios, evitando barreras de entrada.[Ref. 13] El usuario que desea crear una nueva cuenta no debe preocuparse, de antemano, de la actividad que realizará con la aplicación en un futuro (como profesor, o como alumno) ni se le exige un conocimiento previo del funcionamiento de la aplicación.

Al mismo tiempo, facilita que un mismo profesor pueda ser alumno de otras clases (una situación factible) con una única cuenta de usuario, evitando que tenga varias cuentas, para actuar bien como profesor o como alumno.

Una vez que disponemos de usuarios alumnos de la clase y de un usuario profesor de la misma, se programan distintos métodos que permitan saber si un usuario puede acceder a dicha clase, negándose el acceso a aquellos usuarios que no se hayan inscrito.

El profesor, al crear la clase, puede protegerla mediante una contraseña. Esto tiene como objetivo evitar que usuarios que no sean realmente alumnos de una clase —hablamos de forma presencial, en una institución de enseñanza— puedan inscribirse como tales en la aplicación. El profesor, así, puede proteger la inscripción con una clave que facilite personalmente a sus alumnos.

Sin embargo, se ha preferido mantener esta protección como algo opcional para permitir un uso más amplio de la aplicación, más allá de su objetivo principal de apoyo a la docencia presencial. Así, cualquier usuario puede crear clases públicas, abiertas a la inscripción de usuarios sin restricción, algo que podría resultar de utilidad para la realización de los cursos conocidos como MOOC, acrónimo de *Massive Open Online Course*, o cursos en línea, masivos y abiertos.

4.2.3. Iteración 3: Creación de cuestionarios

Se decidió al redactar el anteproyecto de este trabajo que al profesor se le permitiese disponer de un banco de preguntas con cuestiones ya almacenadas, creadas previamente por él, con el objeto de utilizarlas posteriormente en cuestionarios.

Un problema que se plantea a consecuencia de esto es cómo asociar las preguntas previamente creadas con los cuestionarios. Una solución posible es que las preguntas permanezcan independientes a los mismos, de manera que una alteración en una pregunta afecte a todos los cuestionarios donde esta aparezca. Esto permitiría al profesor poder solucionar errores con mayor facilidad, puesto que no sería necesario modificar muchos cuestionarios tras detectar una pregunta errónea, pero también implicaría que no pudiese modificar preguntas para cuestionarios concretos, afectando todos los cambios incluso a cuestionarios tanto futuros como ya realizados.

La otra solución planteada es la de mantener un banco de preguntas, en el que el profesor pueda hacer las modificaciones que estimase oportunas, pero considerar

a estas preguntas como “plantillas”, y no como las preguntas que en sí aparecerán en los cuestionarios. Así, cuando el profesor quisiese añadir una pregunta a un cuestionario, elegiría una de las preguntas disponibles a modo de plantilla, que se copiaría; sería esta copia duplicada la que exclusivamente se convertiría en una pregunta del cuestionario. Esta solución posibilitaría al profesor modificar las preguntas almacenadas a modo de plantilla sin afectar a los cuestionarios futuros o a los ya realizados, pero en caso de querer modificar todas las apariciones de una pregunta, debería buscar todos los cuestionarios donde esta se hubiese asociado y modificarlos, uno a uno.

Finalmente se opta por la primera solución al considerar como uno de los objetivos planteados que la aplicación web realizada fuese fácil de usar. Al ser las preguntas siempre las mismas allá donde se empleen, se le evita al profesor un esquema mental más complejo, consecuencia de tener que darse cuenta de que las preguntas que finalmente aparezcan en los cuestionarios pueden diferir de las preguntas que previamente él creó o que ha modificado en el banco de preguntas.

En cuanto al aspecto técnico, también se tiene en consideración que la primera opción es mucho más sencilla de desarrollar y mantener, así como que requiere un menor almacenamiento en la base de datos al no haber necesidad de duplicar preguntas (por un lado las utilizadas a modo de plantilla y por otro todas aquellas que se hubiesen añadido a los cuestionarios).

En esta iteración, por tanto, se desarrolla la funcionalidad de crear preguntas y cuestionarios en los que dichas preguntas aparezcan. Aún no se pueden programar o realizar en directo, pero sí se crea ya una primitiva forma de responder los cuestionarios mediante un formulario y almacenar las respuestas dadas por los alumnos, que será base para un desarrollo posterior en mayor profundidad, pero que permite ya probar los cuestionarios, ver si funcionaban correctamente, y corregir los errores que se van encontrando sin esperar a una fase más avanzada del desarrollo.

4.2.4. Iteración 4: Programación de cuestionarios

Entre los objetivos del trabajo se encuentran la realización de cuestionarios en directo por parte de los alumnos, pero también que los cuestionarios puedan ser programados para una fecha concreta. Por el mismo motivo de no hacer complejo el uso de la aplicación, se imita la forma en la que se han planteado las preguntas, de manera que el profesor pueda crear cuestionarios de manera independiente y que luego estos puedan ser programados para un futuro o realizados en directo, tantas veces como se desee. De manera coherente, una modificación en los cuestionarios (por ejemplo, cambiar el nombre, o modificar las preguntas que lo conforman)

afectará a sus futuras realizaciones programadas y también a las ya realizadas.

En esta iteración se realiza en primer lugar que los cuestionarios se puedan programar, dejando para más adelante su ejecución en directo. Se permite añadir fechas de apertura y de cierre, entre las cuales podrá el alumno acceder al cuestionario para responder a sus preguntas, y se impide el acceso al cuestionario antes o después de dicha fecha.

También, dentro de la página correspondiente a un cuestionario, se listan todas las futuras programaciones del mismo y se da opción para modificar dichas programaciones o eliminarlas.

Si bien en un primer momento se desarrolla en esta iteración la opción que las programaciones de los cuestionarios se puedan proteger mediante una contraseña, finalmente se decide eliminar esta característica. El objetivo de la protección por contraseña era dar al profesor un modo para que que los cuestionarios se realizasen presencialmente en el aula, ya que él podría asignarles una contraseña y facilitarla a los alumnos de manera presencial durante la clase, impidiendo así que estos accedan antes de que el profesor anuncie la contraseña o que realicen el cuestionario sin estar físicamente presentes en el aula.

Aunque la contraseña cumple con esta tarea, ya forma parte de los objetivos del trabajo el poder realizar cuestionarios en directo que tengan un uso en el propio aula. Dado que los cuestionarios en directo cumplirán esta función, finalmente se opta por separar claramente las funcionalidades, sin mezclarlas entre sí, de forma que un cuestionario se pueda o bien programar, para ser realizado por los alumnos de forma autónoma dentro de un plazo, o bien realizar en directo, de forma que los alumnos tengan que estar presencialmente en el aula y realizar el cuestionario todos en el mismo momento. Se entiende que esta separación de funciones permitirá a los usuarios de la aplicación comprender de manera más clara las distintas opciones que tienen disponibles.

4.2.5. Iteración 5: Cuestionarios en directo

En esta etapa se desarrolla la lógica principal de la realización de cuestionarios en directo, esto es, que se muestren de modo que por un lado aparezcan las preguntas, y que por otro lado los alumnos tengan las distintas opciones con las que responder a cada cuestión.

De modo provisional, el profesor puede hacer avanzar el cuestionario pasando a la siguiente pregunta mediante un botón, y el alumno por su parte refrescar la

página para ver las opciones de la pregunta actual con las que puede responder, tras lo cual se almacenan sus respuestas. Esto permite comprobar el funcionamiento y corregir los errores que se encuentran.

Durante una etapa posterior se hará que tanto el avance de las preguntas como la actualización de las respuestas de los alumnos se hagan automáticamente según el tiempo que el profesor estipule.

4.2.6. Iteración 6: Registros de asistencia

Finalizada la fase relativa a los cuestionarios, queda aún otra de las actividades que se pueden llevar a cabo en las clases: la realización de controles o registros de asistencia.

En un primer momento, en el anteproyecto se indicó que los registros de asistencia podrían ser activados por el profesor de manera puntual o bien programarlos. Era una forma de replicar el funcionamiento de los cuestionarios, dado que se consideraba a los registros de asistencia un tipo particular de cuestionario en el que en vez de responder a una serie de preguntas con distintas opciones, se respondía a una única cuestión (si el alumno se encuentra físicamente presente o no en el aula) con una respuesta afirmativa.

Sin embargo, en esta etapa se considera que no tiene sentido programar un registro de asistencia para fechas concretas, puesto que todos se deben realizar cuando el profesor y los alumnos se encuentran físicamente en el aula a la vez; su funcionamiento será así similar al de un cuestionario en directo, que se activa manualmente por el profesor.

Esta vez sí, a diferencia de los cuestionarios, se hace necesario que el profesor establezca una contraseña que será la que los alumnos deben proporcionar si quieren indicar que se encuentran presentes. El profesor, físicamente en el aula, anunciará la contraseña a los alumnos, dificultando que los usuarios que no estén presentes puedan confirmar su asistencia.

Por lo tanto, se realiza una lógica que permite al profesor iniciar un registro de asistencia e indicar un tiempo máximo en el que los alumnos podrán confirmar su asistencia así como una contraseña, y a los alumnos indicar que se encuentran presentes en el aula proporcionando la misma contraseña establecida por el profesor. El resultado ofrece la funcionalidad que se estableció como objetivo del trabajo, poder gestionar listados de asistencia de alumnos, aunque de manera interna ya no se considera exactamente un tipo particular de cuestionario, sino otra actividad

independiente.

4.2.7. Iteración 7: Estadísticas de asistencia

Dentro de la fase de desarrollo de estadísticas, comenzamos con aquellas correspondientes a la asistencia de alumnos, dado que son más sencillas que las de los cuestionarios, que tendrán que calcular calificaciones. La fase, que se ha denominado de “estadísticas” de forma genérica, incluye todo lo relativo a la generación de datos, la exportación de los mismos, además de algunas medidas de tendencia central.

En esta iteración se crean, fundamentalmente, funciones para calcular medias y medianas de asistencia, que se muestran en una sección de “Estadísticas de clase”, junto con el número de partes de asistencia realizados y el número máximo y mínimo de asistentes. Tanto la media como la mediana de asistencia hacen referencia al número de alumnos asistentes por registro de asistencia realizado.

También se programan métodos que muestran los datos de forma tabulada, que se muestran en la sección de estadísticas de la clase, y que generan los datos en formato CSV,[5] permitiendo su exportación y para su uso posterior en software de hojas de cálculo. En la tabla que se muestra en la sección, se incluye para cada alumno su propio porcentaje de asistencia, así como metadatos que permiten a la interfaz gráfica en HTML generar enlaces para acceder a los datos estadísticos de los alumnos, o bien a cada uno de los registros de asistencia.

4.2.8. Iteración 8: Estadísticas de cuestionarios

En esta iteración se aborda la última parte de la fase de generación de estadísticas, relativa a las cuestionarios realizados y las calificaciones obtenidas por los alumnos en los mismos.

Lo principal es la creación de una la función `mark()`[6] que calcule y devuelva la nota obtenida por un alumno tras la realización de un cuestionario, ya sea programado o en directo.

La función parte de que la nota máxima que se puede obtener es un 100 y divide esta cantidad por igual entre las preguntas asignadas en el cuestionario.

5 Los archivos CSV (del inglés *comma-separated values*) representan datos tabulados, separando mediante comas las columnas y mediante saltos de línea las filas.

6 La función se encuentra en la clase `ScheduledTests`. Para más información, véanse anexos.

Luego, para cada pregunta, se obtiene el número de opciones correctas (ya que pueden ser varias), y divide entre dicho número la cantidad de puntos que entrega la pregunta, obteniendo los puntos de cada opción. De este modo, para conseguir la totalidad de los puntos que otorga la pregunta es que el alumno seleccione como respuesta todas las opciones que sean correctas.

A continuación se miran las opciones respondidas por el alumno a dicha pregunta. Por cada respuesta correcta, se le suman los puntos de opción. Por cada respuesta incorrecta, se le restan los puntos de opción multiplicados por un factor. En cualquier otro caso, ni se le suma ni se le resta puntuación.

De forma similar a como se realizó con la asistencia, se incluye en la página de estadísticas de la clase calificaciones medias y medianas, así como una tabla con los datos que puede ser exportada en formato CSV.

Una cuestión que surge a la hora de mostrar estas medidas de tendencia es si para obtener las notas medias o medianas de un alumno se deben tener en cuenta todos los cuestionarios que se hayan realizado en la clase, o únicamente aquellos en los que el alumno haya participado. En el fondo, lo que se plantea es si hay que distinguir de algún modo entre cuestionarios obligatorios u opcionales, o forzar que todos los cuestionarios sean obligatorios.

Se decide dar al profesor la facultad de decidir, tanto al programar un cuestionario como al lanzarlo, si será obligatoria su realización por parte de los alumnos o no. En el caso de que sea obligatoria, aquellos alumnos que no participen recibirán automáticamente un 0 como calificación.

Esta nueva característica implica mostrar la información estadística de un modo distinto. Así, al mostrar de forma genérica las calificaciones medias y medianas de todos los cuestionarios, por un lado, y las calificaciones medias y medianas de los alumnos, por otro, se tienen en cuenta si cada cuestionario es o no obligatorio.

Además, al acceder a las calificaciones de un cuestionario obligatorio en concreto, se mostrarán la nota obtenida por los alumnos participantes (únicamente los que hayan participado) y la nota genérica de la clase (que incluye a los alumnos que no han participado y que habrán obtenido un 0 en dicha prueba). En el caso de que se quieran conocer las calificaciones de un cuestionario opcional, únicamente aparecerán las notas de los alumnos participantes.

4.2.9. Iteración 9: Borrado

En la primera iteración se programó la opción de que un usuario pudiera borrar su cuenta de usuario. Sin embargo, tras añadir características nuevas esa funcionalidad ha quedado inservible. Hay que decidir qué se hace con todos los datos generados por el usuario, las clases por él creadas, sus calificaciones, etcétera. Llegados a esta fase, en la que ya no se prevén modificaciones en la estructura de las entidades de la base de datos, pues únicamente queda la interfaz gráfica de usuario, es momento de desarrollar los métodos necesarios para efectuar el borrado.

Se considera que todos los elementos creados por los usuarios en la aplicación y almacenados en la base de datos (preguntas, cuestionarios, clases, inscripciones, cuentas de usuario) pueden eliminarse a petición del usuario, pudiendo tener un distinto comportamiento según cada caso.

Si al editar una pregunta se borra algunas de sus opciones por completo (es decir, no se ha sustituido el texto por otra, sino que se ha dejado en blanco), la opción se borra de la base de datos así como las apariciones de dicha opción en las respuestas dadas por los alumnos a la pregunta. Si ya se hubiese respondido a dicha pregunta, se recalcularán posteriormente los resultados teniendo en cuenta que hay una opción menos.

El borrado de una pregunta por parte del profesor en una clase conlleva que dicha cuestión se desasocia de aquellos cuestionarios en los que hubiese incluido. Por tanto, se tiene en cuenta que aquellos cuestionarios programados para un futuro en los que apareciese tendrán una pregunta menos, y que también sucede lo mismo con los que ya han sido realizados, volviendo a calcular en este caso las calificaciones de los alumnos, pues se eliminan las respuestas que hubiesen dado a dicha pregunta.

Si lo que se elimina es un cuestionario al completo, esto ocasiona el borrado tanto de todas sus programaciones futuras como de las ya realizadas, eliminándose así también las calificaciones obtenidas por los alumnos y sus respuestas a las preguntas de dicho cuestionario.

El profesor puede borrar una clase al completo. Esto elimina las preguntas que hubiera almacenadas en ella y también todos los cuestionarios, siguiendo el proceso indicado anteriormente. Además, los alumnos inscritos en la clase dejan de estarlo y se eliminan los datos de asistencia.

Si un usuario decide abandonar una clase en la que estaba inscrito como

alumno, el comportamiento es distinto. Si bien ya no podrá acceder a la misma, y de cara al alumno es como si nunca hubiese estado inscrito, el profesor seguirá teniendo acceso a la actividad que hubiese generado durante su participación en la clase. Podrá seguir viendo las calificaciones que hubiese obtenido así como cada una de sus respuestas, su asistencia, etc.

En el caso de que el mismo usuario vuelva a inscribirse, recuperará la totalidad de su actividad como alumno. Esto se hace así para evitar que un alumno pueda eliminar los datos de sus calificaciones cuando estas son bajas y se inscriba nuevamente para comenzar de nuevo o volver a realizar algún cuestionario.

Cuando un usuario borra su cuenta por completo, se realizan dos acciones distintas. Por un lado se marca como usuario inactivo, pero manteniendo la actividad en las clases en las que estuviese inscrito como alumno. Los profesores, por tanto, podrán seguir teniendo acceso a la actividad generada por los alumnos en sus clases aunque estos se marchen. Por otro lado, se borran por completo las clases en las que fuese profesor, de la misma forma que si él mismo borrase cada una de sus clases como se explicó anteriormente.

Cada uno de los métodos desarrollados que efectúan el borrado, realizan el mismo en una única transacción de la base de datos, asegurando la integridad de los datos. Por ejemplo, al dar la orden de borrar la cuenta de usuario de un profesor, borrará en primer lugar las respuestas dadas por los alumnos en sus clases, y después las clases propiamente dichas. Si fallase el borrado de las clases y este no se llegara a efectuar, no se continuará borrando la cuenta del profesor, pero lo esperable es que entonces tampoco se hayan borrado las respuestas de los alumnos. Esto se consigue mediante el uso de transacciones.

4.2.10. Iteración 10: *Front-end* de cuestionarios en directo

La funcionalidad para realizar cuestionarios en directo ya estaba finalizada, pero en esta iteración se llevará a cabo el front-end: que los cuestionarios avancen de forma sincronizada para el alumno y el profesor, y la interfaz gráfica en HTML para los mismos.

Por un lado, en la vista correspondiente al profesor, aparece una pregunta, la pregunta actual, que va cambiando cada cierto tiempo estipulado por el profesor, haciendo que avance el cuestionario hasta finalizar con todas sus preguntas.

Por otra parte, los alumnos deben ver al mismo tiempo las opciones correspondientes a la pregunta que aparece en la vista del profesor, deben poder

marcar sus respuestas durante el tiempo estipulado, y que tras avanzar a la siguiente pregunta, las opciones que antes salían se sustituyan por las de la nueva pregunta actual.

Para el profesor, esto se realiza con un temporizador que, pasado el intervalo de tiempo correspondiente a la pregunta, hace avanzar el cuestionario, marcando como actual la siguiente pregunta y mostrando esta en su lugar de la anterior; todo ello de manera asíncrona, empleando AJAX.[7]

La solución desarrollada funcionaba correctamente, pero había claramente un problema: al realizarse del lado del cliente, si el profesor abandonaba la página en su navegador de Internet y volvía a entrar, el tiempo volvería a comenzar de nuevo, sin importar lo que hubiese ya transcurrido. En sí, esto no suponía una grave afección por sí mismo, pero dificultaba enormemente la sincronización de las opciones que se muestran a los alumnos (¿cómo saber si el profesor ha salido?, ¿habría que estar constantemente comunicando al sistema si el profesor mantiene la página abierta o no?).

La solución llevada a cabo consiste en que los temporizadores que hacen avanzar las preguntas se inicien teniendo en cuenta el tiempo que ha pasado desde que la pregunta comenzó teóricamente a mostrarse. Es posible calcular este tiempo dado que en la base de datos se almacena el momento exacto en el que el profesor lanzó el cuestionario, cuál es la pregunta actual que se debe mostrar, y el tiempo que se tiene para cada pregunta. Así, al comenzar a mostrar una pregunta, se pone en marcha el temporizador. Si el profesor sale de la página, dejando de mostrar la pregunta, y vuelve a entrar, el cuestionario se pone en marcha ya no con todo el tiempo disponible, sino restándole el que ha pasado desde el momento que la pregunta se comenzó a mostrar.

En el lado del alumno, lo que se hace es obtener el tiempo restante desde que empezó teóricamente a mostrarse la pregunta actual (de igual forma que con el profesor), y utilizarlo para establecer un temporizador. Cuando el temporizador marca el fin del tiempo, el formulario que permite al alumno marcar opciones se envía, almacenando la respuesta dada y redirigiendo al usuario de vuelta a la página con las opciones (que, como el formulario habrá avanzado, serán las correspondientes a la nueva pregunta actual).

Respecto al diseño de la interfaz, realizado en HTML y CSS, se ha optado por simplicidad tanto en la vista del profesor como del alumno, mostrando únicamente la pregunta actual y el número de preguntas ya respondidas y totales, en blanco sobre

7 AJAX, acrónimo de *Asynchronous JavaScript and XML*, es una serie de técnicas de desarrollo web que utilizan distintas tecnologías del lado del cliente para crear aplicaciones web asíncronas.

fondo azul, de manera centrada y dando al texto de la pregunta un tamaño mayor, pensando en una buena legibilidad que facilite a los alumnos la lectura del texto en poco tiempo.[8]

4.2.11. Iteración 11: Resto de la interfaz

En esta etapa se lleva a cabo el desarrollo del resto de la interfaz gráfica de usuario. En primer lugar se decide emplear Bootstrap como base, que incluye una serie de componentes predefinidos, como botones, formularios, etc. Si bien esto facilita la tarea de tener que diseñar componentes particulares, su principal ventaja es que incluye un sistema de rejilla, con el que los distintos elementos de la interfaz se sitúan en filas y columnas, adaptado al diseño web adaptativo. Empleando los elementos y componentes de Bootstrap, la página se adapta a distintos dispositivos, como tabletas, móviles u ordenadores de escritorio.

Salvo la interfaz para los cuestionarios en directo y del registro de asistencia, que tienen características distintas, para el resto de la aplicación se optó por páginas compuestas todas por un menú superior, seguido por un menú de navegación, el contenido principal (formado por una cabecera y una serie de subsecciones), y un pie de página.

8 El color blanco (hex: #ffffff) sobre el fondo azul elegido (hex #0074c1) obtiene un ratio de contraste de 4.92:1, cumpliendo en cuanto al contraste el nivel AAA establecido por el W3C en la guía de accesibilidad WCAG 2.0.

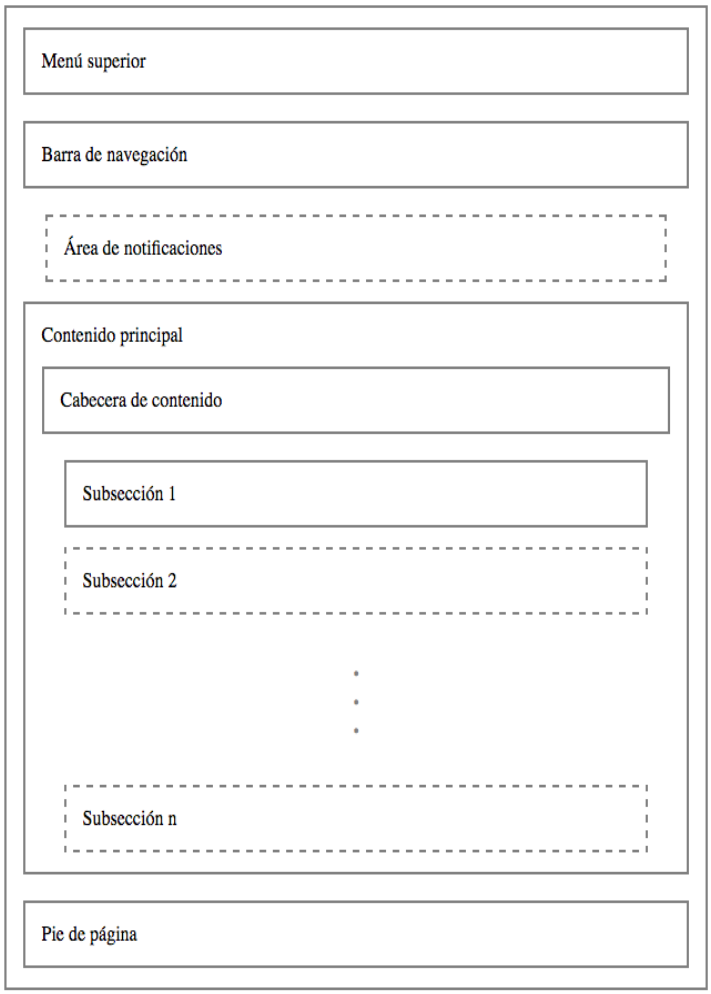


Figura 1: Esquema genérico de la interfaz gráfica.

En la zona de contenido aparece el contenido propio de cada sección de la aplicación. Este contenido se agrupa en una serie de subsecciones, denominadas de forma interna como “tarjetas”, claramente separadas entre sí unas de otras, de manera que cada una de ellas formen una unidad en la que todos los elementos que se incluyen dentro están relacionados entre sí. Una sección en la que esto se aprecia muy fácilmente la de estadísticas, donde hay una tarjeta que muestra los datos propios a la asistencia y otra con los relativos a las calificaciones, haciendo más fácil discriminar visualmente entre unos y otros.

Situada entre la barra de navegación y el contenido principal, se sitúa la zona de notificaciones, donde el usuario recibe distintos mensajes. En general, cuando un usuario realiza una acción en la aplicación, en esta zona se le muestran mensajes con el resultado de la misma. Hay distintos grados de mensajes: de éxito (para comunicación de resultado exitoso), de advertencia (para advertir al usuario de resultados parciales o de una acción necesaria por su parte) y de peligro

(generalmente, para avisar de errores o situaciones especialmente delicadas).

La mayoría de los mensajes se muestran después de que el usuario realiza la acción y desaparecen en cuanto el usuario recarga la página o navega por otra página de la aplicación web. Estos mensajes son los llamados mensajes “flash”, que internamente se generan tras realizar la acción y se añaden a la propia sesión del usuario, mostrándose al mismo tras ser redirigido a una página del sitio web y siendo posteriormente eliminados.

En la zona de notificaciones, además, se incluyen mensajes que avisan de actividades que se están llevando a cabo en ese momento y que requieren atención inmediata por parte del alumno, como cuestionarios en directo o partes de asistencia. A diferencia de los mensajes “flash”, que se muestran únicamente tras efectuar una acción, este tipo de mensajes aparecen en cualquier sección de la aplicación web y en el momento en que las actividades inician, pues se cargan de manera asíncrona.

Por último, hay un mensaje que se muestra únicamente en la sección de clases, principal sección de la aplicación y a la que se accede tras el inicio de sesión, que advierte al usuario en el caso de que tenga actividades sin realizar en alguna clase. Dicho mensaje se complementa con pequeños avisos de que una clase tiene tareas pendientes, junto al nombre de la propia clase.

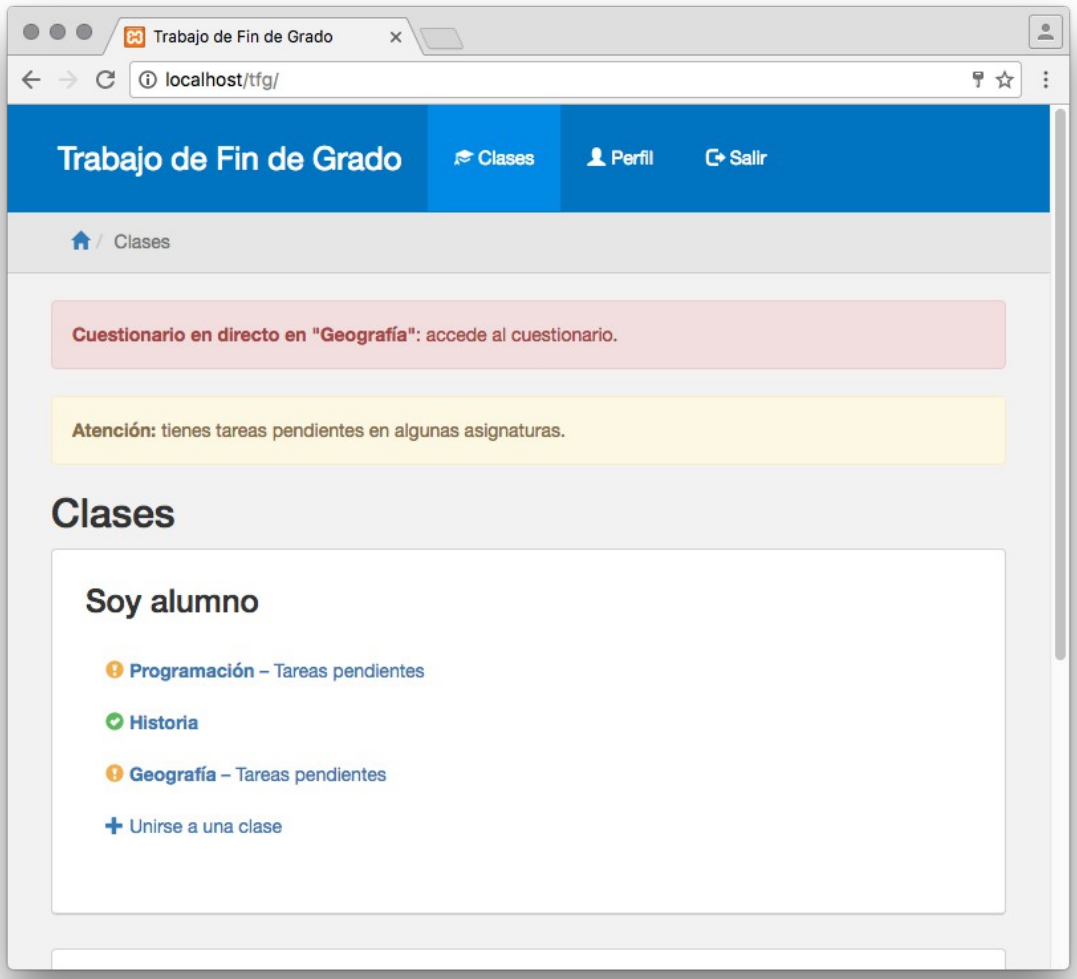


Figura 2: En el área de notificaciones aparecen mensajes para el usuario con distinto grado de importancia.

4.2. Temporización final

En la siguiente tabla se encuentra la duración final de cada iteración y de cada una de las fases que estas componen.

Fase	Iteración	Fecha inicio	Fecha fin	Días
Usuarios	1. Gestión de usuarios	09/06/2016	14/06/2016	6
Cuestionarios	2. Clases	14/06/2016	17/06/2016	32
	3. Creación de cuestionarios	17/06/2016	04/07/2016	
	4. Programación de cuestionarios	05/07/2916	12/07/2016	
	5. Cuestionarios en directo	12/07/2016	15/07/2016	
Asistencia	6. Registro de asistencia	15/07/2016	18/07/2016	4
Estadística	7. Estadísticas de asistencia	19/07/2016	27/07/2016	25
	8. Estadísticas de cuestionarios	28/07/2016	12/18/2016	
Borrado	9. Borrado	13/08/2016	15/08/2016	3
Interfaz	10. *Front-end* de cuestionarios en directo	16/08/2016	24/08/2016	15
	11. Resto de interfaz	24/08/2016	30/08/2016	
Memoria		30/08/2016	17/09/2016	19

Nótese que hay días que han correspondido a dos iteraciones distintas (fin de una, comienzo de la siguiente), por lo que la duración real es inferior a la suma de los días correspondientes a cada iteración o a cada fase.

El trabajo se inició el 9 de junio de 2016 y se dio por finalizado el 17 de septiembre del mismo año, siendo su duración final de 14,4 semanas (101 días). Con respecto a la temporización total inicial, el trabajo se ha realizado en casi 2,4 semanas más de lo inicialmente previsto.

Atendiendo a cada una de las fases encontramos que la fase relativa a los cuestionarios tuvo una duración significativamente mayor a lo planteado (32 días respecto a 14 previstos, algo más del doble). También se excedió la duración de la fase de estadísticas, aunque en menor medida (25 días respecto a los 14 iniciales).

Este retraso se ha visto compensado en parte por la menor duración de las fase relativa al control de asistencia, de 4 días respecto a los 14 previstos, y de la fase de desarrollo de la interfaz, de 15 días respecto a 21.

4.3. Diagrama de Gantt

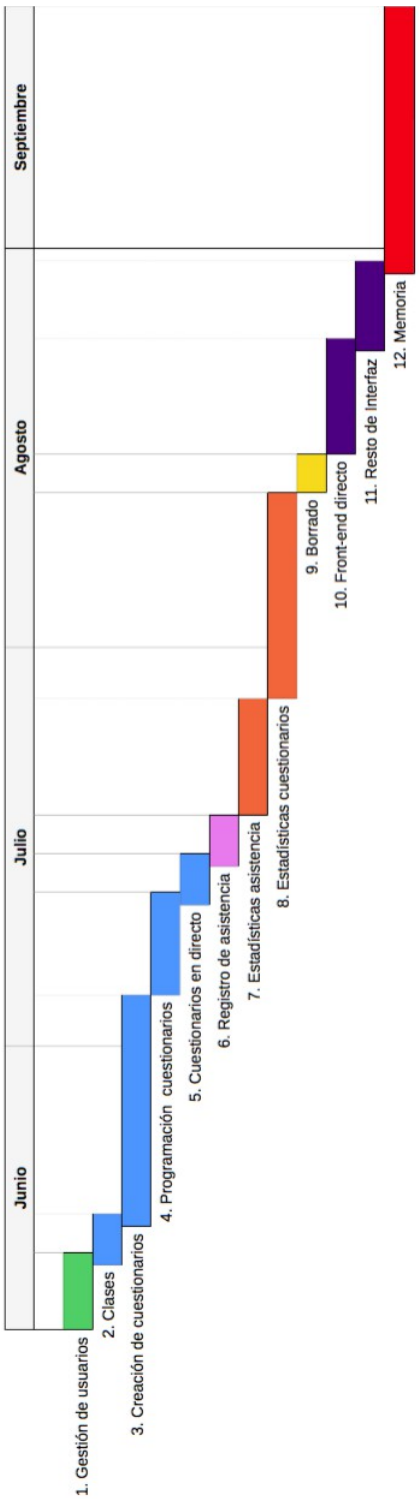


Figura 3: Diagrama de Gantt. Representa la duración de cada iteración (agrupadas en fases según colores)

5. Descripción de la aplicación

Se describe a continuación la funcionalidad de la aplicación resultante tras las distintas iteraciones por las que ha pasado durante todo el proceso de realización de este trabajo.

5.1. Gestión de usuarios

El usuario de la aplicación, al acceder a la misma, dispone de un formulario para iniciar sesión con su cuenta de usuario así como otro formulario para crear una nueva cuenta en el caso de que aún lo haya hecho con anterioridad.

Puede crear una cuenta aportando su dirección de correo electrónico, su nombre y apellidos, y una contraseña que le permitirá el acceso de manera exclusiva. Debe repetir la contraseña aportada como medida para asegurar que ha memorizado la contraseña y no se ha equivocado al introducirla (dado que no será visible, por motivos de seguridad).

También puede aportar un Documento Nacional de Identidad (DNI) o similar. Se incluye este dato, de forma opcional, para facilitar al profesor la posterior labor de identificación de los alumnos inscritos en sus clases.

La aplicación comprueba, una vez enviado el formulario, la validez de sus datos: que haya introducido aquellos campos que son obligatorios (nombre, apellidos, dirección de correo electrónico y contraseña), que la contraseña tenga una longitud de al menos seis caracteres[9] y que haya sido confirmada correctamente, y que no haya introducido una dirección de correo electrónico o un DNI ya empleados. Si no hay errores, se registra al nuevo usuario e inicia sesión de manera automática.

En el caso de que el usuario ya estuviera registrado anteriormente, para acceder tendrá que proporcionar la dirección de correo electrónico y la contraseña que proporcionó durante su registro. La aplicación comprobará que la dirección existe en la base de datos y, en ese caso, que la contraseña indicada coincide con la almacenada en el sistema. Si no hay errores, se inicia sesión.

Una vez el usuario ha iniciado sesión, se le muestra la pantalla principal de la aplicación. Desde ella puede acceder a las clases en las que se encuentra inscrito

9 La constante `PASSWORDLENGTH` del archivo de configuración *config.php* permite variar el tamaño mínimo de las contraseñas.

como alumno, ver si tiene tareas pendientes por realizar en alguna de ellas y también acceder a las clases en las que él es profesor. Existen también enlaces para unirse a una clase como alumno o crear una nueva y ser profesor. En la zona superior se encuentran enlaces para acceder a su perfil de usuario y para cerrar sesión.

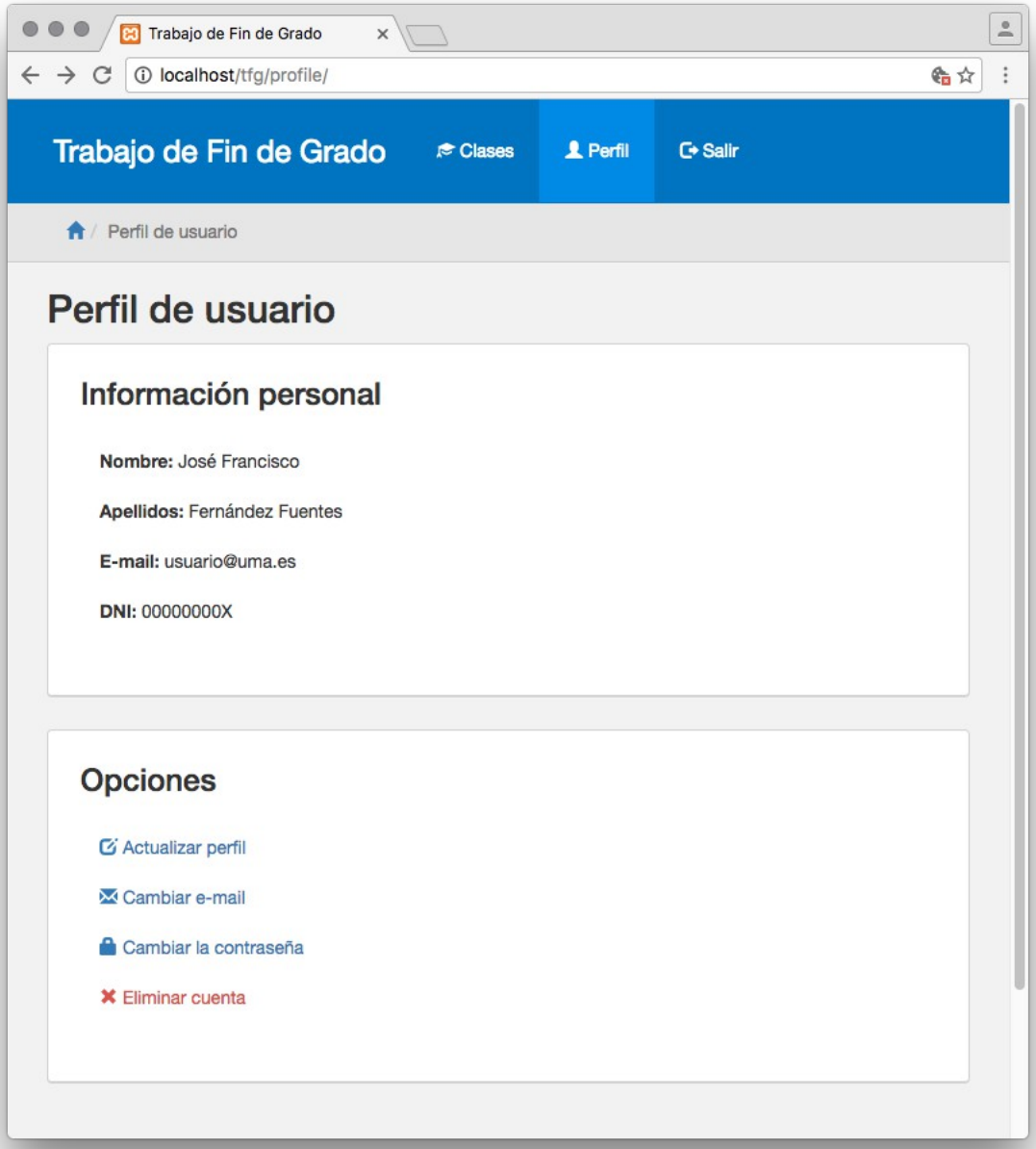


Figura 4: Perfil de usuario

En su perfil, el usuario encuentra los datos personales que introdujo para su registro y dispone de opciones para actualizarlos y para cambiar la dirección de

correo electrónico o la contraseña con los que inicia sesión. Al cambiar la dirección de correo electrónico, la aplicación comprueba que la nueva dirección no exista ya en la base de datos empleada por otra cuenta. En el caso de la contraseña, se comprueba que sea mayor al número mínimo de caracteres y se le pide al usuario que la repita de la misma forma que al registrarse.

Desde el perfil el usuario tiene acceso a la opción de borrado de la cuenta. Para eliminar su cuenta, debe proporcionar su contraseña y confirmar que está seguro de que desea eliminarla. Si la contraseña es errónea o no se confirma la voluntad, no se efectúa el borrado.

La aplicación se asegura de que ningún usuario pueda acceder a la misma sin haber iniciado sesión previamente, y de que no pueda acceder a secciones para las que no tiene permiso (por ejemplo, un alumno no podrá modificar el nombre de una clase, pero sí el profesor que creó la misma).

5.2. Gestión de clases

Un usuario puede inscribirse en una clase para participar en las actividades que en ella se realicen, y se le considerará alumno respecto a dicha clase. El mismo usuario también podrá crear sus propias clases y administrarlas, siendo considerado profesor de las mismas.

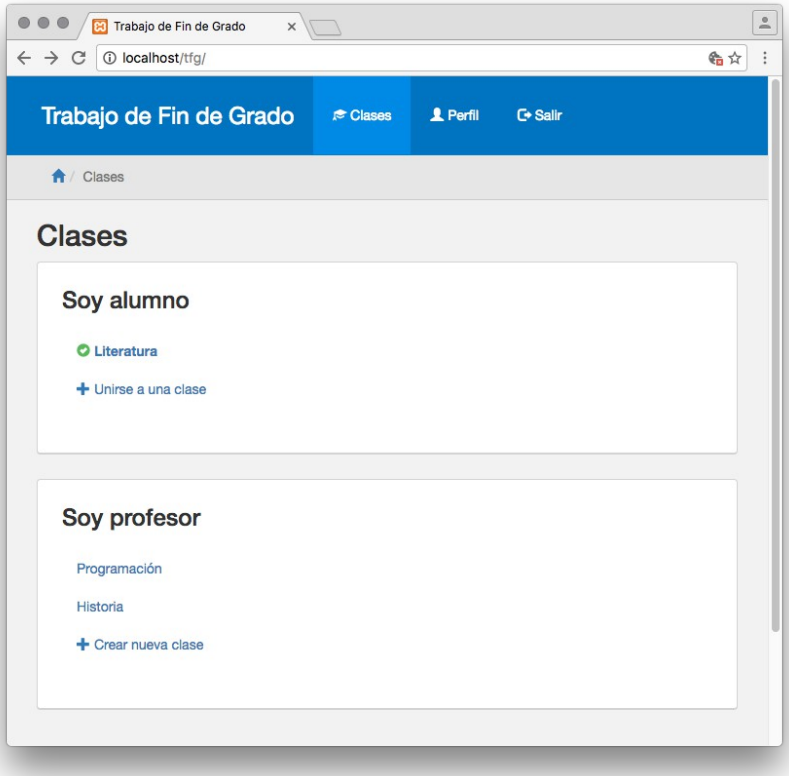


Figura 5: Vista principal de la aplicación web. Un mismo usuario puede acceder a las clases en las que está inscrito como alumno y en las que es profesor.

5.2.1. Creación de clases

El usuario, una vez iniciada su sesión, puede acceder desde la pantalla inicial a un formulario para crear una clase nueva. Se le pide de manera obligatoria únicamente que introduzca un nombre para identificar la clase, y podrá además de manera opcional incluir una contraseña que deberán proporcionar los alumnos en el momento de su inscripción. Debe también confirmar la contraseña una vez más.

Enviado el formulario, se comprueba que se haya introducido un nombre para la clase y que, en el caso de que se desee añadir una contraseña, tenga una longitud adecuada y que haya sido confirmada correctamente. Si todo es correcto, se crea la clase y automáticamente se redirige a ella al usuario.

El profesor, el usuario creador de una clase, puede también eliminar la clase y con ella toda la actividad de la misma.

5.2.2. Inscripción en clases

El usuario, una vez iniciada su sesión, puede acceder desde la pantalla inicial a un formulario para unirse a una clase ya existente. Se le pide que introduzca un identificador de la clase (ID) y una contraseña en el caso de que la clase esté protegida. Ambos datos deben ser proporcionados por el profesor de la misma.

La aplicación comprueba, además de que la contraseña sea correcta (en el caso de que la clase a la que se desea inscribirse esté protegida), que el usuario no sea ya alumno de la clase o que sea profesor de la misma. En ambos casos, se redirigirá automáticamente al usuario a la clase y se le indicará con un mensaje que ya tenía acceso a la misma.

A medida que el usuario se inscribe en clases, aparecen los nombres de las mismas en la pantalla inicial a la que se accede una vez iniciada la sesión. Cada uno de esos nombres es un enlace que lleva a la página específica de la clase correspondiente.

5.2.3. Eliminación de inscripciones

En la página correspondiente a cada clase, el usuario encuentra información disponible sobre la misma y sobre las actividades que ha realizado en ella. También puede acceder a la opción para eliminar su inscripción en la clase.

Para eliminar dicha inscripción, debe proporcionar mediante un formulario su contraseña, con el objetivo de comprobar que es el propio usuario el que desea realizar dicha acción y no otra persona. Además, debe confirmar marcando una casilla que desea eliminar su inscripción y que es consciente que dicha desmatriculación no borrará la información que haya generado con su participación en la clase.

Al eliminar una inscripción, esta se marca como inactiva de manera interna en la base de datos, sin suprimir realmente su información. Así, un usuario puede borrarse de una clase, pero las calificaciones que haya obtenido y los registros de asistencia en los que haya participado seguirán siendo visibles para el profesor (quien sabrá, además, de la retirada del alumno).

Si un usuario decide volver a inscribirse como alumno en una clase de la que previamente ya había estado matriculado y de la que había eliminado su inscripción, no comienza de cero sino que recupera la actividad que había generado

anteriormente. El sistema marca en ese caso la inscripción como activa nuevamente.

5.3. Cuestionarios

La aplicación busca apoyar a la docencia en el propio aula física mediante la realización de cuestionarios de forma presencial por los alumnos, pero simplificando el proceso de realización los mismos, tanto por parte del profesor como de los alumnos, y ofreciendo los resultados al profesor automáticamente.

Así, los cuestionarios se pueden realizar en directo. La aplicación muestra al profesor un marcador en el que aparecen las preguntas, que proyectará a la audiencia presente en el aula, mientras que para los alumnos la aplicación se convierte en una suerte de control remoto con el que pueden responder a cada una de las preguntas.

A medida que van avanzando las preguntas del cuestionario, las opciones disponibles para los alumnos se actualizan para corresponder con cada una de las cuestiones; finalmente, se obtiene la calificación obtenida.

Los mismos cuestionarios también pueden ser programados para la realización en una fecha concreta, en lugar de ser realizados en directo y de forma presencial en el aula. En ese caso, se volverán disponibles para su realización en una fecha de apertura y se cerrarán en otra fecha posterior. Los alumnos pueden acceder a ellos de manera autónoma, obteniendo las preguntas y opciones de las mismas, y realizarlos en su plazo para obtener calificación.

5.3.1. Creación de preguntas

Los cuestionarios están compuestos por una serie de preguntas, que son creadas por el profesor de manera independiente a los cuestionarios. Es decir, antes aún de crear cuestionarios, el profesor puede mantener un banco de preguntas en la clase, siendo posible crear nuevas preguntas con sus correspondientes opciones, modificarlas, o eliminarlas. Serán estas preguntas las que, posteriormente, se asignen a los cuestionarios.

Las preguntas están, por tanto, asociadas a una clase en primer lugar, antes de que formen parte de uno o varios cuestionarios. Esto facilita al profesor la creación de cuestionarios, ya que no es necesario que cada vez que los cree tenga a su vez que crear preguntas, añadirles a estas las opciones y elegir cuáles son las correctas. En lugar de eso, al crear un cuestionario el profesor simplemente tiene que

seleccionar las preguntas de entre un listado de cuestiones ya existentes en la clase. También permite que una misma pregunta pueda ser parte de varios cuestionarios, por lo que las modificaciones en la misma se reflejarán en todos aquellos cuestionarios en los que se encuentre. Solucionar un posible error, en el que por ejemplo se haya dado por correcta en varios cuestionarios una opción que en realidad era incorrecta, supone simplemente modificar la pregunta, y no cada uno de los cuestionarios en los que aparece.

Para añadir una pregunta, el usuario profesor dispone de un enlace dentro de la propia clase que le llevará a un formulario de creación de cuestiones. Debe indicar un título para la pregunta así como una serie de opciones con las que los alumnos podrán responder a la misma (cuatro opciones como máximo)[10]. Para cada opción, el profesor puede indicar si es correcta o no. Tras enviar el formulario, antes de almacenar la pregunta en la base de datos la aplicación comprueba que se haya escrito un título y que haya al menos dos opciones, una de las cuales deberá ser correcta. No se han considerado posibles la creación de preguntas sin opciones, con solo una opción, o sin opciones correctas. En cambio, sí es posible crear preguntas cuyas opciones sean todas correctas.

5.3.2. Edición de preguntas

El profesor dispone en la página de la clase de una lista de las preguntas creadas con enlaces a las mismas. Entrando en ellas puede editarlas mediante un formulario que, de la misma forma que cuando se crearon, comprobará que tras las modificaciones efectuadas siga habiéndose escrito un título para la pregunta, que haya al menos dos opciones y que como mínimo una sea correcta.

Las modificaciones que se realicen sobre una pregunta afectarán a los cuestionarios en los que estas se hayan asignado, incluso aunque ya hayan sido respondidos por los alumnos. Debe tenerse en cuenta que esto afecta a las calificaciones de los cuestionarios, que son reevaluados tras los cambios.

Por ejemplo, si una pregunta contaba con dos opciones correctas, cada una de ellas tenía un valor del 50% del total de la puntuación de dicha pregunta. Un alumno que hubiera respondido ambas opciones habría conseguido una calificación del 100% en dicha pregunta. Si tras editar la pregunta sólo una de esas dos opciones es correcta, el alumno que marcó las dos recibirá un 100% por la respuesta correcta pero se le restará la mitad por haber marcado una respuesta que ahora es errónea, siendo su calificación en dicha pregunta del 50%. Para una explicación más detallada

10 La constante `NUMOPTIONS` del archivo de configuración *config.php* permite cambiar el número máximo de opciones que admite una pregunta.

de cómo se puntúan los cuestionarios, véase la sección 4.2.8 sobre estadísticas de calificaciones.

Es posible también eliminar definitivamente una pregunta de la base de datos.

5.3.3. Creación de cuestionarios

Desde la página principal de la clase el profesor dispone de un enlace que le llevará a un formulario de creación de un cuestionario, que aparece una vez la clase cuenta con alguna pregunta. Para añadir un cuestionario nuevo únicamente es necesario escribir un título para el cuestionario, y seleccionar cuantas preguntas se deseen de entre una lista de preguntas de la clase.

Una vez el nuevo cuestionario se ha guardado en el sistema, el usuario es redirigido a la página de la clase, donde puede ver que a la lista de cuestionarios se ha añadido el recién creado. Desde dicha lista, además, puede acceder a una página para cada cuestionario, donde dispone de opciones para modificar el título o las preguntas que forman parte del mismo (seleccionando o desmarcando aquellas preguntas de la lista que según las desee añadir o retirar), programarlo, lanzarlo en directo, o eliminarlo.

A continuación se verá en más detalle los cuestionarios programados y en directo.

5.3.4. Programación de cuestionarios

Los cuestionarios que el profesor crea en una clase son realmente modelos de cuestionarios que luego pueden ser programados tantas veces como se desee. Las respuestas de los alumnos al cuestionario se asocian a cada una de las veces que el cuestionario se haya programado y no con el cuestionario en sí.

El profesor puede encontrar dentro de la página del cuestionario un enlace desde el que accede al formulario para programar el cuestionario. Debe indicar una fecha de inicio, posterior a la fecha actual, y una fecha de cierre. El cuestionario se abrirá durante dicho periodo y los alumnos podrán acceder a él y responderlo.

Al programarlo, el profesor puede indicar si es obligatorio para los alumnos realizar o no el cuestionario en dicho periodo. Una u otra opción tiene implicaciones a la hora de calcular las calificaciones y al mostrar las estadísticas de la clase. Así, si la realización de un cuestionario es opcional y un alumno no la realiza, no obtiene

calificación alguna; en el caso de que sea obligatoria, su puntuación es cero.

En la página del cuestionario correspondiente se le indica al profesor si ha sido o no programado, y en qué fechas en caso de que así sea. Dispone también de opción para modificar la programación (fecha de inicio, cierre y obligatoriedad) así como para eliminarla. En el caso de que tras programar el cuestionario éste hubiera iniciado, ya no será posible modificar su fecha de inicio; tampoco se podrá eliminar, pero sí forzar su finalización, obteniendo al final sus calificaciones.

El profesor también puede seleccionar la forma en la que los alumnos recibirán la corrección. Es posible configurarla de modo que puedan acceder únicamente a su nota o a una corrección completa que les muestre para cada una de las preguntas en qué se equivocaron o respondieron correctamente; además, que lo puedan hacer una vez han terminado de responder individualmente o que tengan que esperar a que finalice el periodo de realización que se indicó para el cuestionario.

5.3.5. Lanzamiento de cuestionarios en directo

Llamamos “lanzamiento” a iniciar un cuestionario para ser realizado en directo, en el propio aula y con la presencia física de los alumnos. Los lanzamientos son un caso especial de programación de un cuestionario; en ellos su fecha de inicio corresponde el momento del lanzamiento y no a una fecha futura.

Cualquier cuestionario se puede lanzar en todo momento, independientemente de las veces que haya sido programado. Es posible incluso lanzar en directo un cuestionario que en ese mismo momento está abierto para su realización tras haberse programado.

Dado que los cuestionarios en directo buscan ser algo dinámico, para su uso en la clase, se ha intentado facilitar el lanzamiento. Se puede realizar desde con un simple enlace tanto desde la lista de cuestionarios de la clase, como desde la página de un cuestionario concreto. Se le pide al profesor indicar en un formulario un número de segundos (que será el tiempo que tendrán los alumnos para responderlo), establecer la realización de dicho cuestionario como obligatorio u opcional, y seleccionar la forma en la que los alumnos recibirán la corrección. Tras enviar el formulario, el cuestionario es lanzado y comienza a realizarse en directo.

Se ha programado una opción que permite lanzar en directo un cuestionario generado automáticamente de forma aleatoria, disponible desde la página de la clase. En este caso, el formulario de lanzamiento incorporará, además de los campos ya mencionados, uno para indicar el número de preguntas que tendrá el cuestionario.

Automáticamente, se creará un cuestionario, se le añadirán dicho número de preguntas de manera aleatoria, y se lanzará.

También, en el momento de rellenar los parámetros para lanzar un cuestionario, el profesor dispone de un botón que le permite lanzarlo en modo de encuesta. Esto hará que se almacenen las distintas opciones marcadas por los alumnos, pero no se hará una corrección a cada uno de ellos ni se tendrá en cuenta para calcular las calificaciones. Tras finalizar, el profesor obtiene el número de alumnos que han marcado cada opción.

5.4. Realización de cuestionarios

La manera de realizar los cuestionarios por parte de los alumnos varía dependiendo de si los cuestionarios han sido programados o lanzados en directo. También es distinta la implicación del profesor: mientras que un cuestionario programado es realizado por los alumnos sin intervención del profesor, sí será necesario que él esté físicamente en el aula para la realización de un cuestionario en directo, junto a sus alumnos.

5.4.1. Realización de cuestionarios programados

Cuando llegue la fecha de inicio en la que un cuestionario haya sido programado, este se abrirá automáticamente y permitirá que los alumnos lo realicen, mostrando todas las preguntas asignadas con sus respectivas respuestas que pueden ser marcadas por el alumno, quien tras su envío obtendrá una calificación.

El cuestionario podrá ser realizado en cualquier momento desde su fecha de apertura hasta su fecha de cierre, salvo que el profesor fuerce la finalización antes de tiempo. Una vez haya dado comienzo el profesor también podrá editar la programación, actualizando tanto la fecha de inicio como de cierre, o su obligatoriedad.

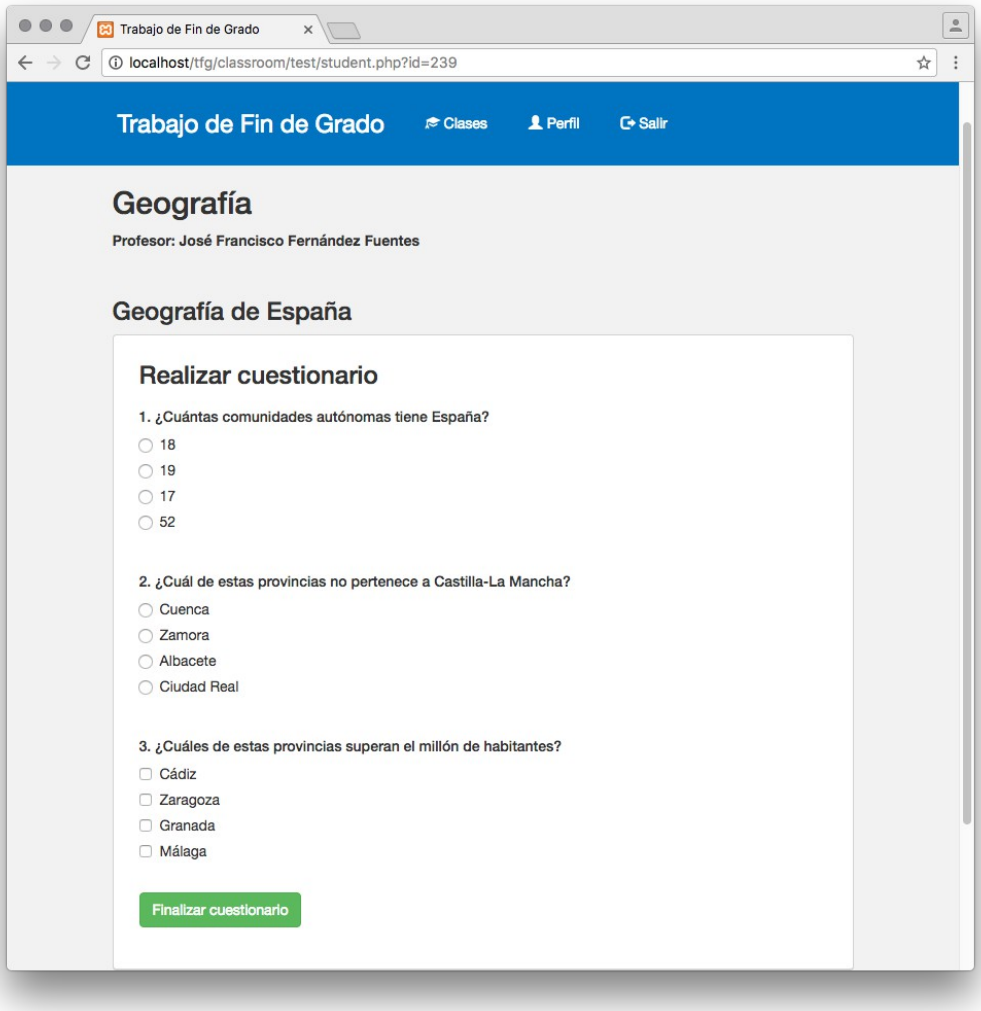


Figura 6: Realización de un cuestionario programado.

5.4.2. Realización de cuestionarios en directo

Tras lanzar el profesor un cuestionario, en el navegador que esté empleando se abrirá una nueva pestaña con una pantalla en la que aparecen las preguntas, tras un tiempo de espera prudencial para que los alumnos entren al cuestionario.

De una en una, las preguntas van avanzando automáticamente pasado el tiempo que se haya fijado para su realización. A los alumnos, por su parte, únicamente se les muestra las respuestas a cada una de las preguntas que aparecen en la pantalla del profesor, de manera sincronizada con el profesor.

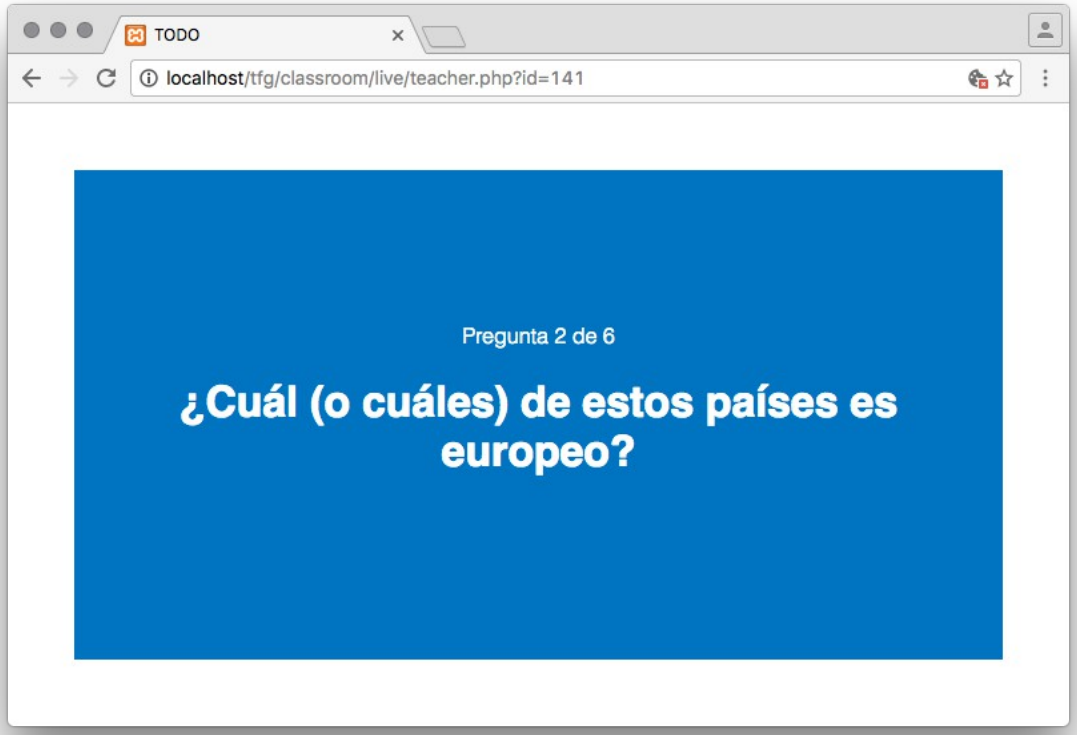


Figura 7: Tras lanzar un cuestionario en directo, el profesor comienza a mostrar las preguntas.

Se espera que el profesor proyecte su pantalla o la muestre públicamente a sus alumnos presentes en el aula. Así, la aplicación convierte el dispositivo del profesor en una especie de marcador, mientras que los de los alumnos funcionan como pulsadores desde los cuales pueden responder. De esta forma, los cuestionarios lanzados en directo únicamente pueden ser realizados en el aula de forma presencial. Si un alumno accediese a un cuestionario en directo, aunque podría contestar a las preguntas, no sabría a qué está respondiendo.

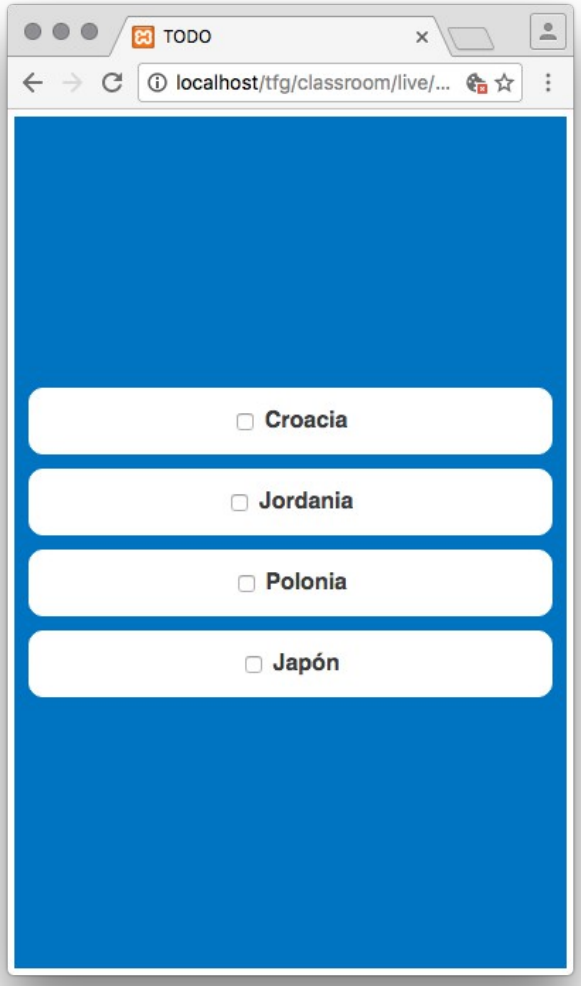


Figura 8: El alumno, por su parte, obtiene un pulsador para responder a cada pregunta.

5.5. Control de asistencia

La aplicación permite ayudar al docente en el control de la asistencia presencial de una clase con la opción de pasar partes o registros de asistencia.

Para crear un parte de asistencia, el profesor debe indicar una contraseña que deberá confirmar, y seleccionar el tiempo durante el que los alumnos pueden indicar que se encuentran presentes. La contraseña, que el profesor deberá anunciar presencialmente en el aula a sus alumnos, tiene como objetivo que únicamente los alumnos físicamente presentes puedan confirmar su asistencia, pues estos deberán

proporcionarla en la aplicación.

Los resultados de cada control de asistencia se almacenan y pueden ser accedidos junto al resto de estadísticas de la clase.

5.6. Estadísticas

Los distintos resultados generados como consecuencia de la realización de actividades en una clase se van almacenando en la base de datos. Estos datos se le ofrecen al profesor de una clase a través de la sección de estadísticas de la misma.

5.6.1. Resultados de asistencia

En dicha sección se pueden obtener medidas relativas a la asistencia, como la media, la mediana, la asistencia máxima y mínima y el número de partes de asistencia realizados en la clase. Junto a dichas medidas se intenta dar información de contexto que permita comprender mejor el dato. Por ejemplo, además de la asistencia media en alumnos/día, se indica a qué porcentaje del total de alumnos corresponde dicha media. En el caso de las asistencias máxima y mínima, se indica la fecha en la que sucedieron estos eventos.

Dichos datos se muestran al profesor a modo de resumen, pero es posible ver los datos de asistencia tabulados desde la misma página, indicando qué alumnos asistieron a cada registro de asistencia, así como el porcentaje de asistencia de cada alumno de la clase.

Desde la tabla se enlazan además cada uno de los resúmenes estadísticos correspondientes a los alumnos (que agrupan sus resultados de asistencia y calificaciones) y a cada registro de asistencia (que listan los nombres de los alumnos asistentes).

Los datos de asistencia almacenados también se pueden descargar en formato CSV que permite importarlos en software de hojas de cálculo como LibreOffice Calc o Microsoft Excel. En dichos archivos no se incluyen medidas calculadas, sino los datos en crudo, con el objeto de que el profesor pueda trabajar con las herramientas de cálculo que prefiera.

5.6.2. Calificaciones

De una forma similar, el profesor puede obtener información sobre las calificaciones obtenidas en los cuestionarios. Se le ofrecen medias y medianas diferenciando entre las de los cuestionarios y las obtenidas por los alumnos.

También se tiene en cuenta que un cuestionario puede haber sido programado con realización obligatoria u opcional para los alumnos. En los casos en los que sean opcionales, no se le tiene en cuenta para calcular las medias y medianas que el alumno no haya realizado un cuestionario; si son obligatorios, las “no participaciones” se califican con un cero y son tenidas en cuenta de la misma forma que un cuestionario en el que sí haya participado.

Nuevamente, se muestra también una tabla con los datos de calificación de los alumnos en cada cuestionario, que pueden ser descargados en formato CSV por el profesor.

Desde dicha tabla se pude acceder, además, a los resúmenes estadísticos correspondientes a cada alumno (que agrupan sus calificaciones y resultados de asistencia) y a cada cuestionario realizado (que muestran un listado con las notas obtenidas por los alumnos, así como las medias y medianas).

6. Conclusiones y líneas de mejora

En los capítulos anteriores se ha podido apreciar la evolución del trabajo, así como las funcionalidades finales de la aplicación tras pasar por una serie de etapas de desarrollo, que deben responder al objetivo de facilitar la labor docente empleando cuestionarios on-line. Como consecuencia del trabajo, se exponen a continuación una serie de conclusiones:

- La metodología empleada ha permitido cumplir los objetivos del proyecto pese a no partir de unos requisitos especificados de forma detallada. Ha hecho posible el avance del proyecto sin tener que hacer demasiadas rectificaciones a lo ya realizado en etapas previas. Esto nos hace plantear la hipótesis de que la mayor duración respecto a lo inicialmente planteado se debe a unas previsiones demasiado optimistas en cuanto al tiempo en algunas fases.

- Bootstrap ha facilitado considerablemente la realización de la interfaz gráfica y su diseño adaptativo, sin apenas necesidad de modificaciones en cuanto a su adaptación a dispositivos móviles.

- El uso de un software de control de versiones como Git ha sido útil no únicamente durante la etapa de desarrollo, para llevar un control de los cambios realizados en el código, sino también para la realización de esta memoria al poder comprobar a posteriori la duración real de cada iteración.

Por otra parte, se proponen a continuación una serie de posibles mejoras técnicas y líneas en las que seguir avanzando:

- Profundizar aún más en temas de usabilidad. A lo largo de la realización del trabajo se han expuesto distintas decisiones tomadas en base a lograr una mayor facilidad de uso por parte de futuros usuarios. En una posible fase posterior, podría llevarse a cabo un estudio de la usabilidad del sitio en un entorno real, siendo ya empleado por distintos usuarios, para corroborar la idoneidad de las decisiones adoptadas.

- Emplear formatos de intercambio de datos como JSON podría aislar más la vista de la lógica del sistema. Se considera especialmente útil en aquellas características en las que el tiempo de carga debe reducirse todo lo posible, como son los cuestionarios en directo. En la carga asíncrona de las preguntas, JSON podría evitar transmitir código HTML, como se hace actualmente.

- Para mostrar las notificaciones al usuario, actualmente se emplean llamadas asíncronas al servidor cada cierto tiempo. Esto genera una cantidad de tráfico que podría reducirse empleando distintas tecnologías *push* o WebSockets.

Naturalmente, la lista de propuestas no pretende ser exhaustiva. En cuanto a nuevas características, la aplicación podría ampliarse tanto como se desease, aunque quizá deban enfocarse los esfuerzos a mejorar los cuestionarios en directo con nuevas funciones. El resto de características (cuestionarios programados, control de asistencia, etc.) ya están cubiertas por aplicaciones y sistemas de terceros con amplia aceptación y con años de desarrollo a sus espaldas, por lo que en este trabajo no han sido una cuestión prioritaria, ni se espera que lo sea en posibles ampliaciones futuras del mismo.

7. Referencias bibliográficas

1. APACHE SOFTWARE FOUNDATION, The. *Apache HTTP Server*. [En línea] Disponible en: *https://httpd.apache.org/*

2. NETCRAFT. *July 2016 Web Server Survey*. [En línea] Disponible en: *https://news.netcraft.com/archives/2016/07/19/july-2016-web-server-survey.html*

3. PHP GROUP. *Manual de PHP. Prefacio*. [En línea] Disponible en: *https://secure.php.net/manual/es/preface.php*

4. LERDORF, R; TATROE, K; MACINTYRE, P. *Programming PHP*. 2ª edición. O'Reilly, 2006. ISBN: 9780596006815

5. TIOBE. *TIOBE Index*. [En línea] Consultado en septiembre de 2016. Disponible en: *http://www.tiobe.com/tiobe-index/*

6. W3TECHS. *Usage of server-side programming languages for websites*. [En línea] Consultado en septiembre de 2016. Disponible en: *https://w3techs.com/technologies/overview/programming_language/all*

7. PHP GROUP. *Supported Versions*. [En línea] Disponible en: *http://php.net/supported-versions.php*

8. APACHE FRIENDS. *XAMPP Installers*. [En línea] Disponible en: *https://www.apachefriends.org/es/index.html*

9. PHPMYADMIN CONTRIBUTORS. *About*. [En línea] Disponible en: *https://www.phpmyadmin.net/*

10. FLANAGAN, David. *JavaScript: The Definitive Guide*. 6ª edición. O'Reilly, 2011. ISBN 978-0-596-80552-4

11. SAWYER, David. *JavaScript y jQuery*. Ediciones Anaya, 2012. ISBN: 978-84-415-3151-2.

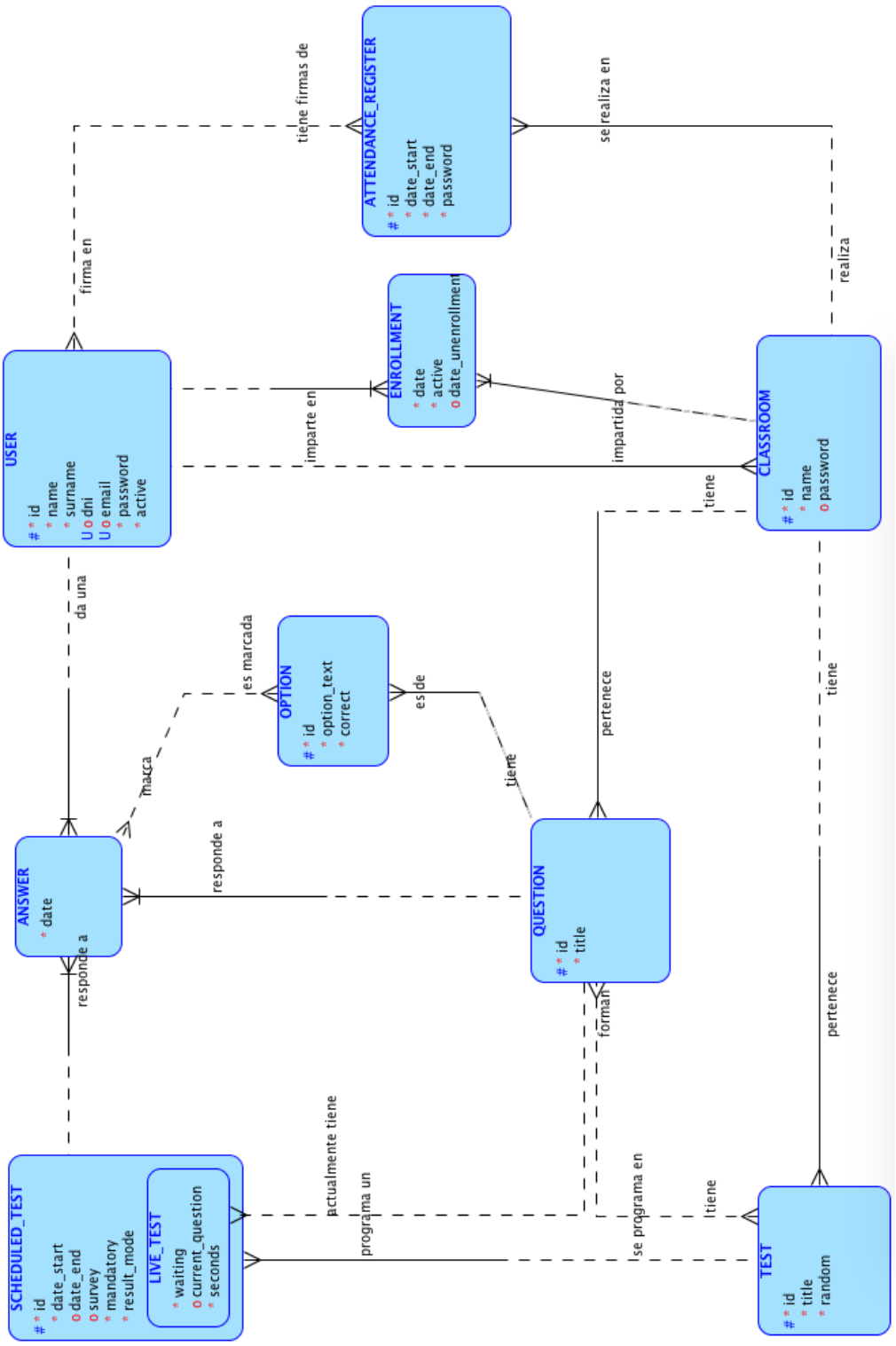
12. CHACON, S; STRAUB, B. *Pro Git*. 2ª edición. Apress, 2014. ISBN: 9781484200773

13. NIELSEN, Jakob. *Usability as Barrier to Entry*. Nielsen Norman Group, 28 de noviembre de 1999. “Impatient users imply increasing difficulties in launching new websites, since the users will not bother with anything that requires additional learning time. Usability becomes a barrier to entry: a new site will fail unless users can grasp it in a few seconds.” [En línea] Disponible en: *https://www.nngroup.com/articles/usability-as-barrier-to-entry/*

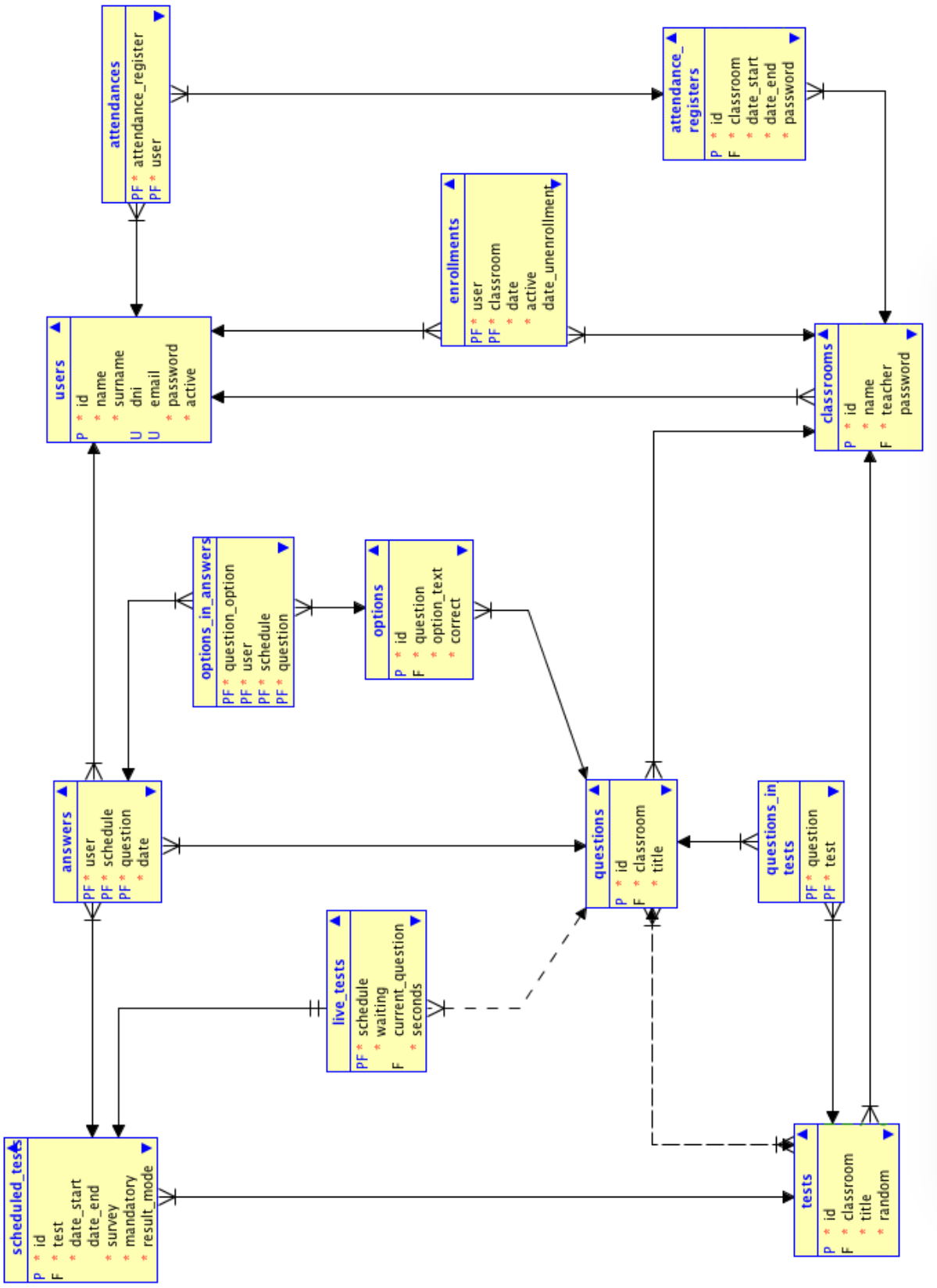
14. VASWANI, V. *Fundamentos de PHP*. McGraw-Hill Interamericana, 2011. ISBN: 9789701071328

Anexo I. Descripción de la base de datos

I.1. Diagrama entidad-relación



I.2. Diagrama relacional



I.3. Descripción de las tablas

A continuación se ofrecerá una descripción de cada una de las tablas de la base de datos desarrollada para la realización de este trabajo.

Para referirse a una columna de una tabla se empleará únicamente su nombre, salvo cuando se relacionen varias tablas, en cuyo caso cada columna estará precedida del nombre de la tabla. Por ejemplo, `classrooms.teacher` indica la columna `teacher` de la tabla `classrooms`.

I.3.1. Tabla `users`

La tabla `users` representa a un usuario del sistema.

Columnas de la tabla `users`

Columna	Descripción	Opción	Tipo	Predet.
`id`	Identificador del usuario.	No	`int(11)`	
`name`	Nombre del usuario (sin apellidos).	No	`varchar(50)`	
`surname`	Apellidos del usuario.	No	`varchar(100)`	
`dni`	Documento Nacional de Identidad o similar del usuario.	Sí	`varchar(9)`	`NULL`
`email`	Dirección de correo electrónico del usuario. Servirá para iniciar sesión.	Sí	`varchar(255)`	`NULL`
`password`	Contraseña del usuario.	No	`varchar(255)`	
`active`	Indica si el usuario se encuentra activo o no.	No	`tinyint(1)`	`1`

Claves de la tabla `users`:

- Clave primaria: `id`
- Clave única 1: `dni`
- Clave única 2: `email`

I.3.2. Tabla `classrooms`

La tabla `classrooms` representa a una clase creada en el sistema.

Columnas de la tabla `classrooms`

Columna	Descripción	Opción	Tipo	Predet.
`id`	Identificador de la clase.	No	`int(11)`	
`name`	Nombre de la clase.	No	`varchar(255)`	
`teacher`	Identificador del usuario creador de esta clase, que actúa como profesor.	No	`int(11)`	
`password`	Documento Nacional de Identidad o similar del usuario.	Sí	`varchar(255)`	`NULL`

Claves de la tabla `classrooms`:

- Clave primaria: `id`

Relaciones de la tabla `classrooms`:

- `classrooms.teacher` debe ser un valor de `users.id`

I.3.3. Tabla `enrollments`

La tabla `enrollments` representa las inscripciones de los usuarios (`users`) en las distintas clases (`classrooms`).

Columnas de la tabla `enrollments`

Columna	Descripción	Opción	Tipo	Predet.
`user`	Identificador del usuario.	No	`int(11)`	
`classroom`	Identificador de la clase.	No	`int(11)`	
`date`	Fecha y hora de inscripción.	No	`datetime`	
`active`	Indica si el usuario está activo o no.	No	`tinyint(1)`	`1`
`date_unenrollment`	Fecha y hora de la desinscripción.	Sí	`datetime`	`NULL`

Claves de la tabla `enrollments`:

- Clave primaria: `user`, `classroom`

Relaciones de la tabla `enrollments`:

- `enrollments.user` debe ser un valor de `users.id`
- `enrollments.classroom` debe ser un valor de `classrooms.id`

I.3.4. Tabla `attendance_registers`

La tabla `attendance_registers` representa los registros de asistencia creados en una clase (`classrooms`).

Columnas de la tabla `attendance_registers`

Columna	Descripción	Opción	Tipo	Predeterminado
`id`	Identificador del registro de asistencia.	No	`int(11)`	
`classroom`	Identificador de la clase a la que pertenece el registro.	No	`int(11)`	
`date_start`	Fecha y hora de inicio del registro.	No	`datetime`	`CURRENT_TIMESTAMP`
`date_end`	Fecha y hora de cierre del registro.	No	`datetime`	
`password`	Contraseña del registro.	No	`varchar(150)`	

Claves de la tabla `attendance_registers`:

- Clave primaria: `id`

Relaciones de la tabla `attendance_registers`:

- `attendance_registers.classroom` debe ser un valor de `classrooms.id`

I.3.5. Tabla `attendances`

La tabla `attendances` representa las asistencias de los usuario (`users`). Es decir, las firma de los usuarios en los registros de asistencia (`attendance_registers`) indicando así que se encuentran presentes en clase.

Columnas de la tabla `attendances`

Columna	Descripción	Opción	Tipo	Predet.
`attendance_register`	Identificador del registro de asistencia.	No	`int(11)`	
`user`	Identificador del usuario que asiste.	No	`int(11)`	

Claves de la tabla `attendances`:

- Clave primaria: `attendance_register`, `user`

Relaciones de la tabla `attendances`:

- `attendances.attendance_register` debe ser un valor de `attendance_registers.id`
- `attendances.user` debe ser un valor de `users.id`

I.3.6. Tabla `questions`

La tabla `questions` representa las preguntas creadas en una clase (`classrooms`).

Columnas de la tabla `questions`

Columna	Descripción	Opción	Tipo	Predet.
`id`	Identificador de la pregunta.	No	`int(11)`	
`classroom`	Identificador de la clase a la que pertenece la pregunta.	No	`int(11)`	
`title`	Título o texto de la pregunta.	No	`text`	

Claves de la tabla `questions`:

- Clave primaria: `id`

Relaciones de la tabla `questions`:

- `questions.classroom` debe ser un valor de `classrooms.id`

I.3.7. Tabla `options`

La tabla `options` representa las opciones de las preguntas creadas en una clase (`questions`).

Columnas de la tabla `options`

Columna	Descripción	Opción	Tipo	Predet.
`id`	Identificador de la opción.	No	`int(11)`	
`question`	Identificador de la pregunta a la que pertenece la opción.	No	`int(11)`	
`option_text`	Texto de la opción.	No	`text`	
`correct`	Indica si la opción es una opción correcta o no.	No	`tinyint(1)`	

Claves de la tabla `options`:

- Clave primaria: `id`

Relaciones de la tabla `options`:

- `options.question` debe ser un valor de `questions.id`

I.3.8. Tabla `tests`

La tabla `tests` representa los cuestionarios de cada clase (`classrooms`).

Columnas de la tabla `tests`

Columna	Descripción	Opción	Tipo	Predet.
`id`	Identificador de la opción.	No	`int(11)`	
`classroom`	Identificador de la pregunta a la que pertenece la opción.	No	`int(11)`	
`title`	Texto de la opción.	No	`varchar(500)`	
`random`	Indica si el cuestionario se ha creado de manera aleatoria o no.	No	`tinyint(1)`	`0`

Claves de la tabla `tests`:

- Clave primaria: `id`

Relaciones de la tabla `tests`:

- `tests.classroom` debe ser un valor de `classrooms.id`

I.3.9. Tabla `questions_in_tests`

La tabla `questions_in_tests` representa la relación entre las preguntas (`questions`) y los cuestionarios (`tests`), es decir, qué pregunta pertenece a cada cuestionario.

Columnas de la tabla `questions_in_tests`

Columna	Descripción	Opción	Tipo	Predet.
`question`	Identificador de la pregunta.	No	`int(11)`	
`test`	Identificador del cuestionario al que pertenece la pregunta.	No	`int(11)`	

Claves de la tabla `questions_in_tests`:

- Clave primaria: `question`, `test`

Relaciones de la tabla `questions_in_tests`:

- `questions_in_tests.question` debe ser un valor de `questions.id`
- `questions_in_tests.test` debe ser un valor de `tests.id`

I.3.10. Tabla `scheduled_tests`

La tabla `scheduled_tests` almacena las programaciones de los cuestionarios. Es decir, cada una de las veces en las que un cuestionario (`tests`) se ha programado para ser realizado por los alumnos.

Columnas de la tabla `scheduled_tests`:

Columna	Descripción	Opción	Tipo	Predeterminado
`id`	Identificador de la programación.	No	`int(11)`	
`test`	Identificador del cuestionario que es programado.	No	`int(11)`	
`date_start`	Fecha y hora de inicio de la programación.	No	`datetime`	`CURRENT_TIMESTAMP`
`date_end`	Fecha y hora de cierre de la programación.	Sí	`datetime`	`NULL`
`survey`	Indica si se trata de una encuesta.	No	`tinyint(1)`	`0`
`mandatory`	Indica si es obligatoria la realización de esta programación o no.	No	`tinyint(1)`	`0`
`result_mode`	Indica el modo en el que se mostrarán los resultados a un usuario.	No	`int(11)`	`1`

Claves de la tabla `scheduled_tests`:

- Clave primaria: `id`

Relaciones de la tabla `scheduled_tests`:

- `scheduled_test.test` debe ser un valor de `tests.id`

I.3.11. Tabla `live_tests`

La tabla `scheduled_tests` almacena las programaciones de los cuestionarios en directo y las preguntas (`questions`) por las que se encuentran cada uno de ellos. Se trata de un caso especial de programación de un cuestionario (`scheduled_tests`).

Columnas de la tabla `live_tests`:

Columna	Descripción	Opción	Tipo	Predet.
`schedule`	Identificador de la programación del cuestionario.	No	`int(11)`	
`waiting`	Indica si el cuestionario en directo se encuentra en espera.	No	`tinyint(1)`	`1`
`current_question`	Identificador de la pregunta actualmente en curso del cuestionario en directo.	Sí	`int(11)`	`NULL`
`seconds`	Segundos asignados por el profesor para responder a cada pregunta.	No	`int(11)`	

Claves de la tabla `live_tests`:

- Clave primaria: `schedule`

Relaciones de la tabla `live_tests`:

- `live_tests.schedule` debe ser un valor de `scheduled_tests.id`
- `live_tests.current_question` debe ser un valor de `questions.id`

I.3.12. Tabla `answers`

La tabla `answers` almacena las respuestas de los usuarios (`users`) a cada una de las preguntas de un cuestionario (`questions`) para cada una de las programaciones (`scheduled_tests`).

Columnas de la tabla `answers`:

Columna	Descripción	Opción	Tipo	Predeterminado
`user`	Identificador del usuario.	No	`int(11)`	
`schedule`	Identificador de la programación del cuestionario.	No	`int(11)`	
`question`	Identificador de la pregunta del cuestionario a la que se responde.	No	`int(11)`	
`date`	Fecha y hora en la que se dio la respuesta.	No	`datetime`	`CURRENT_TIMESTAMP`

Claves de la tabla `answers`:

- Clave primaria: `user`, `schedule`, `question`

Relaciones de la tabla `answers`:

- `answers.user` debe ser un valor de `users.id`
- `answers.schedule` debe ser un valor de `scheduled_tests.id`
- `answers.question` debe ser un valor de `questions.id`

I.3.13. Tabla `options_in_answers`

La tabla `options_in_answers` relaciona cada respuesta a una pregunta (`answers`) con las opciones que da el alumno como respuesta (`options`).

Columnas de la tabla `options_in_answers`:

Columna	Descripción	Opción	Tipo	Predet.
`question_option`	Identificador de la opción respondida.	No	`int(11)`	
`user`	Identificador del usuario.	No	`int(11)`	
`schedule`	Identificador de la programación del cuestionario.	No	`int(11)`	
`question`	Identificador de la pregunta del cuestionario a la que se responde.	No	`int(11)`	

Claves de la tabla `options_in_answers`:

- Clave primaria: `question_option`, `user`, `schedule`, `question`

Relaciones de la tabla `options_in_answers`:

- `options_in_answers.question_option` debe ser un valor de `options.id`
- `options_in_answers.user` debe ser un valor de `users.id`
- `options_in_answers.schedule` debe ser un valor de `scheduled_tests.id`
- `options_in_answers.question` debe ser un valor de `questions.id`

Anexo II. Documentación de código

En este anexo se describen cada uno de los archivos que componen el código desarrollado para la aplicación.

II.1. Árbol de ficheros

El siguiente árbol muestra la jerarquía de los ficheros que componen el código del proyecto:

```
tfg/
|--config.php
|--index.php
|--init.php
|--load_notifications.php
|--logout.php
|--classes/
|   |--AttendanceRegister.php
|   |--Classroom.php
|   |--DB.php
|   |--Error.php
|   |--ExceptionMSG.php
|   |--Interf.php
|   |--LiveTest.php
|   |--Option.php
|   |--Question.php
|   |--ScheduledTest.php
|   |--Security.php
|   |--Stats.php
|   |--Test.php
|   |--User.php
|
|--classroom/
|   |--delete.php
|   |--join.php
|   |--new.php
|   |--student.php
|   |--students.php
|   |--teacher.php
|   |--unenroll.php
|   |
|   |--attendance/
|   |   |--confirm.php
|   |   |--new.php
|   |   |--student.php
|   |   |--teacher.php
```

```
|   |   |--teacher_load.php
|   |
|   |--live/
|   |   |--end-waiting.php
|   |   |--next.php
|   |   |--random.php
|   |   |--release.php
|   |   |--saveanswer.php
|   |   |--start.php
|   |   |--student.php
|   |   |--teacher-waiting.php
|   |   |--teacher.php
|   |   |--teacher_load.php
|   |
|   |--question/
|   |   |--delete.php
|   |   |--index.php
|   |   |--new.php
|   |
|   |--stats/
|   |   |--details.php
|   |   |--exportattendance.php
|   |   |--exportmarks.php
|   |   |--index.php
|   |   |--register.php
|   |   |--student-result.php
|   |   |--student.php
|   |   |--test.php
|   |
|   |--test/
|       |--correct.php
|       |--delete.php
|       |--edit.php
|       |--new.php
|       |--schedule-edit.php
|       |--schedule-end.php
|       |--schedule-remove.php
|       |--schedule.php
|       |--student.php
|       |--teacher.php
|
|--includes/
|   |--css/
|   |   |--bootstrap-theme.css
|   |   |--bootstrap-theme.min.css
|   |   |--bootstrap.css
|   |   |--bootstrap.min.css
|   |   |--custom.css
|   |
|   |--fonts/
|   |   |--glyphicons-halflings-regular.eot
```

```
|   |   |--glyphicons-halflings-regular.svg
|   |   |--glyphicons-halflings-regular.ttf
|   |   |--glyphicons-halflings-regular.woff
|   |   |--glyphicons-halflings-regular.woff2
|   |
|   |--js/
|       |--bootstrap.js
|       |--bootstrap.min.js
|       |--jquery.min.js
|       |--load-notifications.js
|       |--npm.js
|
|--login/
|   |--index.php
|
|--profile/
|   |--changemail.php
|   |--changepassword.php
|   |--delete.php
|   |--index.php
|   |--update.php
|
|--signin/
    |--index.php
```

II.2. Clases

II.2.1. Clase `AttendanceRegister`

Clase facilita la manipulación de registros de asistencia.

Funciones:

`public function __construct()`
Descripción: Constructor de la clase `AttendanceRegister`.

`public static function newByID()`
Descripción: Devuelve una instancia de la clase `AttendanceRegister` a partir de un ID del registro, estableciendo sus variables de objeto según los valores guardados en la base de datos.
Return: Un objeto `AttendanceRegister`. `Null` en caso de que no exista el ID o error.

`public function getID()`
Descripción: Devuelve el valor de la variable `$_id` del objeto.
Return: El valor de la variable `$_id`.

`public function setID()`
Descripción: Establece el valor de la variable `$_id` del objeto.
Parámetros: `$question_id`: ID del registro de asistencia..

`public function` **`getClassroom()`**

Descripción:

Devuelve el valor de la variable `$_classroom` del objeto, que indica el ID de la clase a la que pertenece el registro de asistencia.

Return:

El valor de la variable `$_classroom`.

`public function` **`setClassroom()`**

Descripción:

Establece el valor de la variable `$_classroom` del objeto, que indica el ID de la clase a la que pertenece el registro de asistencia.

Parámetros:

`$classroom`: ID de la clase a la que pertenece el registro de asistencia.

`public function` **`getTime()`**

Descripción:

Devuelve el valor de la variable `$_time` del objeto, que indica número de segundos que estará activo el registro de asistencia.

Return:

El valor de la variable `$_time`.

`public function` **`setTime()`**

Descripción:

Establece el valor de la variable `$_time` del objeto, que indica número de segundos que estará activo el registro de asistencia.

Parámetros:

`$time`: Duración en número de segundos.

`public function` **`getPassword()`**

Descripción:

Devuelve el valor de la variable `$_password` del objeto, que almacena la contraseña del registro de asistencia.

Return:

El valor de la variable `$_password`.

`public function` **`setPassword()`**

Descripción:

Establece el valor de la variable `$_password` del objeto, que almacena la contraseña del registro de asistencia.

Parámetros:

`$password`: Cadena de caracteres con la contraseña.

`public function` **`getDateStart()`**

Descripción:

Obtiene la fecha de inicio del registro de asistencia en la base de datos.

Return:

Una cadena de caracteres que representa la fecha. En caso de error, devuelve `null`.

`public function` **`getDateEnd()`**

Descripción:

Obtiene la fecha de finalización del registro de asistencia en la base de datos.

Return:

Una cadena de caracteres que representa la fecha. En caso de error, devuelve `null`.

`public function` **`isLive()`**

Descripción:

Consulta si el registro de asistencia está activo en este momento.

Return:

`true` en caso de que esté activo, `false` en caso contrario.

`public function` **`finished()`**

Descripción:

Consulta si el registro de asistencia ha finalizado.

Return:

`true` en caso de que haya finalizado, `false` en caso contrario.

`public function` **`create()`**
Descripción: Guarda el registro de asistencia en la base de datos.
Return: `true` en caso de éxito, `false` en caso contrario.

`public function` **`confirm()`**
Descripción: Confirma la asistencia de un usuario en el registro de asistencia.
Parámetros: `$user_id`: Número de ID del usuario.
Return: `true` en caso de éxito, `false` en caso contrario.

`public function` **`attended()`**
Descripción: Consulta si un usuario ha confirmado su asistencia en el registro de asistencia.
Parámetros: `$user_id`: Número de ID del usuario.
Return: `true` en caso afirmativo, `false` en caso contrario.

`public function` **`attendance()`**
Descripción: Consulta el número de usuarios que han confirmado su asistencia en el registro.
Return: Número de asistentes; `false` en caso de error.

`public function` **`attendanceList()`**
Descripción: Obtiene una lista de los ID de usuarios que confirmaron su asistencia en el registro.
Return: Array con los ID de los usuarios asistentes; `false` en caso de error.

II.2.2. Clase `Classroom`

Clase que representa a un grupo de alumnos al frente del cual se encuentra un profesor.

Funciones:

`public function __construct()`
Descripción: Constructor de la clase `Classroom`.
Parámetros: Admite tres opciones: ◦ 1 parámetro: será el número de ID de la clase. Para instanciar clases ya existentes. ◦ 2 parámetros: nombre de la clase y número de ID del profesor. Para instanciar clases aún no creadas. ◦ 3 parámetros: nombre de la clase, número de ID del profesor y contraseña. Para instanciar clases aún no creadas.

`public function getID()`
Descripción: Devuelve el valor de la variable `$_id` del objeto.
Return: El valor de la variable `$_id`.

`public function setID()`
Descripción: Establece el valor de la variable `$_id` del objeto.
Parámetros: `$id`: ID de la clase.

`public function getName()`
Descripción: Devuelve el valor de la variable `$_name` del objeto, que representa el nombre de la clase.
Return: El valor de la variable `$_name`.

`public function` **`setName()`**
Descripción: Establece el valor de la variable `$_name` del objeto.
Parámetros: `$name`: Cadena de caracteres que representa el nombre de la clase.

`public function` **`getTeacher()`**
Descripción: Devuelve el valor de la variable `$_teacher` del objeto, que representa el ID del profesor.
Return: El valor de la variable `$_teacher`.

`public function` **`setTeacher()`**
Descripción: Establece el valor de la variable `$_teacher` del objeto, que representa el ID del profesor.
Parámetros: `$teacher`: Número con el ID de usuario del profesor.

`public function` **`getPassword()`**
Descripción: Devuelve el valor de la variable `$_password` del objeto, que almacena la contraseña de la clase.
Return: El valor de la variable `$_password`.

`public function` **`setPassword()`**
Descripción: Establece el valor de la variable `$_password` del objeto, que almacena la contraseña de la clase.
Parámetros: `$password`: Cadena de caracteres con la contraseña de la clase.

`public function` **`setClassroomByID()`**
Descripción: Dado el ID de la claseflo, instancia las variables del objeto con la información correspondiente a dicha clase en la base de datos.
Parámetros: `$classroom_id`: Número de ID de la clase.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`getTeacherFullName()`**
Descripción: Obtiene el nombre completo del profesor.
Return: Cadena de caracteres con el nombre del profesor.

`public function` **`create()`**
Descripción: Guarda la clase en la base de datos.
Return: `true` en caso de éxito, `false` en caso contrario.

`public function` **`hasPassword()`**
Descripción: Comprueba si una clase tiene contraseña.
Return: `true` en caso afirmativo, `false` en caso contrario.

`public function` **`pass()`**
Descripción: Comprueba si la contraseña proporcionada es la de la clase.
Parámetros: `$password`: String con la contraseña.
Return: `true` si la contraseña es correcta, `false` en caso contrario.

`public function` **`delete()`**
Descripción: Borra la clase de la base de datos, así como toda su actividad.
Return: `true` si el borrado se ha realizado con éxito, `false` en caso contrario.

`public function` **`isStudent()`**
Descripción: Consulta si un usuario es alumno de la clase.
Parámetros: `$user_id`: Número de ID del usuario.
Return: `true` si es alumno, `false` en caso contrario.

`public function` **`wasStudent()`**
Descripción: Consulta si un usuario fue alumno de la clase.
Parámetros: `$user_id`: Número de ID del usuario.
Return: `true` si fue alumno, `false` en caso contrario.

`public function` **`isTeacher()`**
Descripción: Consulta si un usuario es profesor de la clase.
Parámetros: `$user_id`: Número de ID del usuario..
Return: `true` si es profesor, `false` en caso contrario.

`public function` **`hasAccess()`**

Descripción:
Consulta si un usuario tiene acceso a la clase.

Parámetros:
`$user_id`: Número de ID del usuario..

Return:
`true` si tiene acceso, `false` en caso contrario.

`public function` **`getStudents()`**

Descripción:
Obtiene una lista de ID alumnos de la clase. No tiene en cuenta si el alumno ha borrado su cuenta o si se ha desmatriculado.

Return:
Array con los ID de los alumnos de la clase.

`public function` **`numStudents()`**

Descripción:
Obtiene el número de alumnos de la clase.

Return:
Número de alumnos de la clase.

`public function` **`getTests()`**

Descripción:
Obtiene los cuestionarios de la clase generados de forma manual.

Return:
Objeto mysqli con los cuestionarios generados de forma manual.

`public function` **`getAllTests()`**

Descripción:
Obtiene los cuestionarios de la clase incluyendo aleatorios.

Return:
Objeto mysqli con los cuestionarios de la clase.

`public function` **`getQuestions()`**

Descripción:
Obtiene los ID de las preguntas de la clase.

Return:
Array con los ID de las preguntas de la clase.

`public function` **`numQuestions()`**

Descripción:
Obtiene el número de preguntas de la clase.

Return:
Número de preguntas de la clase.

`public function` **`getRandomQuestions()`**

Descripción:
Obtiene un número de ID de preguntas de la clase seleccionadas de forma aleatoria.

Parámetros:
`$numquestions`: Número de preguntas a obtener.

Return:
Array con los ID de las preguntas seleccionadas aleatoriamente.

`public function` **`enroll()`**

Descripción:
Inscribe a un alumno en la clase.

Parámetros:
`$user_id`: Número de ID del usuario.

Return:
`true` en caso de éxito, `false` en caso contrario.

`public function` **`enrollmentDate()`**

Descripción:
Desmatricula a un alumno de la clase.

Parámetros:
`$user_id`: Número de ID del usuario.

Return:
`true` en caso de éxito, `false` en caso contrario.

`public function` **`unenroll()`**

Descripción:
Obtiene la fecha en la que un alumno se inscribió en la clase.

Parámetros:
`$user_id`: Número de ID del usuario.

Return:
Una cadena de caracteres que representa la fecha. En caso de error, devuelve `null`.

`public function` **`registers()`**

Descripción:
Obtiene un los registros de asistencia realizados en la clase ordenados por fecha.

Return:
Array con los ID de los registros de asistencia.

`public function` **`numRegisters()`**

Descripción:
Obtiene el número de registros de asistencia de la clase.

Return:
Número de registros de asistencia.

`public function` **`exportAttendance()`**

Descripción:
Devuelve un array bidimensional con la asistencia a la clase. Útil para exportarlo en formato CSV.

Return:
Array bidimensional con la asistencia a la clase.

`public function` **`printAttendance()`**

Descripción:
Imprime una tabla HTML con la asistencia a la clase.

`public function` **`studentAttendance()`**

Descripción:

Obtiene la asistencia de un alumno a la clase.

Parámetros:

`$user_id`: Número de ID del usuario.

Return:

Número de asistencias del alumno. En caso de error devuelve `false`.

`private function` **`extremAttendance()`**

Descripción:

Obtiene la asistencia extrema (máxima o mínima) de la clase.

Parámetros:

`$extrema`: Puede tomar los valores `max` o `min`, según se desee la asistencia máxima o mínima. El valor por defecto es `max`.

Return:

Número de asistentes. En caso de error devuelve `false`.

`private function` **`extremAttendanceDate()`**

Descripción:

Obtiene la fecha de asistencia extrema (máxima o mínima) de la clase.

Parámetros:

`$extrema`: Puede tomar los valores `max` o `min`, según se desee la asistencia máxima o mínima. El valor por defecto es `max`.

Return:

Cadena de caracteres que representa la fecha. En caso de error devuelve `null`.

`public function` **`maxAttendance()`**

Descripción:

Obtiene la asistencia máxima de la clase.

Return:

Número de asistentes. En caso de error devuelve `false`.

`public function` **`minAttendance()`**
Descripción: Obtiene la asistencia mínima de la clase.
Return: Número de asistentes. En caso de error devuelve `false`.

`public function` **`maxAttendanceDate()`**
Descripción: Obtiene la fecha de asistencia máxima de la clase.
Return: Cadena de caracteres que representa la fecha. En caso de error devuelve `null`.

`public function` **`minAttendanceDate()`**
Descripción: Obtiene la fecha de asistencia mínima de la clase.
Return: Cadena de caracteres que representa la fecha. En caso de error devuelve `null`.

`public function` **`mediumAttendance()`**
Descripción: Obtiene la asistencia media de la clase.
Return: Número medio de asistentes. En caso de error devuelve `false`.

`public function` **`medianAttendance()`**
Descripción: Obtiene la asistencia mediana de la clase.
Return: Mediana del número de asistentes. En caso de error devuelve `false`.

`public function` **`numFinishedLiveTests()`**

Descripción:

Obtiene el número de cuestionarios realizados en directo en la clase.

Return:

Número de cuestionarios realizados en directo. En caso de error devuelve `false`.

`public function` **`numFinishedScheduledTests()`**

Descripción:

Obtiene el número de cuestionarios programados realizados en la clase.

Return:

Número de cuestionarios programados realizados. En caso de error devuelve `false`.

`public function` **`finishedScheduledTests()`**

Descripción:

Obtiene los cuestionarios programados realizados en la clase.

Return:

Array con los ID de las programaciones de cuestionarios realizadas. En caso de error devuelve `false`.

`public function` **`mediumMarkTests()`**

Descripción:

Obtiene la nota media de los cuestionarios realizados la clase.

Return:

Número con la nota media. En caso de error devuelve `false`.

`public function` **`mediumMark()`**

Descripción:

Obtiene la nota media de los alumnos de la clase.

Return:

Número con la nota media.

```
public function medianMarkTests()
```

Descripción:

Obtiene la mediana de las notas medias de los cuestionarios realizados la clase.

Return:

Número con la nota mediana.

```
public function medianMark()
```

Descripción:

Obtiene la mediana de las notas medias de los alumnos de la clase.

Return:

Número con la nota mediana.

```
public function exportMarks()
```

Descripción:

Devuelve un array bidimensional con las notas de la clase. Útil para exportarlo en formato CSV.

Return:

Array bidimensional con las notas de la clase.

```
public function printMarks()
```

Descripción:

Imprime una tabla HTML con las notas de la clase.

```
public function finishedSurveys()
```

Descripción:

Obtiene las encuestas realizadas en clase.

Return:

Array con los números de ID de las encuestas realizadas en la clase.

II.2.3. Clase `DB`

Clase que facilita conexiones a la base de datos siguiendo el patrón *singleton* y permitiendo una única conexión. Actúa como *wrapper* de la extensión `mysqli`.

Funciones:

`private function` **`__construct()`**
Descripción: Constructor de la clase `DB`. Establece conexiones a la base de datos cuyos parámetros de conexión se indican en el archivo de configuración *config.php*.

`public function` **`__destruct()`**
Descripción: Destructor de la clase `DB`.

`public static function` **`getInstance()`**
Descripción: Devuelve una instancia de la clase `DB`, evitando crear nuevas instancias si ya existe previamente una.
Return: Un objeto `DB`.

`public function` **`getConnection()`**
Descripción: Devuelve una conexión entre PHP y la base de datos en un objeto `mysqli`.
Return: Un objeto `mysqli`.

`public static function` **`query()`**

Descripción:
Realiza una consulta a la base de datos.

Parámetros:
`$query`: Cadena de texto con la consulta.

Return:
`false` en caso de error. Si una consulta del tipo SELECT, SHOW, DESCRIBE o EXPLAIN es exitosa, devolverá un objeto `mysqli_result`. Para otras consultas exitosas, devolverá `true`.

`public static function` **`autocommit()`**

Descripción:
Activa o desactiva las modificaciones de la base de datos autoconsignadas.

Parámetros:
`$mode`: Valor booleano: true para activar la autoconsignación, false en caso contrario.

Return:
`true` en caso de éxito, `false` en caso contrario.

`public static function` **`commit()`**

Descripción:
Consigna la transacción actual.

Return:
`true` en caso de éxito, `false` en caso contrario.

`public static function` **`rollback()`**

Descripción:
Revierte la transacción actual.

Return:
`true` en caso de éxito, `false` en caso contrario.

`public static function` **`real_escape_string()`**

Descripción:

Escapa los caracteres especiales de una cadena para usarla en una consulta a la base de datos.

Parámetros:

`$string`: Cadena a escapar.

Return:

La cadena escapada.

`public static function` **`nextID()`**

Descripción:

Devuelve el siguiente número de ID para la inserción en una tabla con un campo `id` autoincremental.

Parámetros:

`$table`: Nombre de la tabla de la base de datos.

Return:

En caso de éxito devuelve el siguiente número de ID, `false` en caso contrario.

`public static function` **`exists()`**

Descripción:

Consulta si en una tabla de la base de datos existe un único valor en el campo `id`.

Parámetros:

`$table`: Nombre de la tabla de la base de datos.

`$id`: valor de ID que se quiere consultar si existe.

Return:

`true` si existe un único valor, `false` en caso contrario.

II.2.4. Clase `Error`

Clase que facilita el manejo de errores en formularios.

Funciones:

`public function __construct()`
Descripción: Constructor de la clase `Error`.

`public function add()`
Descripción: Añade un error.
Parámetros: `$message`: Cadena de texto con el mensaje de error a añadir.

`public function getList()`
Descripción: Devuelve una lista con los errores almacenados.
Return: Array con los mensajes de error almacenados.

`public function printHTML()`
Descripción: Imprime la lista con los errores almacenados.

`public function isEmpty()`
Descripción: Consulta si la lista de errores está vacía.
Return: `true` si no hay errores, `false` en caso contrario.

II.2.5. Clase `ExceptionMSG`

Clase para manipular los mensajes de excepción.

Funciones:

`public static function` **`printE()`**
Descripción: Imprime información relativa a la excepción capturada en el caso de que esté activo el modo de desarrollo.[11]
Parámetros: `$exception`: La excepción de la que se desea imprimir sul mensaje.

II.2.6. Clase `Interf`

Clase que facilita la generación de la interfaz gráfica en HTML.

Funciones:

`public function` **`__construct()`**
Descripción: Constructor de la clase `Interf`.
Parámetros: `$title`: Nombre de la página a generar. `$menu`: Array asociativo *Nombre => URL* que representa el menú superior.

`public function` **`getTitle()`**
Descripción: Devuelve el valor de la variable `$_title` del objeto.
Return: El valor de la variable `$_title` del objeto.

11 El modo de desarrollo se activa o desactiva mediante el flag `DEVELOPERMODE` en el archivo de configuración *config.php* y sirve para mostrar u ocultar mensajes de error y advertencias.

`public function` **`setTitle()`**

Descripción:

Establece el valor de la variable `$_title` del objeto.

Parámetros:

`$title`: Cadena de texto con el nombre de la página.

`public function` **`getMenu()`**

Descripción:

Devuelve el valor de la variable `$_menu` del objeto.

Return:

Cadena de texto con el nombre de la página.

`public function` **`setMenu()`**

Descripción:

Establece el valor de la variable `$_menu` del objeto.

Parámetros:

`$menu`: Array asociativo *Nombre => URL* que representa el menú superior.

`public function` **`printHead()`**

Descripción:

Imprime el inicio del código HTML, incluyendo toda la cabecera y la etiqueta de apertura del cuerpo.

`public function` **`printEnd()`**

Descripción:

Imprime el final del código HTML, incluyendo *scripts* finales y la etiqueta de cierre del cuerpo.

`public function` **`printHeader()`**

Descripción:

Imprime el encabezado de una página, incluyendo el título de la página, el menú y la barra de navegación.

Parámetros:

`$breadcrumb`: Array asociativo *Nombre => URL* que representa la barra de navegación.
`$active`: Número que indica el enlace del menú que permanece activo (por defecto toma el valor `null`, que no activa ningún enlace).

`public function` **`printFooter()`**
Descripción: Imprime el pie de página.

II.2.7. Clase `LiveTest`

Clase que facilita la realización de un cuestionario en directo. Hereda de la clase `ScheduledTest`, que representa una programación de un cuestionario.

Funciones:

`public function` **`__construct()`**
Descripción: Constructor de la clase `LiveTest`.
Parámetros: Admite dos opciones: ○ 1 parámetro: será el número de ID de la programación del cuestionario (`ScheduledTest`). Para instanciar cuestionarios en directo ya existentes. ○ 2 parámetros: número de ID del cuestionario y cadena de texto `new`, para instanciar cuestionarios en directo cuando no se ha creado aún una programación para dicho cuestionario.

`public function` **`getWaiting()`**
Descripción: Devuelve el valor de la variable `$_waiting` del objeto, que indica si el cuestionario en directo se encuentra en espera.
Return: El valor de la variable `$_waiting` del objeto.

`public function` **`setWaiting()`**
Descripción: Establece el valor de la variable `$_waiting` del objeto, que indica si el cuestionario en directo se encuentra en espera.
Parámetros: `$waiting`: Booleano que indica si está a la espera.

`public function` **`isWaiting()`**
Descripción: Consulta si el cuestionario en directo se encuentra en espera.
Return: `true` si se encuentra a la espera, `false` en caso contrario.

`public function` **`getCurrentQuestion()`**
Descripción: Devuelve el valor de la variable `$_current_question` del objeto, que indica el ID de la pregunta actual.
Return: El valor de la variable `$_current_question` del objeto.

`public function` **`setCurrentQuestion()`**
Descripción: Establece el valor de la variable `$_current_question` del objeto, que indica el ID de la pregunta actual.
Parámetros: `$question`: Número de ID de la pregunta.

`public function` **`getSeconds()`**
Descripción: Devuelve el valor de la variable `$_seconds` del objeto, que indica el tiempo para responder a cada pregunta.
Return: El valor de la variable `$_seconds` del objeto.

`public function` **`setSeconds()`**
Descripción: Establece el valor de la variable `$_seconds` del objeto, que indica el tiempo para responder a cada pregunta.
Parámetros: `$seconds`: Tiempo en número de segundos.

`public function` **`getResultMode()`**

Descripción:

Devuelve el valor de la variable `$_result_mode` del objeto, que indica el tipo de salida que obtendrán los alumnos tras realizar el cuestionario.

Return:

El valor de la variable `$_result_mode` del objeto.

`public function` **`setResultMode()`**

Descripción:

Establece el valor de la variable `$mode` del objeto que indica el tipo de salida que obtendrán los alumnos tras realizar el cuestionario.

Parámetros:

`$mode`: Número que indica el tipo de salida que obtendrán los alumnos tras realizar el cuestionario. Hay dos modos:

- 0: Tras contestar, muestra nota y detalles al alumno.
- 1: Tras contestar, muestra únicamente la nota al alumno

En el caso de que el parámetro tenga otro valor, se establecerá el modo por defecto.[12]

`public function` **`setLiveTestByID()`**

Descripción:

Dado el ID de una programación del cuestionario realizada en directo, instancia las variables del objeto con la información correspondiente a dicha programación en la base de datos.

Parámetros:

`$test`: Número de ID de la programación del cuestionario.

Return:

`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`getFirstQuestion()`**

Descripción:

Obtiene la primera pregunta del cuestionario en directo.

Return:

El número de ID de la pregunta. En caso de error devuelve `null`.

12 Indicado con la opción `RESULTMODE` del fichero de configuración *config.php*.

`private function` **`getNextQuestion()`**
Descripción: Obtiene la siguiente pregunta del cuestionario en directo.
Return: El número de ID de la pregunta. En caso de que no queden más preguntas, devuelve `null`.

`public function` **`numCurrentQuestion()`**
Descripción: Obtiene la posición de pregunta actual respecto al total de preguntas del cuestionario en directo.
Return: El número de la posición de la pregunta.

`public function` **`schedule()`**
Descripción: Inicia el cuestionario en directo con el tiempo establecido por defecto.[13]

`public function` **`start()`**
Descripción: Inicia el cuestionario en directo.
Parámetros: `$seconds`: Número segundos para responder cada pregunta. Si no se indica se toma el valor por defecto.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`continueTest()`**
Descripción: Hace avanzar la pregunta actual del cuestionario o lo finaliza si no quedan más preguntas.
Return: `true` mientras se avance, `false` si error o `null` si no puede avanzar porque no quedan más preguntas..

13 Indicado por la opción `SECONDS` del archivo de configuración *config.php*.

`public function` **`end()`**
Descripción: Finaliza el cuestionario en directo.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public static function` **`ensureLiveTests()`**
Descripción: Finaliza aquellos tests en directo que no se hayan finalizado porque el profesor cerrase la ventana.

II.2.8. Clase `Option`

Clase que representa cada una de las opciones asociadas a una pregunta.

Funciones:

`public function` **`__construct()`**
Descripción: Constructor de la clase `Option`.
Parámetros: Admite dos opciones: ● 1 parámetro: será el número de ID de la pregunta. Para instanciar opciones ya existentes. ● 3 parámetros: número de ID de pregunta, texto de la opción y valor booleano que indica si la opción es correcta o no. Para instanciar opciones aún no creadas.

`public function` **`getID()`**
Descripción: Devuelve el valor de la variable `$_id` del objeto.
Return: El valor de la variable `$_id`.

public function **setID()**
Descripción: Establece el valor de la variable `$_id` del objeto.
Parámetros: `$id`: ID de la opción.

public function **getQuestion()**
Descripción: Devuelve el valor de la variable `$_question` del objeto.
Return: El valor de la variable `$_question`.

public function **setQuestion()**
Descripción: Establece el valor de la variable `$_question` del objeto.
Parámetros: `$question`: ID de la pregunta de la que es opción.

public function **getOptionText()**
Descripción: Devuelve el valor de la variable `$_option_text` del objeto.
Return: El valor de la variable `$_option_text`.

public function **setOptionText()**
Descripción: Establece el valor de la variable `$_option_text` del objeto.
Parámetros: `$option`: Texto de la opción.

`public function` **`getCorrect()`**

Descripción:
Devuelve el valor de la variable `$_correct` del objeto, que indica si la opción es correcta.

Return:
El valor de la variable `$_correct`.

`public function` **`setCorrect()`**

Descripción:
Establece el valor de la variable `$_correct` del objeto, que indica si la opción es correcta.

Parámetros:
`$correct`: Valor booleano que indica si la opción es correcta.

`public function` **`isCorrect()`**

Descripción:
Consulta si la opción es correcta.

Return:
El valor de la variable `$_correct` del objeto.

`public function` **`setOptionByID()`**

Descripción:
Dado el ID de la opción, instancia las variables del objeto con la información correspondiente a dicha opción en la base de datos.

Parámetros:
`$option_id`: Número de ID de la opción.

Return:
`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`create()`**

Descripción:
Guarda la opción en la base de datos.

Return:
`true` si se se realiza con éxito, `false` en caso contrario.

`public function` **`update()`**
Descripción: Actualiza la opción en la base de datos.
Parámetros: `$option_text`: Texto de la opción. `$correct`: Valor booleano que indica si la opción es correcta o no.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`delete()`**
Descripción: Borra la opción de la base de datos, así como las veces que fue marcada como respuesta.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

II.2.9. Clase `Question`

Clase que representa una pregunta.

Funciones:

`public function` **`__construct()`**
Descripción: Constructor de la clase `Question`.
Parámetros: Admite dos opciones: • 1 parámetro: será el número de ID de la pregunta. Para instanciar preguntas ya existentes. • 2 parámetros: número de ID de la pregunta y su título. Para instanciar preguntas aún no creadas.

`public function` **`getID()`**
Descripción: Devuelve el valor de la variable `$_id` del objeto.
Return: El valor de la variable `$_id`.

`public function` **`setID()`**
Descripción: Establece el valor de la variable `$_id` del objeto.
Parámetros: `$question_id`: ID de la pregunta.

`public function` **`getClassroom()`**
Descripción: Devuelve el valor de la variable `$_classroom` del objeto, que indica el ID de la clase a la que pertenece la pregunta.
Return: El valor de la variable `$_classroom`.

`public function` **`setClassroom()`**
Descripción: Establece el valor de la variable `$_classroom` del objeto, que indica el ID de la clase a la que pertenece la pregunta.
Parámetros: `$classroom_id`: ID de la clase a la que pertenece la pregunta.

`public function` **`getTitle()`**
Descripción: Devuelve el valor de la variable `$_title` del objeto, que contiene el título de la pregunta.
Return: El valor de la variable `$_title`.

`public function` **`setTitle()`**

Descripción:
Establece el valor de la variable `$_title` del objeto, que contiene el título de la pregunta.

Parámetros:
`$title`: ID de la clase a la que pertenece la pregunta.

`public function` **`setQuestionByID()`**

Descripción:
Dado el ID de la pregunta, instancia las variables del objeto con la información correspondiente a dicha pregunta en la base de datos.

Parámetros:
`$question_id`: Número de ID de la pregunta.

Return:
`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`create()`**

Descripción:
Guarda la pregunta en la base de datos.

Return:
`true` si se se realiza con éxito, `false` en caso contrario.

`public function` **`update()`**

Descripción:
Actualiza la pregunta en la base de datos.

Parámetros:
`$title`: Texto de la pregunta.

Return:
`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`delete()`**

Descripción:
Borra la pregunta de la base de datos, así como sus apariciones en otras tablas.

Return:
`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`getOptions()`**
Descripción: Obtiene los ID de las opciones asociadas a esta pregunta.
Return: Array con los números de ID de las opciones.

`public function` **`getAnswers()`**
Descripción: Obtiene los ID de las opciones asociadas a esta pregunta que son correctas.
Return: Array con los números de ID de las opciones correctas.

`public function` **`hasMultipleAnswers()`**
Descripción: Consulta si una pregunta es de respuesta múltiple o no.
Return: `true` si la pregunta tiene más de una opción correcta, `false` en caso contrario.

II.2.10. Clase `ScheduledTest`

Clase que representa una programación de cuestionario.

Funciones:

`public function` **`__construct()`**
Descripción: Constructor de la clase `ScheduledTest`.
Parámetros: Admite dos opciones: ◦ 1 parámetro: será el número de ID de la programación del cuestionario. Para instanciar programaciones ya existentes. ◦ 6 parámetros: número de ID del cuestionario, fecha de inicio, fecha de fin, valor booleano para indicar si es una encuesta o no, valor booleano para indicar si su realización es obligatoria o no y valor numérico para indicar el modo. Para instanciar programaciones aún no creadas.

`public function` **`getID()`**
Descripción: Devuelve el valor de la variable `$_id` del objeto.
Return: El valor de la variable `$_id`.

`public function` **`setID()`**
Descripción: Establece el valor de la variable `$_id` del objeto.
Parámetros: `$id`: ID de la programación de cuestionario.

`public function` **`getTest()`**
Descripción: Devuelve el valor de la variable `$_test` del objeto, que almacena el número de ID del cuestionario.
Return: El valor de la variable `$_test`.

`public function` **`setTest()`**
Descripción: Establece el valor de la variable `$_test` del objeto, que almacena el número de ID del cuestionario.
Parámetros: `$test`: Número de ID del cuestionario.

`public function` **`getDateStart()`**
Descripción: Devuelve el valor de la variable `$_date_start` del objeto, que almacena la fecha de inicio de la programación de cuestionario.
Return: El valor de la variable `$_date_start`.

public function **setDateStart()**
Descripción: Establece el valor de la variable `$_date_start` del objeto, que almacena la fecha de inicio de la programación de cuestionario.
Parámetros: `$date_start`: Cadena de caracteres que representa la fecha de inicio.

public function **getDateEnd()**
Descripción: Devuelve el valor de la variable `$_date_end` del objeto, que almacena la fecha de finalización de la programación de cuestionario.
Return: El valor de la variable `$_date_end`.

public function **setDateEnd()**
Descripción: Establece el valor de la variable `$_date_end` del objeto, que almacena la fecha de finalización de la programación de cuestionario.
Parámetros: `$date_end`: Cadena de caracteres que representa la fecha de finalización.

public function **getSurvey()**
Descripción: Devuelve el valor de la variable `$_survey` del objeto, que indica si dicha programación de cuestionario es una encuesta o no.
Return: El valor de la variable `$_survey`.

public function **setSurvey()**
Descripción: Establece el valor de la variable `$_survey` del objeto, que indica si dicha programación de cuestionario es una encuesta o no.
Parámetros: `$survey`: Valor booleano que indica si dicha programación de cuestionario es una encuesta o no. Por defecto toma el valor `true`.

`public function` **`getMandatory()`**

Descripción:

Devuelve el valor de la variable `$_mandatory` del objeto, que indica si la realización de dicha programación de cuestionario es obligatoria o no.

Return:

El valor de la variable `$_mandatory`.

`public function` **`setMandatory()`**

Descripción:

Establece el valor de la variable `$_mandatory` del objeto, que indica si la realización de dicha programación de cuestionario es obligatoria o no.

Parámetros:

`$mandatory`: Valor booleano que indica si la realización de dicha programación de cuestionario es obligatoria o no. Por defecto toma el valor `true`.

`public function` **`isMandatory()`**

Descripción:

Consulta si la realización de la programación del cuestionario es obligatoria o no.

Return:

Un valor booleano que indica la obligatoriedad.

`public function` **`getResultMode()`**

Descripción:

Devuelve el valor de la variable `$_result_mode` del objeto, que indica el tipo de salida que obtendrán los alumnos tras realizar el cuestionario.

Return:

El valor de la variable `$_result_mode` del objeto.

`public function` **`setResultMode()`**

Descripción:

Establece el valor de la variable `$mode` del objeto que indica el tipo de salida que obtendrán los alumnos tras realizar el cuestionario.

Parámetros:

`$mode`: Número que indica el tipo de salida que obtendrán los alumnos tras realizar el cuestionario. Hay cuatro modos:

- 3: Cuando el test ha finalizado, muestra la nota al alumno
- 2: Cuando el test ha finalizado, muestra nota y detalles al alumno
- 0: Tras contestar, muestra nota y detalles al alumno.
- 1: Tras contestar, muestra únicamente la nota al alumno

En el caso de que el parámetro tenga otro valor, se establecerá el modo por defecto.[14]

`public function` **`setScheduledTestByID()`**

Descripción:

Dado el ID de la programación de cuestionario, instancia las variables del objeto con la información correspondiente a dicha programación en la base de datos.

Parámetros:

`$id`: Número de ID de la programación de cuestionario.

Return:

`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`schedule()`**

Descripción:

Programa el cuestionario almacenándolo en la base de datos.

Return:

`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`getQuestions()`**

Descripción:

Obtiene las preguntas del cuestionario al que está asociado esta programación.

Return:

Array con los números de ID de las preguntas.

14 Indicado con la opción `RESULTMODE` del fichero de configuración *config.php*.

`public function` **`isLive()`**
Descripción: Consulta si la programación del cuestionario es en directo
Return: `true` si se trata de una programación en directo, `false` en caso contrario.

`public function` **`isCurrentlyActive()`**
Descripción: Consulta si la programación del cuestionario está en curso actualmente.
Return: `true` si está en curso, `false` en caso contrario.

`public function` **`update()`**
Descripción: Modifica en la base de datos los parámetros de la programación de cuestionario
Return: `true` en caso de éxito, `false` en caso contrario.

`public function` **`finished()`**
Descripción: Consulta si la programación ha terminado.
Return: `true` en caso de éxito, `false` en caso contrario.

`public function` **`remove()`**
Descripción: Elimina la programación de cuestionario si aún no ha comenzado.
Return: `true` en caso de éxito, `false` en caso contrario.

`public function` **`end()`**

Descripción:
Finaliza en el instante actual la programación del cuestionario.

Return:
`true` en caso de éxito, `false` en caso contrario.

`public function` **`getStudents()`**

Descripción:
Obtiene una lista con los alumnos que han realizado la programación del cuestionario.

Return:
Array con los ID de los alumnos. Devuelve `false` en caso de error.

`public function` **`getDoneDate()`**

Descripción:
Obtiene la fecha en la que un usuario realizó una programación del cuestionario.

Parámetros:
`$user_id`: Número de ID del usuario.

Return:
Cadena de caracteres que representa la fecha. Devuelve `false` si la programación aún no ha terminado o en caso de error.

`public function` **`getDoneDate()`**

Descripción:
Obtiene la nota obtenida por un usuario tras realizar la programación de cuestionario.

Parámetros:
`$user_id`: Número de ID del usuario.

Return:
Número con la nota obtenida. Devuelve `false` si el usuario no ha participado en dicha programación la programación siendo opcional o en caso de error.

`public function` **`printMarkDetails()`**

Descripción:
Imprime código HTML con los los detalles de la nota de un usuario en la programación de cuestionario.

Parámetros:
`$user_id`: Número de ID del usuario.

`public function` **`mediumMark()`**
Descripción: Obtiene la nota media de la programación en función de la obligatoriedad.
Parámetros: `$type`: Indica el tipo de obligatoriedad. Puede tomar el valor `all`, que incluye cuestionarios opcionales y obligatorios, o el valor `optional`, con sólo los opcionales. El valor por defecto es `all`.
Return: Número con la nota media.

`public function` **`getAnswersDetails()`**
Descripción: Obtiene el número de alumnos que han marcado como respuesta cada opción de un cuestionario programado.
Return: Array con el número de alumnos para cada opción de cada pregunta.

`public function` **`getAnswersDetails()`**
Descripción: Obtiene el número de alumnos que han seleccionado una opción como respuesta.
Parámetros: `$option_id`: Número con el ID de la opción.
Return: Número de alumnos. Devuelve `false` en caso de error.

II.2.11. Clase `Security`

Clase con funciones relativas a seguridad y control de acceso de la aplicación.

Funciones:

`public static function` **`sanitizeArray()`**
Descripción: Escapa los caracteres especiales en un array para ser usado en consultas a la base de datos.
Parámetros: `$array`: Array a escapar.
Return: El array escapado.

`public static function` **`ensureSession()`**
Descripción: Se asegura de que la sesión está iniciada y redirige al usuario al inicio de sesión en caso contrario.

II.2.12. Clase `Stats`

Clase que que facilita funciones para obtener medidas estadísticas.

Funciones:

`public static function` **`median()`**
Descripción: Calcula la mediana dado un array de datos.
Parámetros: `$data`: Array con datos del que se desea obtener su mediana. `$sort_flags`: Establece el modo de ordenación del array. Puede tomar los mismos valores que la función `sort()` de PHP.[15] Por defecto toma el valor `SORT_REGULAR`, que compara los elementos sin cambiar los tipos.
Return: El valor de la mediana.

II.2.13. Clase `Test`

Clase que representa a un cuestionario de una clase.

Funciones:

`public function` **`__construct()`**
Descripción: Constructor de la clase `Test`.
Parámetros: Admite dos opciones: ◦ 1 parámetro: será el número de ID del cuestionario. Para instanciar cuestionarios ya existentes. ◦ 3 parámetros: número de ID de la clase, título del cuestionario y valor booleano que indica si el cuestionario se ha generado aleatoriamente. Para instanciar cuestionarios aún no creados.

15 Sobre la función `sort()`, véase *http://php.net/manual/es/function.sort.php*

`public function` **`getID()`**

Descripción:
Devuelve el valor de la variable `$_id` del objeto.

Return:
El valor de la variable `$_id`.

`public function` **`setID()`**

Descripción:
Establece el valor de la variable `$_id` del objeto.

Parámetros:
`$id`: Número de ID del cuestionario.

`public function` **`getClassroom()`**

Descripción:
Devuelve el valor de la variable `$_classroom` del objeto, que indica el ID de la clase a la que pertenece el cuestionario.

Return:
El valor de la variable `$_classroom`.

`public function` **`setClassroom()`**

Descripción:
Establece el valor de la variable `$_classroom` del objeto, que indica el ID de la clase a la que pertenece el cuestionario.

Parámetros:
`$classroom_id`: ID de la clase a la que pertenece la pregunta.

`public function` **`getTitle()`**

Descripción:
Devuelve el valor de la variable `$_title` del objeto, que contiene el título del cuestionario.

Return:
El valor de la variable `$_title`.

`public function` **`setTitle()`**
Descripción: Establece el valor de la variable `$_title` del objeto, que contiene el título del cuestionario.
Parámetros: `$title`: ID de la clase a la que pertenece el cuestionario.

`public function` **`isRandom()`**
Descripción: Consulta si el cuestionario se ha generado aleatoriamente.
Return: Un valor booleano que indica si se ha generado de forma aleatoria.

`public function` **`setTestByID()`**
Descripción: Dado el ID del cuestionario, instancia las variables del objeto con la información correspondiente a dicho cuestionario en la base de datos.
Parámetros: `$id`: Número de ID del cuestionario.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`create()`**
Descripción: Guarda el cuestionario en la base de datos.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`update()`**
Descripción: Actualiza la información del cuestionario.
Parámetros: `$title`: Cadena de caracteres con el título del cuestionario.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`delete()`**

Descripción:
Borra el cuestionario de la base de datos, así como sus apariciones en otras tablas.

Return:
`true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`isScheduledNow()`**

Descripción:
Comprueba si el cuestionario está programado y si está en curso en este momento.

Return:
`true` si está programado en este instante, `false` en caso contrario.

`public function` **`currentSchedules()`**

Descripción:
Obtiene cada una de las programaciones del cuestionario actualmente en curso.

Return:
Array con los números de ID de las programaciones de cuestionario.

`public function` **`isScheduled()`**

Descripción:
Consulta si un cuestionario está programado para un futuro.

Return:
`true` si está programado, `false` en caso contrario.

`public function` **`nextSchedules()`**

Descripción:
Obtiene cada una de las futuras programaciones del cuestionario.

Return:
Array con los números de ID de las futuras programaciones de cuestionario.

`public function` **`nextStart()`**
Descripción: Obtiene la fecha de inicio de la próxima programación del cuestionario.
Return: Cadena de caracteres con una representación de la fecha de inicio.

`public function` **`setQuestions()`**
Descripción: Asigna al cuestionario un conjunto de preguntas.
Parámetros: `$questions`: Array con los números de ID de las preguntas.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`unsetQuestions()`**
Descripción: Elimina las preguntas asociadas al cuestionario.
Return: `true` si se se ha realizado con éxito, `false` en caso contrario.

`public function` **`getQuestions()`**
Descripción: Obtiene las preguntas asociadas al cuestionario.
Return: Array con los números de ID de las preguntas asociadas.

`public function` **`numQuestions()`**
Descripción: Obtiene el número de preguntas asociadas al cuestionario.
Return: Número de preguntas.

`public function` **`isQuestion()`**

Descripción:

Consulta si una pregunta está asociada al cuestionario.

Parámetros:

`$question`: Número de ID de la pregunta.

Return:

`true` si la pregunta está asociada al cuestionario, `false` en caso contrario.

`public function` **`saveAnswer()`**

Descripción:

Guarda en la base de datos las opciones respondida por un alumno a la pregunta.

Parámetros:

`$user`: Número de ID del alumno.
`$schedule`: Número de ID de la programación del cuestionario.
`$question`: Número de ID de la pregunta.
`$options`: Array con los números de ID de las opciones de la pregunta respondidas.

Return:

`true` si se ha realizado con éxito, `false` en caso contrario.

`public function` **`answered()`**

Descripción:

Consulta si una pregunta ha sido respondida por un usuario en una programación del cuestionario concreta.

Parámetros:

`$user`: Número de ID del alumno.
`$schedule`: Número de ID de la programación del cuestionario.
`$question`: Número de ID de la pregunta.

Return:

`true` si la pregunta ha sido respondida, `false` en caso contrario.

II.2.14. Clase `User`

Clase que representa a un usuario de la aplicación y permite su registro, el manejo de la sesión o la consulta y modificación su su información directa.

Funciones:

`public function __construct()`
Descripción: Constructor de la clase.
Parámetros: `$id_or_data`: Puede ser un número, que representa el ID del usuario, o un array de datos (email, password, name, surname, dni).

`public function getID()`
Descripción: Devuelve el valor de la variable `$_id` del objeto.
Return: El valor de la variable `$_id`.

`public function setID()`
Descripción: Establece el valor de la variable `$_id` del objeto.
Parámetros: `$id`: ID del usuario.

`public function getData()`
Descripción: Obtiene los datos del objeto (variable `$_data`).
Return: El valor de la variable `$_data`.

`public function` **`setData()`**

Descripción:

Establece el valor de la variable `$_data` del objeto.

Parámetros:

`$data`: Un array asociativo con datos del usuario (cuyas claves tienen los nombres 'name', 'surname', 'dni', 'email', 'password').

`public function` **`setUserByID()`**

Descripción:

Dado un ID, almacena en la variable `$_data` la información almacenada sobre dicho usuario en la base de datos.

Parámetros:

`$user_id`: Número de ID del usuario.

`public function` **`register()`**

Descripción:

Inserta en la base de datos un usuario con los datos del objeto.

Return:

`true` si el registro ha sido efectuado, `false` en caso contrario.

`public function` **`login()`**

Descripción:

Inicia sesión del usuario.

`public static function` **`logout()`**

Descripción:

Cierra la sesión del usuario.

`public static function` **`addFlash()`**

Descripción:

Asigna a la sesión un mensaje flash para mostrarlo al usuario.

Parámetros:

`$text`: String con el mensaje a mostrar.

`$type`: String con tipo del mensaje (a elegir entre `info`, `success`, `warning`, o `danger`). El valor por defecto es `info`.

`public static function` **`flash()`**

Descripción:
Imprime los mensajes flash asignados a la sesión y después los elimina.

`public function` **`pass()`**

Descripción:
Comprueba si la contraseña proporcionada es la del usuario.

Parámetros:
`$password`: String con la contraseña.

Return:
`true` si la contraseña es correcta, `false` en caso contrario.

`public function` **`update()`**

Descripción:
Actualiza datos del usuario tanto en la base de datos (si existe) como en la instancia.

Parámetros:
`$field`: String con el campo a actualizar.
`$value`: String con el valor para dicho campo.

Return:
`true` si se ha podido actualizar, `false` en caso contrario.

`public static function` **`getIDByEmail()`**

Descripción:
Devuelve el ID de usuario a partir de un email.

Parámetros:
`$email`: String con el email.

Return:
Un número con el ID del usuario asociado al email, si este existe;, `false` en caso contrario.

`public static function` **`getIDByDNI()`**
Descripción: Devuelve el ID de usuario a partir de un DNI.
Parámetros: `$dni`: String con el DNI.
Return: Un número con el ID del usuario asociado al DNI, si este existe;, `false` en caso contrario.

`public static function` **`existsEmail()`**
Descripción: Comprueba si existe o no un email de usuario.
Parámetros: `$email`: String con el email.
Return: `true` si se existe, `false` en caso contrario.

`public static function` **`existsDNI()`**
Descripción: Comprueba si existe o no un DNI de usuario.
Parámetros: `$dni`: String con el DNI.
Return: `true` si se existe, `false` en caso contrario.

`public function` **`delete()`**
Descripción: Borra un usuario.
Return: `true` si se se ha podido borrar, `false` en caso contrario.

`public function` **`isActive()`**
Descripción: Consulta si un usuario está activo o ha borrado su cuenta.
Return: `true` si se el usuario está activo, `false` en caso contrario (ha borrado su cuenta).

`public function` **`getClassrooms()`**
Descripción: Devuelve una lista de ID de las clases en las que el usuario es profesor.
Return: Array con los números de ID de las clases.

`public function` **`getClassrooms()`**
Descripción: Devuelve una lista de ID de las clases en las que el usuario es alumno.
Return: Array con los números de ID de las clases.

`public function` **`hasRemainingRegisters()`**
Descripción: Devuelve si el usuario tiene registros de asistencia pendientes de hacer.
Return: `true` si se el usuario tiene registros de asistencia pendientes, `false` en caso contrario.

`public function` **`getRemainingRegisters()`**
Descripción: Devuelve los ID de los registros de asistencia pendientes de hacer.
Return: Array con los ID de los registros de asistencia pendientes de hacer.

`public function` **`hasRemainingLiveTests()`**
Descripción: Consulta si el usuario tiene cuestionarios en directo pendientes de hacer
Return: `true` si se el usuario tiene cuestionarios en directo pendientes de hacer, `false` en caso contrario.

`public function` **`getRemainingLiveTests()`**

Descripción:

Devuelve los ID de los cuestionarios en directo pendientes de hacer.

Return:

Array con los ID de los cuestionarios en directo pendientes de hacer.

`public function` **`hasRemainingTests()`**

Descripción:

Consulta si el usuario tiene cuestionarios programados (no en directo) pendientes de hacer.

Return:

`true` si se el usuario tiene cuestionarios programados pendientes de hacer, `false` en caso contrario.

`public function` **`getRemainingTestsOfClass()`**

Descripción:

Devuelve los cuestionarios programados (no en directo) pendientes de usuario en una clase concreta.

Parámetros:

`$classroom_id`: Número de ID de la clase.

Return:

Array con los ID de los cuestionarios en programados pendientes de hacer en dicha clase.

`public function` **`hasRemainingTestsOfClass()`**

Descripción:

Consulta si el usuario tiene cuestionarios programados (no en directo) pendientes de una clase concreta.

Parámetros:

`$classroom_id`: Número de ID de la clase.

Return:

`true` si se el usuario tiene cuestionarios programados pendientes de hacer en dicha clase, `false` en caso contrario.

`public function` **`canDoTest()`**

Descripción:

Consulta si un usuario puede hacer un cuestionario programado (no en directo). Es decir, si tiene acceso a un cuestionario actualmente en curso.

Parámetros:

`$schedule_id`: Número de ID de la programación del cuestionario.

Return:

`true` si se el usuario puede acceder al cuestionario, `false` en caso contrario.

`public function` **`hasDoneScheduledTest()`**

Descripción:

Consulta si un usuario ha realizado una programación de un cuestionario.

Parámetros:

`$schedule_id`: Número de ID de la programación del cuestionario.

Return:

`true` si ha realizado dicha programación, `false` en caso contrario.

`public function` **`getDoneTestsOfClass()`**

Descripción:

Devuelve una lista de las programaciones de cuestionario finalizadas de una clase que el usuario ha realizado.

Parámetros:

`$classroom_id`: Número de ID de la clase.

`$type`: String con tipo de obligatoriedad de la programación del cuestionario (a elegir entre `mandatory`, `optional` o `all`). El valor por defecto es `all`.

Return:

Array con los ID de las programaciones de cuestionarios finalizadas en dicha clase.

`public function` **`numDoneTestsOfClass()`**

Descripción:

Devuelve el número de programaciones de cuestionario realizadas por el usuario en una clase.

Parámetros:

`$classroom_id`: Número de ID de la clase.
`$type`: String con tipo de obligatoriedad de la programación del cuestionario (a elegir entre `mandatory`, `optional` o `all`). El valor por defecto es `all`.

Return:

Número de programaciones realizadas.

`public function` **`finishedScheduledTests()`**

Descripción:

Devuelve una lista con ID de las programaciones de cuestionario finalizadas a las que ha tenido acceso el usuario en una clase (no tiene por qué haberlas hecho).

No se tiene cuenta si el alumno se ha desmatriculado de la clase, para que su media se siga calculando en los cuestionarios obligatorios.

Parámetros:

`$classroom_id`: Número de ID de la clase.
`$type`: String con tipo de obligatoriedad de la programación del cuestionario (a elegir entre `mandatory`, `optional` o `all`). El valor por defecto es `all`.

Return:

Array con los ID de las programaciones de cuestionario.

`public function` **`answered()`**

Descripción:

Consulta si un alumno ha respondido con una opción a una pregunta de una programación de cuestionario.

Parámetros:

`$scheduled_test_id`: Número de ID de la programación.
`$question_id`: Número de ID de la pregunta.
`$option_id`: Número de ID de la opción.

Return:

`true` si ha realizado respondido, `false` en caso contrario.

`public function` **`mediumMark()`**
Descripción: Devuelve la nota media del usuario en una clase.
Parámetros: `$classroom_id`: Número de ID de la clase. `$type`: String con tipo de obligatoriedad de las programaciones de cuestionario de la clase a tener en cuenta (a elegir entre `mandatory`, `optional` o `all`). El valor por defecto es `all`.
Return: Número con la nota media del usuario.

`public function` **`medianMark()`**
Descripción: Devuelve la nota mediana del usuario en una clase.
Parámetros: `$classroom_id`: Número de ID de la clase. `$type`: String con tipo de obligatoriedad de las programaciones de cuestionario de la clase a tener en cuenta (a elegir entre `mandatory`, `optional` o `all`). El valor por defecto es `all`.
Return: Número con la nota mediana del usuario.

II.3. Resto de ficheros

A continuación se describe brevemente cada uno de los ficheros en el orden en el que aparecen en el árbol de ficheros mostrado anteriormente. En el caso de los ficheros del directorio includes, únicamente se describen los desarrollados exclusivamente para este trabajo, obviando los ficheros correspondientes a las librerías jQuery y Bootstrap.

/tfg/config.php

Fichero de configuración. En él se definen constantes que se emplean en el resto de la aplicación.

Parámetros de configuración en *config.php*	
DBHOST	Ruta del servidor de la base de datos.
DBUSER	Nombre de usuario de la base de datos.
DBPASS	Contraseña de la base de datos.
DBNAME	Nombre de la base de datos.
TITLE	Título del sitio.
MENU	Menú superior del sitio.
COLOPHON	Pie de página.
PASSWORDLENGHT	Tamaño mínimo de las contraseñas.
NUMOPTIONS	Número de opciones de una pregunta.
DATEFORMAT	Formato de fecha.
DBDATEFORMAT	Formato de fecha de trabajo de la base de datos.
USUALDATEFORMAT	Formato de fecha común.
MULTIPLIER	Factor para calcular la resta en opciones erróneas respondidas en un test en relación a un opción correcta.
NEGATIVE_MARKS	Habilita o inhabilita las notas negativas.
PRECISION	Número de decimales. Si hay más decimales se redondearán.
RESULTMODE	Tipo de salida de los cuestionarios para el alumno por defecto.
SECONDS	Número de segundos por defecto en un test en directo.
WAITINGTIME	Número de segundos de espera en un test en directo.
DEVELOPERMODE	Habilita o inhabilita opciones de desarrollo (aviso de errores, etc.).

/tfg/index.php

Página principal del sitio. Muestra, principalmente, aquellas clases en las que el usuario se encuentra inscrito como alumno y aquellas en las que es profesor.

/tfg/init.php

Script de inicialización. Realiza importaciones de código y clases, controla sesiones, inicializa la interfaz, etc.

/tfg/load-notifications.php

Script que muestra las notificaciones.

/tfg/logout.php

Script que finaliza la sesión actualmente activa.

/tfg/classroom/delete.php

Muestra el formulario de eliminación de la clase, lo valida y llama a las funciones pertinentes para efectuar el borrado.

/tfg/classroom/join.php

Muestra el formulario para unirse a una clase, lo valida y llama a las funciones pertinentes para efectuar la inscripción.

/tfg/classroom/new.php

Muestra el formulario para crear una clase nueva, lo valida y llama a las funciones pertinentes para efectuar la creación.

/tfg/classroom/student.php

Muestra la información de una clase disponible para el alumno.

/tfg/classroom/students.php

Muestra al profesor una lista de alumnos de una clase.

/tfg/classroom/teacher.php

Muestra la información de una clase disponible para el profesor.

/tfg/classroom/unenroll.php

Muestra el formulario para abandonar una clase, lo valida y llama a las funciones pertinentes para eliminar la inscripción.

/tfg/classroom/attendance/confirm.php

Confirma la asistencia de un alumno en un registro de asistencia.

/tfg/classroom/attendance/new.php

Muestra al profesor el formulario para crear un registro de asistencia, valida el formulario y llama a las funciones pertinentes para efectuar la creación.

/tfg/classroom/attendance/student.php

Muestra al alumno el formulario para crear confirmar su asistencia y lo envía a *confirm.php*.

/tfg/classroom/attendance/teacher_load.php

Imprime el número de alumnos que han confirmado su asistencia para ser mostrado en *teacher.php*.

/tfg/classroom/attendance/teacher.php

Muestra al profesor una pantalla en la que aparecerá el número de personas que han confirmado su asistencia generado en *teacher_load.php*.

/tfg/classroom/live/end-waiting.php

Finaliza el periodo de espera de un cuestionario en directo.

/tfg/classroom/live/next.php

Hace avanzar un cuestionario en directo.

/tfg/classroom/live/random.php

Muestra el formulario que permite al profesor lanzar un cuestionario en directo generado de forma aleatoria, lo valida y llama a las funciones pertinentes para crear en primer lugar un cuestionario aleatorio y posteriormente lanzarlo.

/tfg/classroom/live/release.php

Muestra el formulario que permite al profesor establecer los parámetros para lanzar un cuestionario en directo y los envía al script *start.php*.

/tfg/classroom/live/saveanswer.php

Almacena la respuesta de un alumno a una pregunta de un cuestionario en directo.

/tfg/classroom/live/start.php

Lanza un cuestionario en directo.

/tfg/classroom/live/student.php

Muestra al alumno el formulario para responder a una pregunta en directo y envía la respuesta pasado el tiempo estipulado por el profesor para dicha pregunta.

/tfg/classroom/live/teacher_load.php

Imprime la pregunta actual para ser mostrada en *teacher.php*.

/tfg/classroom/live/teacher-waiting.php

Muestra al profesor una cuenta atrás de espera mientras inicia el formulario, y pasado el tiempo llama al script *end-waiting.php* para finalizar la espera y comenzar a mostrar las preguntas.

/tfg/classroom/live/teacher.php

Muestra al profesor una pantalla en la que aparecerán las preguntas cargadas en *teacher_load.php*, al que llama después de que finalice el tiempo para la pregunta actual y haya llamado a la función *next.php* para hacer avanzar el cuestionario.

/tfg/classroom/question/delete.php

Muestra el formulario que permite al profesor eliminar una pregunta, lo valida y llama a las funciones pertinentes para efectuar el borrado.

/tfg/classroom/question/index.php

Muestra al profesor una pregunta actual, el formulario que permite editarla, lo valida y llama a las funciones pertinentes para efectuar las modificaciones.

/tfg/classroom/question/new.php

Muestra el formulario que permite al profesor crear una pregunta, lo valida y llama a las funciones pertinentes para efectuar la creación.

/tfg/classroom/stats/details.php

Muestra a al profesor los resultados detallados de un alumno en un cuestionario concreto.

/tfg/classroom/stats/exportattendance.php

Permite la descarga de un fichero en formato CSV de la asistencia en una clase.

/tfg/classroom/stats/exportmarks.php

Permite la descarga de un fichero en formato CSV de las calificaciones en una clase.

/tfg/classroom/stats/teacher.php

Muestra a al profesor la información estadística de una clase.

/tfg/classroom/stats/register.php

Muestra a al profesor los resultados de asistencia en un registro concreto de una clase.

/tfg/classroom/stats/student-result.php

Muestra a al alumno sus resultados tras la realización de un cuestionario.

/tfg/classroom/stats/student.php

Muestra a al profesor las estadísticas obtenidas por un alumno en una clase.

/tfg/classroom/stats/test.php

Muestra a al profesor las estadísticas obtenidas en un cuestionario concreto de una clase.

/tfg/classroom/test/correct.php

Almacena las respuestas de un alumno a un cuestionario programado.

/tfg/classroom/test/delete.php

Muestra el formulario para eliminar un cuestionario, lo valida y llama a las funciones pertinentes para efectuar el borrado.

/tfg/classroom/test/edit.php

Muestra el formulario para editar un cuestionario, lo valida y llama a las funciones pertinentes para actualizar el cuestionario.

/tfg/classroom/test/new.php

Muestra el formulario para crear un cuestionario, lo valida y llama a las funciones pertinentes para guardarlo.

/tfg/classroom/test/schedule-edit.php

Muestra el formulario para editar los parámetros de una programación de cuestionario, lo valida y llama a las funciones pertinentes para efectuar las modificaciones.

/tfg/classroom/test/schedule-end.php

Finaliza una programación de cuestionario activa.

/tfg/classroom/test/schedule-remove.php

Elimina una programación de cuestionario futura.

/tfg/classroom/test/schedule.php

Muestra el formulario para programar un cuestionario, lo valida y llama a las funciones pertinentes para almacenar la programación.

/tfg/classroom/test/student.php

Muestra el formulario que permite al alumno realizar un cuestionario programado, lo valida y envía sus respuestas al script *correct.php*.

/tfg/classroom/test/teacher.php

Muestra a al profesor la información y opciones de un cuestionario.

/tfg/includes/css/custom.css

Hoja de estilos personalizados.

/tfg/includes/js/load-notifications.js

Actualiza las notificaciones.

/tfg/login/index.php

Muestra el formulario de inicio de sesión, lo valida y llama a las funciones pertinentes para iniciar la sesión.

/tfg/profile/changemail.php

Muestra el formulario de cambio de correo electrónico, lo valida y llama a las funciones pertinentes para efectuar la modificación.

/tfg/profile/changepassword.php

Muestra el formulario de cambio de contraseña, lo valida y llama a las funciones pertinentes para efectuar la modificación.

/tfg/profile/delete.php

Muestra el formulario de eliminación de la cuenta de usuario, lo valida y llama a las funciones pertinentes para efectuar el borrado.

/tfg/profile/index.php

Muestra el perfil de usuario con su información.

/tfg/profile/update.php

Muestra el formulario para modificar la información del perfil, lo valida y llama a las funciones pertinentes para efectuar los cambios.

/tfg/signin/index.php

Muestra el formulario de registro, lo valida y llama a las funciones pertinentes para efectuar el registro del nuevo usuario.