Proyecto Fin de Carrera

Diseño y Desarrollo de una Aplicación de Preguntas Tipo Test On-line

Alberto Vila Tena

Universidad Carlos III de Madrid • Escuela Politécnica Superior

Ingeniería en Informática

Índice General

Índice de Figuras

Índice de Tablas

1. INTRODUCCIÓN

ESTADO DEL ARTE

La llegada de la Web 2.0 ha supuesto un modelo completamente nuevo tanto en el uso como en el diseño y desarrollo de aplicaciones web y no hay más que mirar a nuestro alrededor para darnos cuenta del gran éxito que ha supuesto.

Nombres como *Facebook, Wikipedia, Twitter, Tuenti (en España), blogspot,* etc. resultan hoy extremadamente conocidos y han sido visitados por el usuario medio de internet, y todos ellos tienen un punto en común en la socialización. Todas las webs mencionadas antes son actualmente grandes comunidades de usuarios donde sus distintos miembros pueden publicar información libremente y compartirla al instante con los otros participantes o con los visitantes ocasionales de la web.

El gran éxito de estas aplicaciones deja un claro mensaje: El usuario no se conforma ya con leer los contenidos de una web, sino que desea también ser protagonista de la misma y poder contrastar sus conocimientos, gustos y opiniones con otros como él. Este nuevo punto de vista respecto a la web altera por tanto los que eran los cimientos del diseño y el desarrollo web: El diseñador de una web no debe establecer ya los contenidos de la misma, sino los medios necesarios para que la publicación y difusión de contenidos por parte de los usuarios sea lo más fácil y eficiente posible.

Aunque las distintas webs antes mencionadas tienen un carácter muy general en cuanto a sus contenidos, el auge de la Web 2.0 ha propiciado la aparición de sitios de estas características aplicados a temáticas muy concretas, ya sea cine (*filmaffinity)*, música *(last.fm)* o a aspectos como la educación y la docencia. Ligados a esto se encuentran aplicaciones como *Moodle*, que hoy en día es usado por numerosas universidades en todo el mundo.

Las aportaciones que la Web 2.0 puede traer a la docencia son muy significativas pues permite establecer un marco de interacción entre alumnos y profesores fuera de las aulas en el cual la información fluye a gran velocidad. En aplicaciones como *Moodle*, entre otras cosas,un profesor puede controlar a aquellos alumnos que están suscritos a su asignatura, facilitarles el acceso a los materiales de la misma, interactuar con ellos y resolver sus dudas; y los propios alumnos pueden también consultar al profesor o ayudarse mutuamente mediante los foros o chats. En conclusión, puede decirse que muchos de los patrones de comportamiento que se dan en el mundo de la docencia se trasladan a la red con este tipo de aplicaciones.

OBJETIVOS

El objetivo de este Proyecto Fin de Carrera es el de realizar el estudio, el diseño, el desarrollo y el despliegue de una aplicación web basada en los principios de la Web 2.0 aplicada a la docencia y orientada al estudio de materias concretas.

El método que se va a utilizar para ello es la publicación y el intercambio de preguntas tipo test entre los distintos usuarios del sistema, que podrán crear sus propias preguntas y responder a las preguntas de los otros participantes pudiendo evaluar el valor docente de las mismas.

Los docentes también tendrán su espacio en la aplicación, estableciendo la temática de las preguntas que sus alumnos realizarán, y pudiendo consultar y analizar fácilmente la actividad de sus alumnos en la aplicación a través de ella.

ENFOQUE

Al consistir en una aplicación web completamente funcional la finalidad del proyecto, puede afirmarse que el enfoque del mismo va a ser eminentemente práctico.

Asimismo, la naturaleza del problema da claras pistas de como puede ser enfocado, ya que al enmarcarse como Web 2.0, la aplicación a desarrollar poseerá una serie de características concretas que se resumen bien en la siguiente definición:

"Todas aquellas utilidades y servicios de Internet que se sustentan en una base de datos, la cual puede ser modificada por los usuarios del servicio, ya sea en su contenido (añadiendo, cambiando o borrando información o asociando datos a la información existente), pues bien en la forma de presentarlos, o en contenido y forma simultáneamente."

"(Wikipedia - "Web 2.0" n.d.)**."**- (Ribes, 2007)

Puede comprobarse como esta definición encaja con un patrón de arquitectura Modelo Vista Controlador (MVC), en el cual, en efecto, se basan todas las aplicaciones Web 2.0 y el increíble auge de estas últimas ha dado lugar a la aparición de metodologías y marcos de desarrollo específicos para la resolución de problemas que se basen en el patrón MVC de una forma eficiente.

Por tanto, este proyecto se servirá de alguna de estas metodologías y marcos de desarrollo creados recientemente; en concreto de una metodología dentro del paradigma de *Desarrollo Web Ágil* y de *Ruby on Rails* como *framework* para la implementación del proyecto.

ESTRUCTURA DEL DOCUMENTO

El documento estará dividido en nueve capítulos, cada uno correspondiente a una fase distinta en el desarrollo del proyecto

En la **Introducción** se hablará del estado del arte concerniente al proyecto y se dará información a grandes rasgos sobre como va a enfocarse el problema y sobre distintos detalles sobre el documento

El Capítulo 2, **Base tecnológica del proyecto**, dará detalles sobre los aspectos técnicos más relevantes del proyecto, como el patrón arquitectónico a utilizar, la metodología de desarrollo que se va a emplear y el *framework* sobre el que se implementará la aplicación.

El Capítulo 3, **Funcionalidades de la aplicación**, contendrá los requisitos especificados por el cliente para la aplicación contenidos en distintos escenarios y casos de uso, con el fin de aclarar lo máximo posible los detalles relativos a lo que será su funcionamiento.

El cuarto capítulo, **Diseño Inicial**, contendrá distintos modelos de alto nivel que darán una idea de cómo estará organizada la información dentro de la aplicación tanto como del esqueleto de la misma.

El capítulo 5, relativo a la **Implementación**, contendrá aquellos aspectos más destacables que tengan lugar durante la fase de implementación

El capítulo 6, **Pruebas Realizadas**, contendrá los resultados de las distintas pruebas unitarias que se realizarán sobre la aplicación una vez que esta esté terminada para verificar su correcto funcionamiento.

El capítulo 7, **Conclusiones y Trabajos Futuros** dará una opinión crítica sobre el proyecto realizado y su desarrollo además de proponer posibles extensiones futuras al mismo

Y finalmente, el octavo y último capítulo, **Bibliografía**, contendrá aquellas fuentes y recursos externos que se hayan empleado para la realización del proyecto.

NOTACIÓN

Para facilitar la lectura del documento, a lo largo de él se han empleado el mismo estilo eligiéndose los siguientes tipos de letra:

- Texto general: Helvetica, tamaño 10 pt.
- Código: Monaco, tamaño 9 pt.
- ***"Citas"***: Helvetica, tamaño 10 pt., negrita, cursiva, texto entrecomillado
- Direcciones Web: Helvetica, subrayado, color azul, tamaño 10 pt.

2. BASE TECNOLÓGICA DEL PROYECTO

E-LEARNING

E-learning, literalmente aprendizaje electrónico, es una propuesta de formación que sitúa en Internet su principal entorno sirviéndose de las herramientas y servicios que ésta provee.

La principal ventaja de esta propuesta reside en su flexibilidad, pues el estudiante no necesita desplazarse ni verse sometido a ningún horario para formarse de acuerdo a este método, todo sucede como y cuando él desea.

Sin embargo, el hecho de llevar a cabo este proceso de aprendizaje de una forma tan flexible e individual pone de facto grandes limitaciones, ya que en la enseñanza tradicional la interacción entre los distintos agentes que intervienen en ella, ya sea profesor-alumno o entre los propios alumnos entre si, supone un gran incentivo para el aprendizaje ya sea para la resolución de dudas que surjan en torno a los contenidos como para generar nuevas inquietudes en torno a la materia a estudiar. Además, muchas veces es necesario tener en cuenta otros factores como los distintos métodos de aprendizaje que existen entre las personas, métodos de control y de evaluación de los conocimientos que van adquiriendo los usuarios, etc.

Del párrafo anterior se deduce que para construir una plataforma de e-learning pueden tenerse en cuenta un gran número de factores y que por tanto, según el énfasis que se haga en cada uno de ellos, pueden implementarse una infinidad de soluciones. Sin embargo, puede decirse que existen tres elementos que son comunes a toda solución de e-learning:

- La tecnología: Referida a la plataforma software en la que se crearán, almacenarán y gestionarán los contenidos.
- Los contenidos: El material académico utilizado en el proceso de aprendizaje.
- Los servicios: Referidos a las herramientas que la red ofrece para la comunicación e interacción entre usuarios (Publicación o intercambios de mensajes, pizarra electrónica, videoconferencias, etc.).

Por ello, como este proyecto trata sobre el diseño y desarrollo de una aplicación relacionada con el e-learning, a continuación se definirán estos tres elementos para el mismo:

- Tecnología: Una plataforma on-line que será capaz de crear, almacenar y soportar varios usuarios así como los contenidos que estos creen.

- Contenidos: Preguntas tipo test sobre una materia determinada.
- Servicios: Publicación de estas preguntas tipo test y del feedback sobre las mismas en un gestor de contenidos accesible para todos los usuarios de la aplicación.

ARQUITECTURA MODELO-VISTA-CONTROLADOR

La arquitectura Modelo-Vista-Controlador (MVC) es una solución frecuentemente aplicada en las aplicaciones web. Su principal característica es la de aislar la presentación de una aplicación de su lógica interna con el objetivo de facilitar la modificación individual de cualquiera de estas dos partes sin que la otra se vea afectada.

El nombre de esta arquitectura viene dado por sus tres componentes:

El **modelo**, que se refiere a la información que va a manejar la aplicación y que se encontrará almacenada en la mayoría de los casos, y sobre todo en aquellos orientados a desarrollo web, en una base de datos.

La **vista**, referida a aquellos componentes de la aplicación que son visibles para el usuario y con los que éste puede interactuar.

El **controlador**, a modo de enlace entre los dos anteriores, modificando la información que se encuentra en el modelo de acuerdo a la información que recibe del usuario desde la vista, y actualizando a su vez la información en ésta última de tal manera que el usuario pueda continuar la operación que esté llevando a cabo y recibir información sobre su desarrollo.

Cómo puede verse, cada uno de estos componentes maneja elementos de naturaleza distinta y su funcionalidad básica por tanto puede aislarse fácilmente, sin embargo, si van a formar parte de una misma aplicación necesitarán compartir la misma información, a continuación se detalla gráficamente como están asociados modelo, vista y controlador en este sentido.

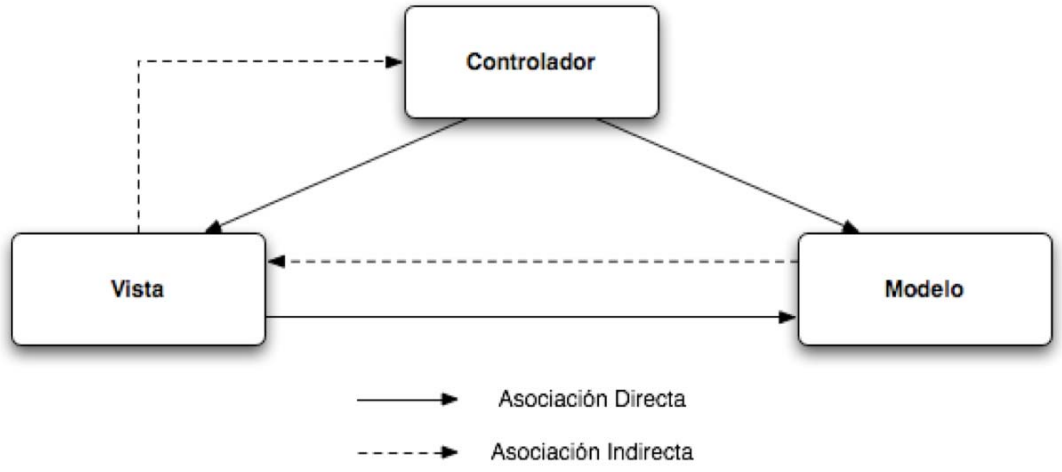


Figura 1: Esquema MVC

La elección de esta arquitectura en el año 2002 por el W3C, la gran expansión de internet en los últimos años y su perfecta adecuación al modelo propuesto por la Web 2.0 han motivado su gran popularidad, a raíz de la cual recientemente han aparecido distintos frameworks que intentan aplicar la arquitectura MVC de una forma clara y eficiente con el objetivo de agilizar el desarrollo de aplicaciones web.

DESARROLLO WEB ÁGIL

Se conoce como Desarrollo Web Ágil a un modelo de desarrollo basado en el paradigma de Desarrollo Ágil de Software orientado a aplicaciones web.

Las metodologías incluidas en este modelo de desarrollo cumplen con los valores especificados en el *Manifiesto Ágil (2001).*

- ***"Los individuos y su interacción, sobre los procesos y las herramientas."***
- ***"El software que funciona, sobre la documentación exhaustiva."***
- ***"La colaboración con el cliente, sobre la negociación contractual."***
- ***"La respuesta al cambio, sobre el seguimiento de un plan."***

  - Agile Manifesto [http://www.agilemanifesto.org/]

Este modelo es de carácter adaptativo y se basa en la aplicación paralela de las fases de planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Su idea es la de proveer metodologías de desarrollo basadas en la interacción con el cliente y flexibles a la inclusión o cambio de requisitos por parte de este último en contraposición a las metodologías tradicionales, más focalizadas en procesos burocráticos y de documentación y menos flexibles.

Por tanto, puede decirse que el Desarrollo Web Ágil sitúa su meta en la satisfacción del cliente con la aplicación, empleando para ello una interacción y un feedback constante por parte del cliente así como un estricto cumplimiento de plazos.

Para conseguir una interacción y un feedback efectivos por parte del cliente, el desarrollo de la aplicación se basará en el desarrollo de distintos incrementos funcionales no muy extensos, los cuales podrán ser utilizados y revisados por el cliente, y modificados posteriormente de acuerdo a la información obtenida. De esta forma se pretende ir contando con una aquiescencia progresiva del cliente con lo que va siendo desarrollado y convertir la aparición de nuevos requisitos en algo menos traumático, logrando como resultado una mayor satisfacción del cliente con el producto.

El tipo de proyectos que mejor se adaptan a metodologías ágiles es un tema que se ha debatido de forma constante desde la aparición de las mismas suponiendo un gran dilema para

muchas compañías a la hora de plantearse un proyecto de software desde el punto de vista adaptativo que ofrecen éstas o desde un punto de vista más planificado como el que ofrecen las metodologías tradicionales.

Diversas investigaciones y, sobre todo, la experiencia, han concluido que las metodologías ágiles dan sus mejores resultados en proyectos de pequeño tamaño que reúnen las siguientes condiciones.

- Criticalidad baja
- Desarrolladores con experiencia
- Constantes cambios de requisitos
- Plantillas cortas de desarrolladores (< 10 personas)

Sin embargo, existe también constancia de proyectos a gran escala que finalizaron de forma exitosa utilizando metodologías ágiles. Una compañía que apuesta especialmente por la utilización de metodologías ágiles incluyendo el desarrollo de proyectos a gran escala es la British Telecom (BT)[http://www.methodsandtools.com/archive/archive.php?id=43]

RUBY ON RAILS

Ruby on Rails es un framework de aplicaciones web de código abierto orientado a las metodologías de desarrollo web ágiles.

La finalidad de *Ruby on Rails* es la de proveer un entorno de desarrollo web funcional cómodo y simple para el desarrollador, permitiéndole programar a más alto nivel y empleando menos líneas de código que si empleara otros frameworks.

Para cumplir con lo mencionado en las líneas anteriores *Ruby on Rails* se basa en los siguientes principios como filosofía.

- *Don't Repeat Yourself (DRY)*: Referido a no usar el mismo código o las mismas definiciones en múltiples lugares de la aplicación. Rails para ello provee de los medios necesarios para que las variables y los fragmentos de código que puedan tender a repetirse ocupen un único y accesible lugar dentro de la aplicación.
- *Convention over Configuration (CoC):* Significa que el programador solo ha de especificar los aspectos no convencionales de la aplicación. Un ejemplo de esto se encuentra en la relación de las clases que representan el modelo de la arquitectura MVC con la base de datos. Por convención, una clase del modelo se corresponderá con una tabla de la base de datos y tendrá como nombre el singular del título de la tabla. (Por ejemplo para una tabla llamada STUDENTS, el nombre de la clase del modelo asociada será *Student).*

Este framework se basa en el lenguaje *Ruby* y en una implementación de la arquitectura Modelo-Vista-Controlador para el desarrollo de aplicaciones web, aspectos que a continuación se comentan en mayor detalle.

Lenguaje Ruby

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos creado en 1995 por el programador japonés Yukihiro *Matz* Matsumoto.

El creador del lenguaje, fascinado por la potencia y las posibilidades de los lenguajes de scripting, y admirador del paradigma de programación orientado a objetos se sintió atraído por unir ambos encontrando en los lenguajes Perl y Python los referentes más cercanos a lo que buscaba. Sin embargo, sus deseos por utilizar un lenguaje "más potente que Perl y más orientado a objetos que Python" le llevaron a desarrollar su propio lenguaje de programación.

Además de lo anterior, *Ruby* focaliza sus objetivos en la productividad y en la diversión del programador, centrando el diseño de sistemas en las necesidades humanas (el programador) en lugar de en las de la máquina e intentando proveer para ello una sintaxis lo más clara y legible posible.

En definitiva, *Ruby* presenta las siguientes características:

- Orientado a objetos
- Cuatro niveles de ámbito de variable: global, clase, instancia y local.
- Manejo de excepciones
- Iteradores y clausuras o closures (pasando bloques de código)
- expresiones regulares nativas similares a las de Perl a nivel del lenguaje
- posibilidad de redefinir los operadores (sobrecarga de operadores)
- recolección de basura automática
- altamente portable
- Hilos de ejecución simultáneos en todas las plataformas usando green threads
- Carga dinámica de DLL/librerías compartidas en la mayoría de las plataformas
- introspección, reflexión y metaprogramación
- amplia librería estándar
- soporta inyección de dependencias

- soporta alteración de objetos en tiempo de ejecución
- continuaciones y generadores

La implementación oficial de *Ruby* es distribuida bajo una licencia de software libre

Implementación MVC

Ruby on Rails se basa en la arquitectura MVC explicada anteriormente para el desarrollo de aplicaciones web, implementándose la misma de la siguiente forma

- Implementación del **modelo**

El modelo maneja los datos de las distintas tablas de las bases de datos. Para ello, Rails asocia cada tabla en la base de datos con una clase *Model*. Estos ficheros contendrán en concreto los distintos campos de la tabla así como sus cardinalidades y relaciones con otras tablas, además de posibles metadatos e instrucciones adicionales para las operaciones de creación, actualización, búsqueda y eliminación de registros.

Para facilitar la labor del programador al trabajar con el modelo, Rails implementa el patrón ***Active Record*** para envolver las sentencias SQL que operarán sobre la base de datos en sentencias a más alto nivel que serán las que se incluirán en las mencionadas clases *Model.*

- Implementación del **controlador**

Los controladores en Rails consisten en distintas clases *Ruby* que serán las encargadas de transformar e interactuar con los datos del modelo. Cada acción que se realice será representada por un método determinado de una de estas clases.

- Implementación de la **vista**

La vista tiene como objetivo mostrar el resultado de los métodos que han sido implementados en los controladores. Por defecto, el método empleado por Rails para representar la vista son ficheros ***.rhtml o html.erb***, también conocidos como de *Ruby Embebido,* cuyo contenido combina sentencias HTML con sentencias *Ruby* de una forma similar a la que sucede por ejemplo con los ficheros JSP. Además, se podrán incluir también en esta parte ficheros de hojas de estilo, javascript o HTML relacionados.

3. FUNCIONALIDADES DE LA APLICACIÓN

Este capítulo contendrá una descripción de los requisitos en los cuales se basará el funcionamiento de la aplicación final.

La metodología de desarrollo empleada no contempla específicamente una fase de análisis de requisitos previa al desarrollo sobre la cual basar éste, sino que confía a su alta adaptabilidad y a su cercanía al cliente la aparición de los distintos requisitos que marcarán el futuro funcionamiento de la aplicación. Sin embargo, para facilitar la labor del lector y empezar a dar una visión clara de lo que será el funcionamiento del sistema, se ha optado por incluir todos los requisitos surgidos durante el desarrollo de la aplicación, recogidos en casos de uso, en este capítulo.

ACTORES DE LA APLICACIÓN

Antes de empezar a pensar en posibles requisitos, resulta conveniente definir quiénes van a ser los usuarios del sistema, y si es posible dividir a estos en distintos roles según el funcionamiento que estos esperen del sistema.

Los principales protagonistas y destinatarios de esta aplicación son estudiantes, los cuales podrán realizar y responder preguntas tipo-test, además de añadir un conocimiento adicional según formulen o respondan a una pregunta.

Por otro lado, un sistema de esta clase suele requerir un administrador que lo regule; y además los profesores de estos estudiantes pueden estar interesados en obtener información sobre el uso que sus alumnos estén haciendo de la aplicación, así como de tomar nota de aquellas preguntas más interesantes que hayan sido formuladas.

Por tanto, puede concluirse que esta aplicación poseerá dos actores:

El **alumno**, quien poseerá un papel activo en la aplicación.

El **profesor**, el cual ejercitará labores de observación y de administración.

FUNCIONES DE LA APLICACIÓN

Una vez han quedado definidos los actores de la aplicación, el siguiente paso es ir definiendo las diferentes funciones que cada uno de estos actores espera de la aplicación. Estas funciones se representarán primero a grandes rasgos a partir de diagramas de casos de uso, y pos-

teriormente, estos serán explicados en profundidad, con lo que los requisitos principales de la aplicación podrán resultar fácilmente extraídos.

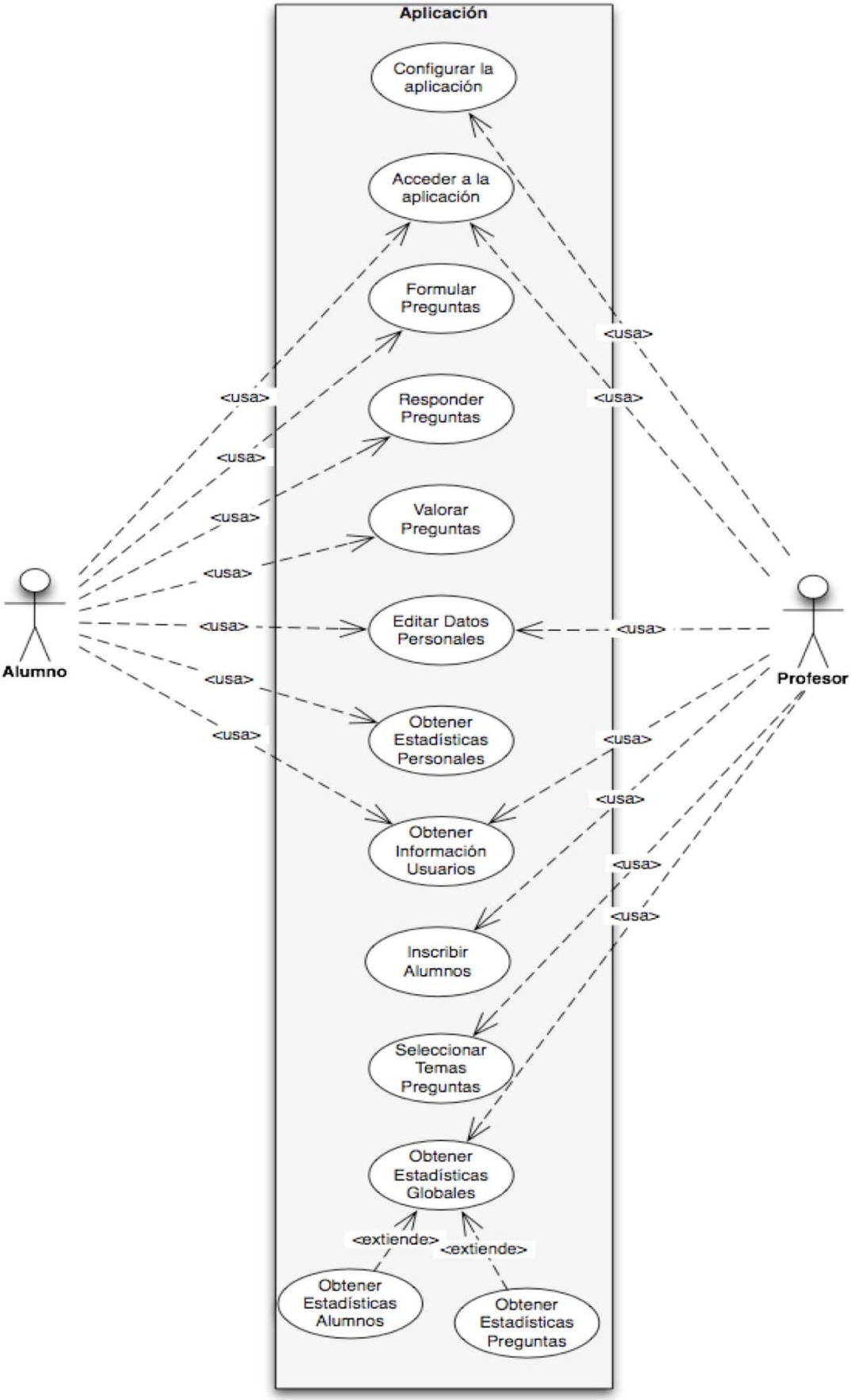


Figura 2: Diagrama de casos de uso

Descripción de los casos de uso:

Nombre:	**Configurar la aplicación**
Actores:	Profesor
Meta:	En su primera ejecución, establecer el primer usuario de la aplicación (un profesor), el cual será el encargado de ir registrando a otros alumnos y profesores en la aplicación. En la primera ejecución y sucesivas, configurar los parámetros correspondientes al servidor encargado del envío de correos electrónicos.
Precondiciones:	No tiene
Descripción:	
1. El profesor ha iniciado la aplicación por primera vez o ha abierto la URL de la página de configuración en su navegador 2. El profesor rellena la información relativa a sus datos personales y al envío de correos electrónicos. 3. Un usuario de rol profesor es creado en la base de datos de la aplicación con los datos personales (pudiendo a partir de este momento el profesor acceder mediante su *login* y contraseña) y los parámetros correspondientes al servidor de correo se almacenan en el fichero correspondiente de configuración de *Rails*.	
Post-condiciones:	No tiene

Tabla 1: Caso de Uso. Configurar la aplicación

Nombre:	**Acceder a la aplicación**
Actores:	Profesor, Alumno
Meta:	Iniciar una sesión como usuario de la aplicación
Precondiciones:	El usuario debe de haber sido previamente inscrito en la aplicación
Descripción:	
1. Al usuario se le muestra la pantalla de bienvenida a la aplicación en la cual se encuentra el cuadro de diálogo de acceso 2. El usuario introduce su *login* y su contraseña en el cuadro de diálogo de acceso 3. La aplicación contrasta con la base de datos la existencia de los datos introducidos por el usuario. *<si los datos son correctos>* 4. El usuario inicia una sesión en la aplicación entrando en el área restringida a usuarios registrados *<si los datos no son correctos>* 4. Se vuelve a mostrar la pantalla de bienvenida alertando al usuario de que la información introducida no es correcta	
Post-condiciones:	El usuario puede ahora ejecutar cualquier otra función correspondiente a su rol dentro de la aplicación.

Tabla 2: Caso de Uso. Acceder a la aplicación

Nombre:	**Formular preguntas**
Actores:	Alumno
Meta:	Crear una pregunta dentro de la aplicación
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El usuario selecciona la opción de “Realizar Pregunta” en el menú principal 2. El usuario rellena un formulario para la creación de una nueva pregunta con todos los campos requeridos para esta función. 3. Una vez rellenado el formulario, se pulsa el botón de “Enviar”. *<si el formulario se ha rellenado correctamente>* 4. Se crea y se carga la pregunta en la base de datos. *<si el formulario no se ha rellenado correctamente>* 4. Se notifica al usuario de los campos que contienen error con el objetivo de que los corrija y pueda publicar la pregunta.	
Post-condiciones:	La pregunta creada estará disponible para ser respondida y valorada por el resto de usuarios justo tras el momento de su creación

Tabla 3:Caso de Uso. Formular preguntas

Nombre:	**Responder preguntas**
Actores:	Alumno
Meta:	Responder una pregunta dentro de la aplicación
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña. Ha de haber al menos una pregunta publicada que el usuario pueda responder.
Descripción:	
1. El usuario selecciona “Responder pregunta” en el menú principal 2. El usuario selecciona de la lista de preguntas que se le presenta aquella que desea responder. 3. El usuario elige una de las posibles respuestas a la pregunta. 4. Se informa al usuario sobre el acierto en su elección.	
Post-condiciones:	Una vez ha respondido una pregunta, el usuario puede pasar a valorarla y comentarla.

Tabla 4: Caso de Uso. Responder preguntas

Nombre:	**Valorar preguntas**
Actores:	Alumno
Meta:	Dar una valoración a una pregunta según el interés que tenga.
Precondiciones:	La pregunta a valorar ha tenido que ser respondida.
Descripción:	
1. El usuario puntúa según su criterio y dentro de un determinado rango la utilidad y el interés de la pregunta. Además puede tener la opción de justificar su puntuación o de añadir algún conocimiento adicional relacionado con la pregunta. 2. El usuario envía su valoración, quedando ésta publicada.	
Post-condiciones:	Una vez que una pregunta ha sido respondida y valorada, ya no puede ser respondida de nuevo por ese mismo usuario.

Tabla 5: Caso de Uso. Valorar preguntas

Nombre:	**Editar datos personales**
Actores:	Profesor, Alumno
Meta:	Introducir y editar información personal y de contacto para el resto de usuarios de la aplicación
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El usuario selecciona la opción de “Modificar” en el menú principal 2. El usuario rellena un formulario para la publicación de información personal o de contacto 3. Una vez rellenado el formulario, se pulsa el botón de “Enviar”. *<si el formulario se ha rellenado correctamente>* 4. Se carga la información introducida en la base de datos. *<si el formulario no se ha rellenado correctamente>* 4. Se notifica al usuario de los campos que contienen error para su corrección.	
Post-condiciones:	Los cambios introducidos pasan a estar inmediatamente activos en la aplicación.

Tabla 6: Caso de Uso. Editar datos personales

Nombre:	**Obtener estadísticas personales**
Actores:	Alumno
Meta:	Obtener información sobre la actividad propia en la aplicación
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El usuario selecciona la opción de “Estadísticas Personales” en el menú principal 2. El usuario obtiene datos y gráficas actualizados de su actividad en la aplicación. La información con la que se monitorizará dicha actividad concernerá tanto a las valoraciones realizadas como recibidas en las preguntas que respectivamente haya realizado o contestado, así como los aciertos o los errores en las respuestas.	
Post-condiciones:	No hay.

Tabla 7: Caso de Uso. Obtener estadísticas personales

Nombre:	**Obtener información sobre usuarios**
Actores:	Profesor, Alumno
Meta:	Obtener información personal y de contacto que haya sido compartida por otros usuarios del sistema
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El usuario selecciona la opción de “Ver Usuarios” en el menú principal 2. El usuario obtiene un listado de los usuarios del sistema donde se muestra su información personal y de contacto.	
Post-condiciones:	No hay.

Tabla 8: Caso de Uso. Obtener información sobre usuarios

Nombre:	**Inscribir usuarios**
Actores:	Profesor
Meta:	Incorporar un nuevo usuario (profesor o alumno) a la aplicación
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El profesor selecciona la opción de “Añadir Usuario” en el menú principal 2. El profesor introduce el identificador y el correo electrónico del usuario a inscribir. 3. La aplicación genera una contraseña aleatoria para el nuevo usuario y le envía un correo electrónico informándole del registro y de sus datos de acceso a la aplicación. 4. El nuevo usuario puede acceder a la aplicación e iniciar una sesión en ella con los datos del correo electrónico recibido.	
Post-condiciones:	El usuario inscrito podrá a partir de este momento iniciar una sesión en la aplicación.

Tabla 9: Caso de Uso. Inscribir usuarios

Nombre:	**Establecer temas de preguntas**
Actores:	Profesor
Meta:	Seleccionar la temática de las preguntas de la aplicación
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El profesor selecciona la opción de “Añadir Tema” en el menú principal. 2. El profesor rellena un formulario para la creación de un nuevo tema. 3. Una vez rellenado el formulario, se pulsa el botón de “Enviar”. *<si el formulario se ha rellenado correctamente>* 4. Se carga la información introducida en la base de datos. *<si el formulario no se ha rellenado correctamente>* 4. Se notifica al profesor de los campos que contienen error para su corrección.	
Post-condiciones:	Podrán crearse nuevas preguntas que entren dentro de la nueva temática que haya sido creada.

Tabla 10: Caso de Uso. Establecer temas de preguntas

Nombre:	**Obtener estadísticas de preguntas**
Actores:	Profesor
Meta:	Obtener información detallada sobre la actividad relacionada con las preguntas que se haya desarrollado dentro de la aplicación.
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El profesor hace click sobre la opción “Estadísticas Preguntas” del menú principal 2. El profesor obtiene un listado con todas las preguntas del sistema. 3. El profesor hace click en el link “Estadísticas” situado al final de la descripción de la pregunta. 4. Los datos y gráficos referentes a la actividad relacionada con la pregunta seleccionada se muestran por pantalla. Estos datos concernerán a las veces que la pregunta ha sido respondida correcta o incorrectamente y a cómo ha sido valorada por los usuarios del sistema.	
Post-condiciones:	No tiene

Tabla 11: Caso de Uso. Obtener estadísticas de preguntas

Nombre:	**Obtener estadísticas de alumnos**
Actores:	Profesor
Meta:	Obtener información detallada sobre la actividad de los distintos alumnos registrados en la aplicación.
Precondiciones:	El usuario debe haber accedido previamente a la aplicación mediante su *login* y contraseña.
Descripción:	
1. El profesor hace click sobre la opción “Ver Usuarios” del menú principal 2. El profesor obtiene un listado con los usuarios del sistema 3. El profesor hace click en el link “Estadísticas” situado al final de la descripción del usuario 4. Una descripción de los datos del usuario seleccionado aparecerán en pantalla. Si el usuario seleccionado es un alumno, además se mostrarán gráficos y datos que reflejarán la actividad del mismo en la aplicación. La información mostrada se referirá en concreto a cómo un usuario ha ido respondiendo las preguntas y cómo el resto de alumnos han respondido las suyas, al igual que a las valoraciones y comentarios a las preguntas que el usuario seleccionado haya realizado o recibido.	
Post-condiciones:	No tiene

Tabla 12: Caso de Uso. Obtener estadísticas de alumnos

FLUJO DE PÁGINAS.

La finalidad de este apartado será detallar a un nivel más bajo las distintas funcionalidades extraídas a partir de los casos de uso. Utilizando como base las descripciones de los casos de uso del apartado anterior se harán diagramas de flujo de páginas, cuyo fin será el de proporcionar una visión más detallada e intuitiva de lo que será la aplicación web a desarrollar.

Un diagrama de flujo de páginas consiste en la realización de una serie de bocetos sobre las páginas principales de la futura aplicación web y sobre cómo los usuarios navegarán entre ellas. De estos bocetos se podrá extraer una idea clara de cómo un usuario navegará para poder llevar a cabo las distintas funcionalidades descritas por los casos de uso así como ayudar a especificar algunos de los requisitos que la futura aplicación deberá cumplir.

Configurar la Aplicación

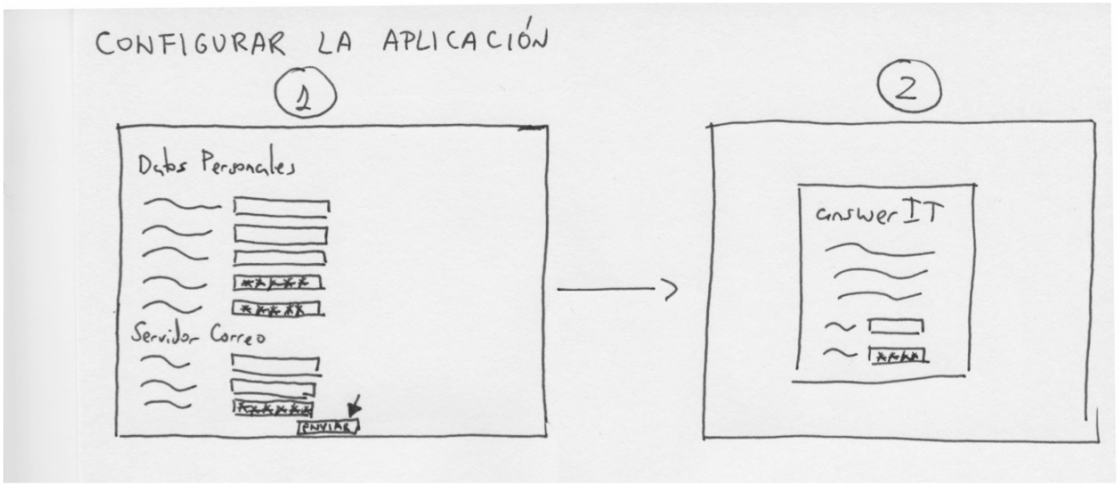


Figura 3: Flujo de Páginas. Configurar la aplicación

1. La primera vez que se ejecute la aplicación, nada más ser instalada, el profesor será automáticamente redireccionado a la página de configuración, en la cual podrá introducir sus datos para ser registrado como primer usuario de la aplicación, y los datos del servidor de correo encargado del envío de notificaciones y contraseñas. En caso de querer acceder a la página de configuración en cualquier otra ocasión, bastará con teclear su URL en la barra de direcciones del navegador (Por razones de seguridad, es recomendable que esta página de configuración solo sea accesible localmente). Una vez completado el formulario, se hará clic en el botón de “Enviar”.

2. Tras pulsar el botón de “Enviar”, la aplicación redirigirá al profesor a la página de acceso, desde donde le será posible iniciar una sesión con su *login* y contraseña.

Acceder a la Aplicación

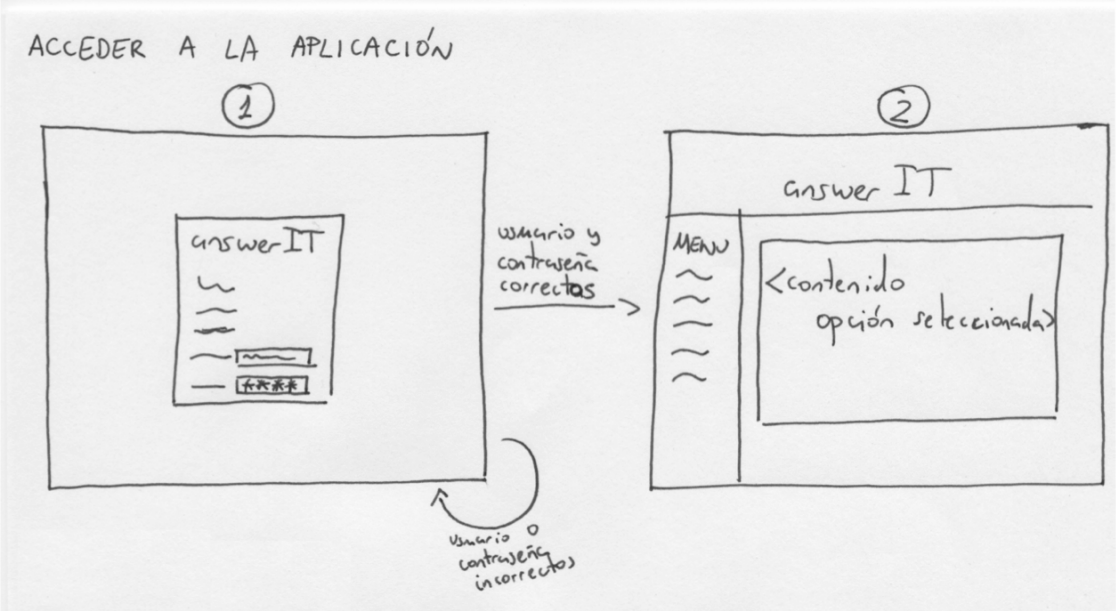


Figura 4: Flujo de Páginas. Acceder a la aplicación

1. Al acceder a la URL de la aplicación, lo primero que aparece es la pantalla de bienvenida de la misma, donde el usuario deberá autenticarse para poder acceder introduciendo su identificador de usuario y contraseña.

2. Si el identificador y la contraseña introducidos por el usuario son correctos, el usuario accede a la aplicación mostrándosele el contenido de la opción seleccionada por defecto y un menú a partir del cual podrá ejecutar cualquiera de las operaciones asociadas a su rol.

Formular Pregunta

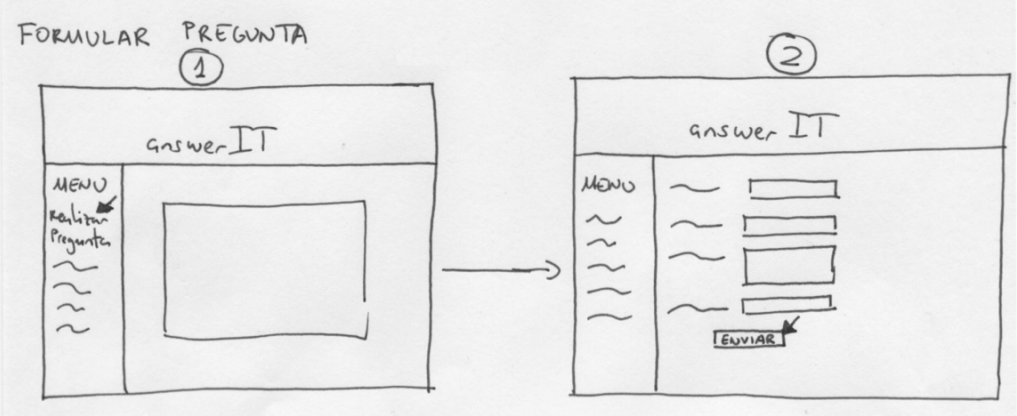


Figura 5. Flujo de Páginas. Formular pregunta

1. Un usuario de rol alumno poseerá una opción “Realizar Pregunta” en su menú que le permitirá llevar a cabo esta operación

2. El usuario rellena el formulario correspondiente a una nueva pregunta en la que deberá especificar:

   - El enunciado de la pregunta
   - El tema en el cual se encuadra la pregunta
   - Las 4 posibles respuestas para la pregunta
   - La respuesta de las cuatro anteriores que es la correcta (a,b,c o d)
   - Opcionalmente, alguna información adicional explicando la veracidad de la respuesta, o bien detalles de interés relacionados con la pregunta.

Responder Preguntas

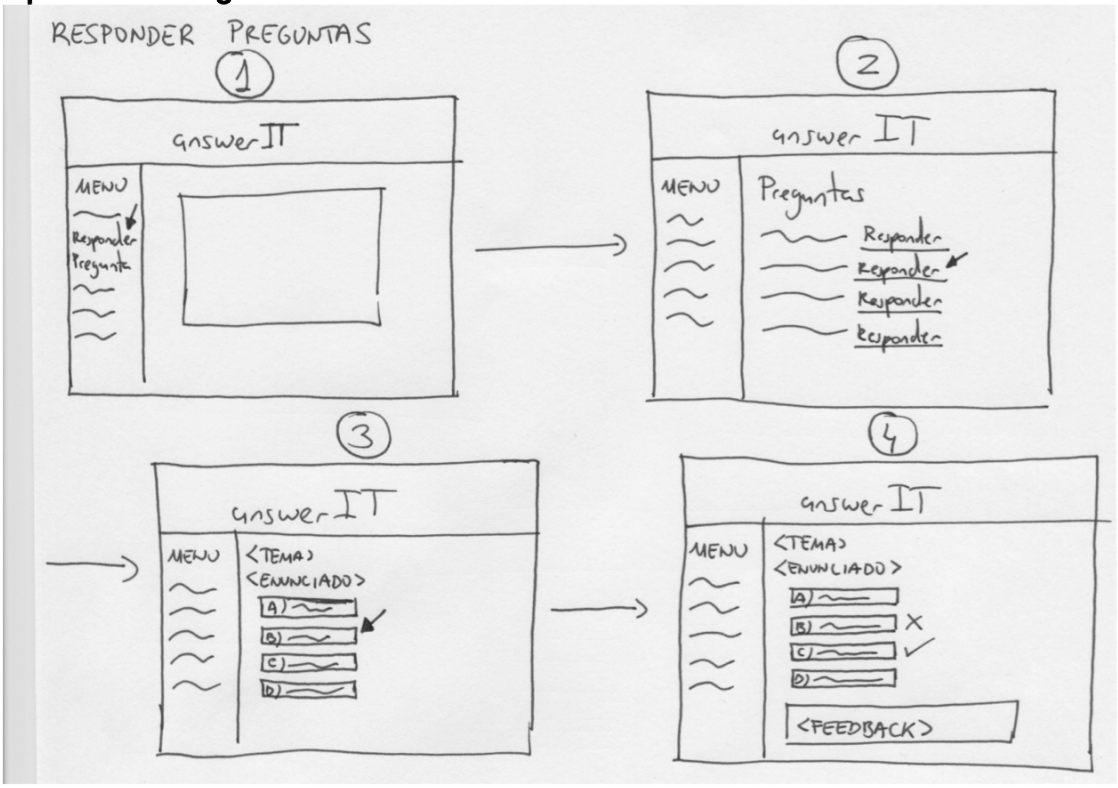


Figura 6: Flujo de Páginas. Responder preguntas

1. El usuario de rol alumno poseerá una opción “Responder Preguntas” en su menú que le permitirá llevar a cabo esta operación. Además, “Responder Preguntas” será la opción que aparecerá por defecto cuando un usuario de rol alumno acceda a la aplicación.

2. Tras seleccionar la opción mencionada, al usuario se le aparecerá una lista aleatoria de preguntas formuladas por los otros usuarios de la aplicación a las cuales el usuario que esté ejecutando la sesión no haya respondido. Presionando el enlace “Responder” situado al lado de cada una de las preguntas, el usuario podrá responder la pregunta relacionada por dicho enlace.

3. Cuando se ha escogido una pregunta en concreto para responder, aparece una pantalla con cuatro campos seleccionables correspondientes a cada una de las cuatro respuestas a la pregunta.

4. Una vez se hace clic sobre una de las respuestas, la aplicación inmediatamente mostrará si la respuesta escogida es correcta o no, y en caso de no serlo, resaltará además la respuesta correcta. Además aparecerá debajo de las respuestas un recuadro en el cual aparecerá el feedback o información adicional que el autor de la pregunta haya deseado incluir.

Valorar Preguntas:

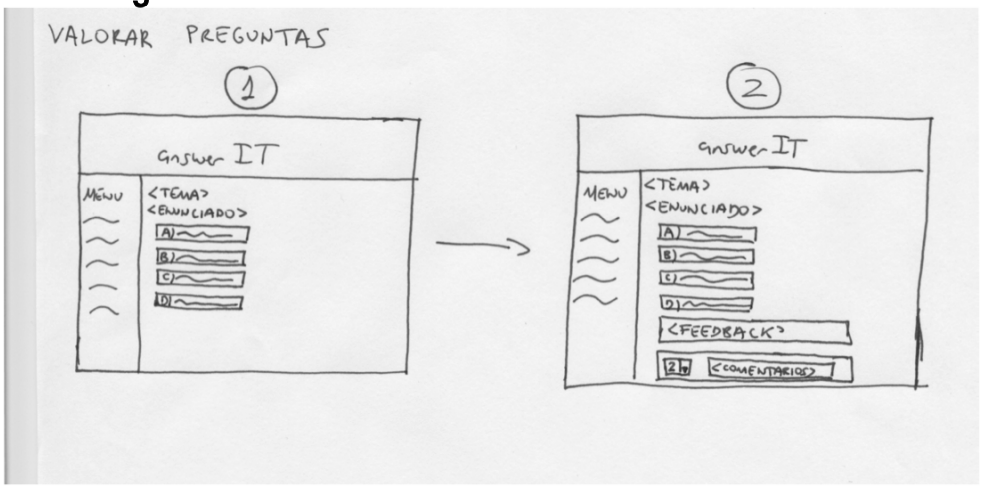


Figura 7: Flujo de Páginas. Valorar Pregunta

1. Para valorar una pregunta es preciso responderla antes, por ello, habrá que seleccionar “Responder Preguntas” en el menú y seleccionar una pregunta a responder y responderla (apartados 1, 2 y 3 de la funcionalidad “Responder Preguntas” anteriormente descrita) para poder llevar esta operación a cabo.

2. Junto al ya mencionado feedback que aparece al responder una pregunta, aparecerá otro cuadro de diálogo que permitirá al usuario otorgar una puntuación entre 0 y 5 a la pregunta que acaba de responder y realizar un comentario sobre la misma si lo desea.

Editar Datos Personales

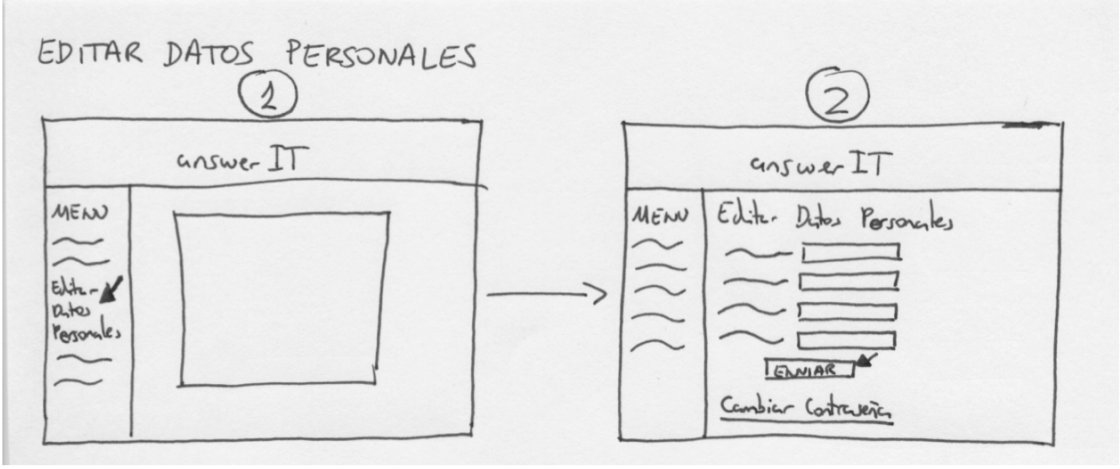


Figura 8: Flujo de Páginas. Editar datos personales

1. Para poder editar sus datos personales, cualquier usuario de la aplicación deberá hacer clic sobre la opción correspondiente de su menú.

2. Una vez seleccionada la opción, el usuario encontrará un formulario donde le será posible modificar cualquier información relacionada consigo mismo en la aplicación a excepción del identificador de usuario.

Cambio de contraseña

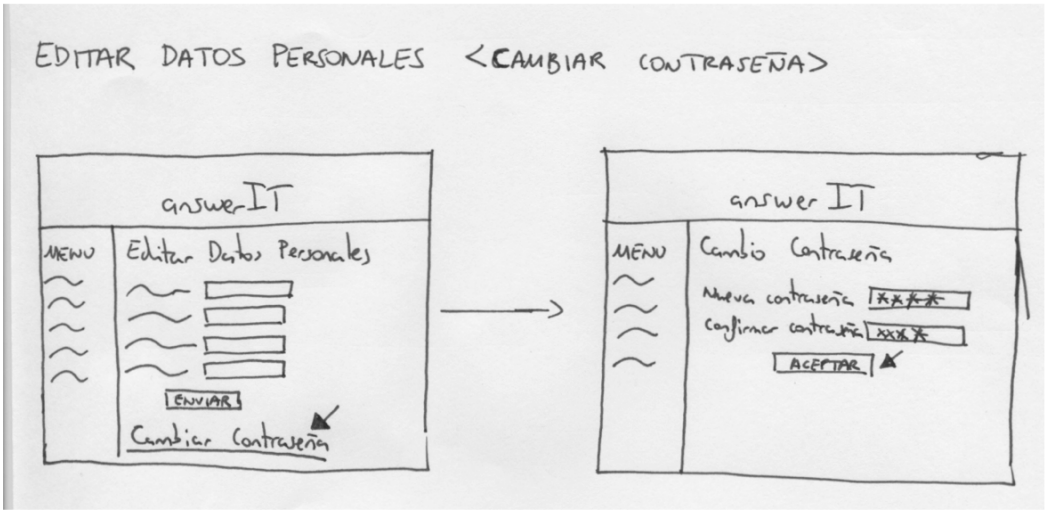


Figura 9: Flujo de Páginas: Cambio de contraseña

1. Debajo del formulario mencionado en el apartado 2 anterior, aparecerá un enlace que permitirá a un usuario cambiar su clave de acceso a la aplicación.

2. Presionado dicho enlace, el usuario deberá introducir su nueva clave y confirmarla para habilitarla para las siguientes sesiones que inicie en la aplicación.

Obtener Estadísticas Personales

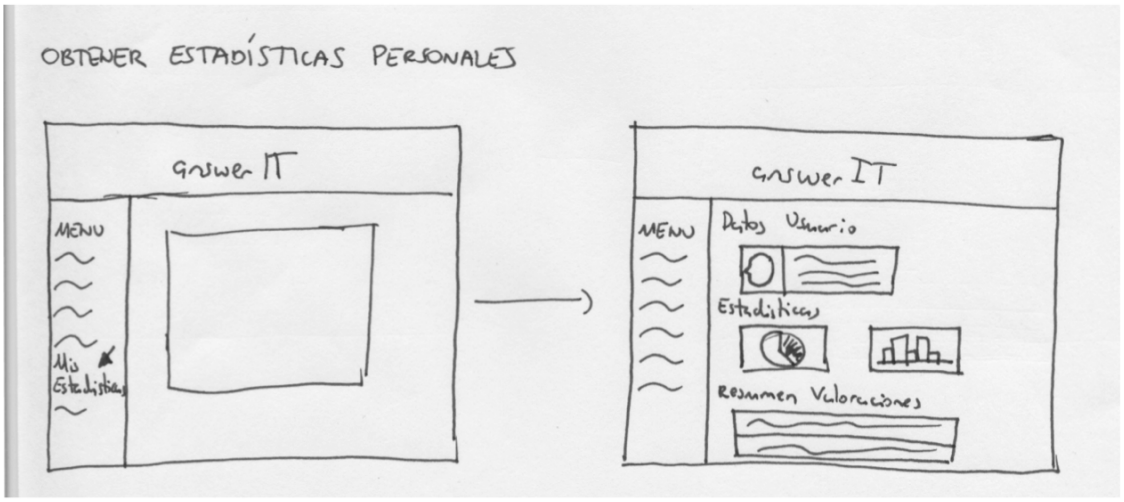


Figura 10: Flujo de Páginas. Obtener estadísticas personales

1. Para poder acceder a sus estadísticas personales, los alumnos deberán hacer clic en la opción correspondiente en el menú principal. Un alumno también podrá acceder a sus estadísticas seleccionándose a si mismo dentro de la operación “Ver Usuarios”.

2. Seleccionando esta opción, el usuario podrá acceder a detalles y estadísticas referentes a su actividad, entre los que se encontrarán:

   - Resumen de las respuestas recibidas por las preguntas creadas por el usuario, recogiendo tanto si estas respuestas fueron acertadas o fallidas como las valoraciones numéricas otorgadas por los usuarios al responder. Un gráfico de tarta será utilizado para representar los aciertos y fallos y uno de barras para las valoraciones.

   - Resumen de las respuestas que haya realizado el usuario a preguntas de otros usuarios reflejando sus aciertos y sus fallos así como las valoraciones que haya ido otorgando. De nuevo un gráfico de tarta será utilizado para representar los aciertos y fallos y uno de barras para las valoraciones.

   - Resumen completo de las valoraciones y los comentarios asociados que el usuario ha recibido en sus preguntas. El usuario podrá ver tanto las distintas puntuaciones que ha recibido como los comentarios, pero ningún dato que identifique al autor de las mismas.

   - Resumen completo de las valoraciones y los comentarios asociados que el usuario ha realizado a preguntas de otros usuarios. El usuario podrá ver tanto la puntuación que en su momento otorgó como el comentario, pero en ningún momento podrá ver a qué usuario ha valorado.

Obtener Información sobre Usuarios

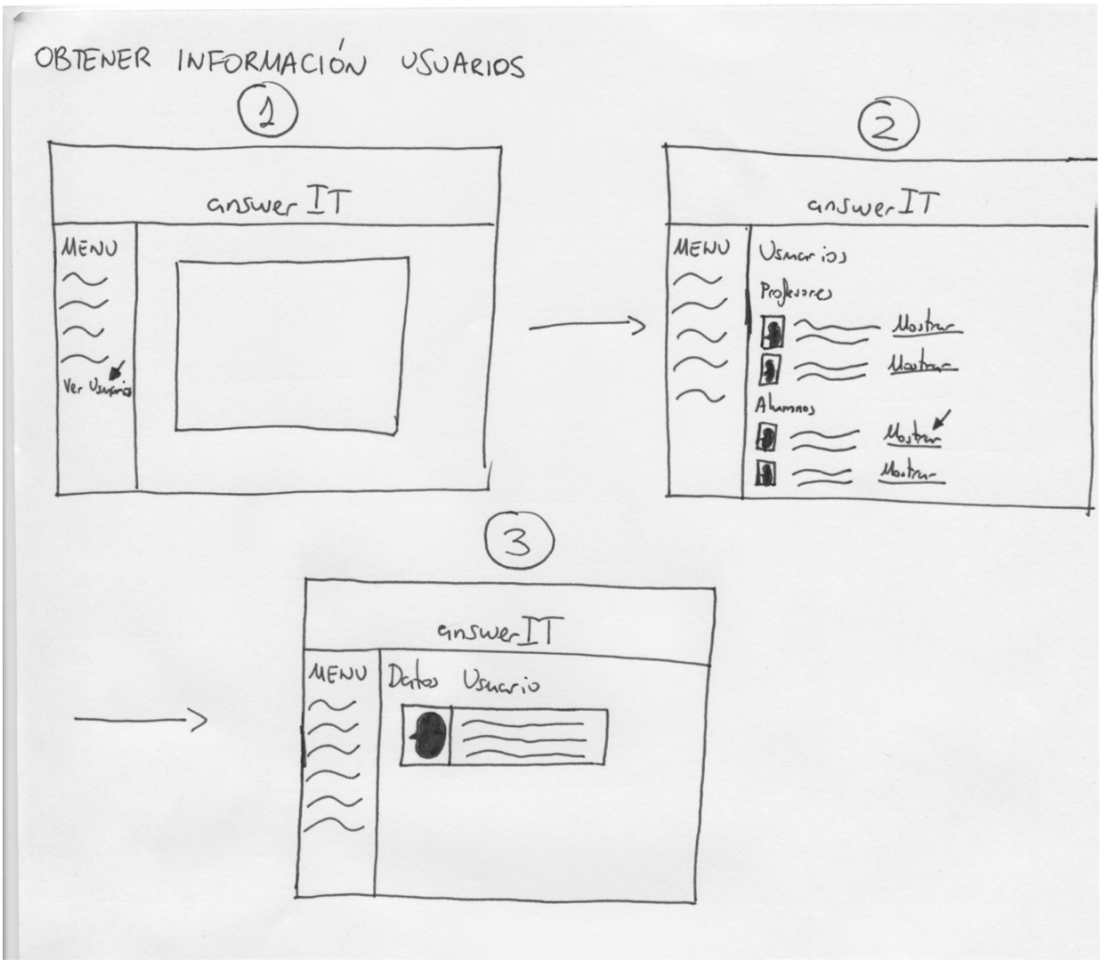


Figura 11: Flujo de Páginas. Obtener información sobre usuarios

1. Cualquier usuario de la aplicación puede consultar información personal o de contacto de otros usuarios en la aplicación a partir de la opción “Ver Usuarios” del menú principal.

2. Activada esta opción aparece un listado de los distintos usuarios del sistema separados por rol y ordenados alfabéticamente. Al lado de la breve descripción de cada usuario aparecerá un enlace “Mostrar”

3. Al activar este enlace, aparece una ficha en la cual pueden verse el identificador para la aplicación, el nombre, los apellidos y el e-mail de contacto de un usuario.

Inscribir Usuarios

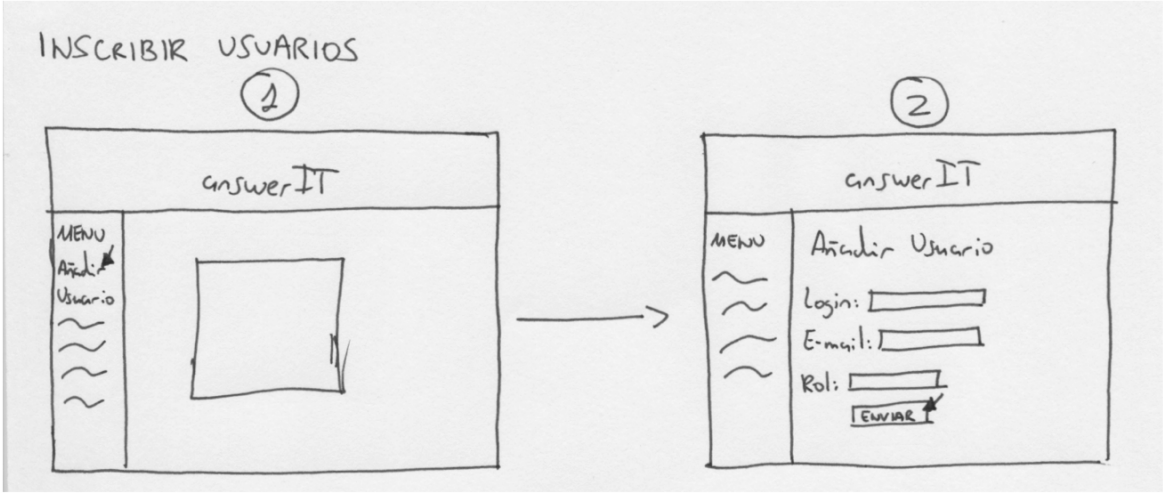


Figura 12: Flujo de Páginas. Inscribir usuarios

1. Esta opción solo está disponible para un usuario con el rol de profesor, que podrá llevarla a cabo haciendo clic en la opción correspondiente del menú principal.

2. En el formulario que aparece, el profesor deberá introducir el identificador de usuario, el rol y el e-mail del usuario que va a inscribir en la aplicación.

Establecer Temas de Preguntas

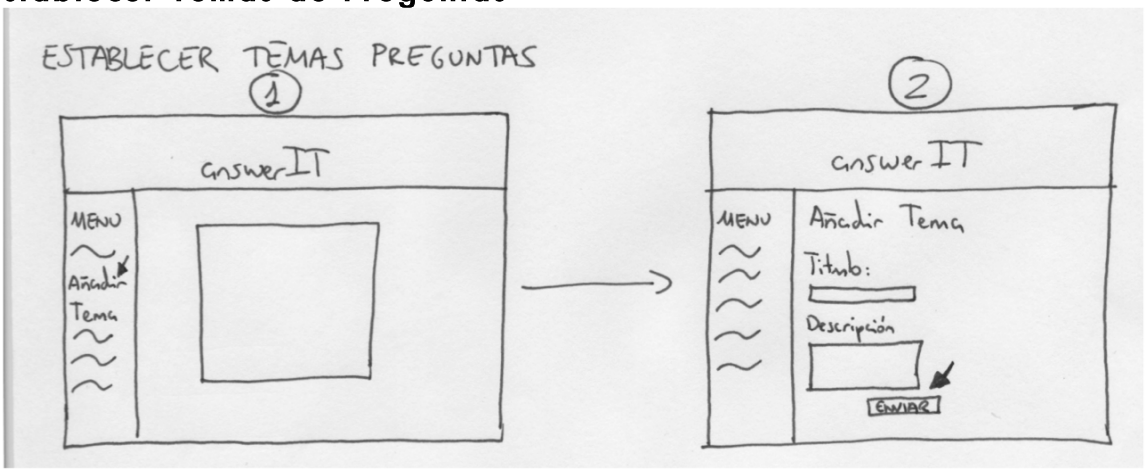


Figura 13: Flujo de Páginas. Establecer temas de preguntas

1. Mediante esta opción, el profesor establece los temas en los cuales se encuadrarán las preguntas que harán los alumnos. El profesor puede crear un nuevo tema seleccionando la correspondiente acción del menú principal.

2. En el formulario que aparece, el profesor deberá introducir el título del tema y una descripción del mismo. Una vez creado el tema, los alumnos podrán incluir preguntas en él.

Obtener Estadísticas de Preguntas

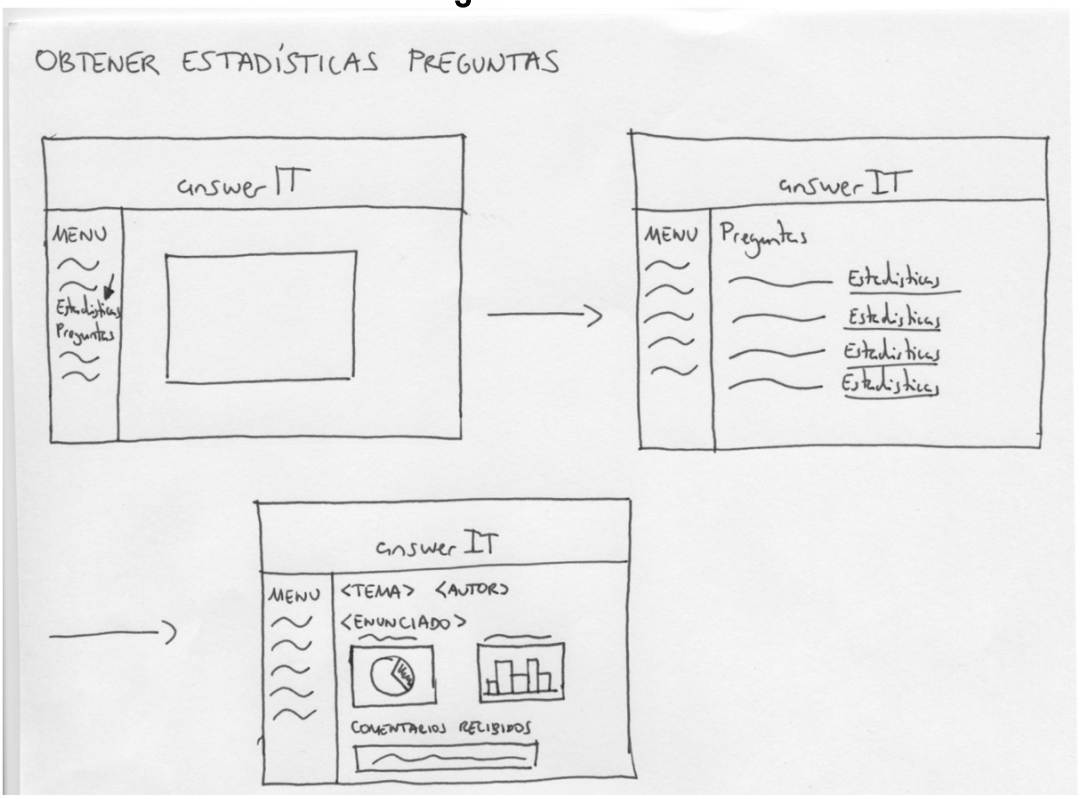


Figura 14: Flujo de Páginas. Obtener estadísticas de preguntas

1. El usuario de rol profesor poseerá una opción “Estadísticas Preguntas” en su menú principal que le permitirá obtener información sobre todas las preguntas que han sido realizadas por los alumnos hasta el momento.

2. Tras hacer clic en la mencionada opción del menú, aparecerá un listado con todas las preguntas. Presionando el enlace “Estadísticas” que aparece al lado de cada una de las preguntas se podrá acceder a las estadísticas concretas de la pregunta relacionada con dicho enlace.

3. Una vez seleccionada una pregunta para mostrar sus estadísticas aparecerán datos y gráficos que indicarán cómo han respondido los alumnos a dicha pregunta (registrando sus aciertos y errores) y cómo la han ido valorando. También será posible encontrar al final un resumen de todas las valoraciones que la pregunta ha recibido junto con los comentarios que adicionalmente se hayan realizado.

Obtener Estadísticas de Alumnos

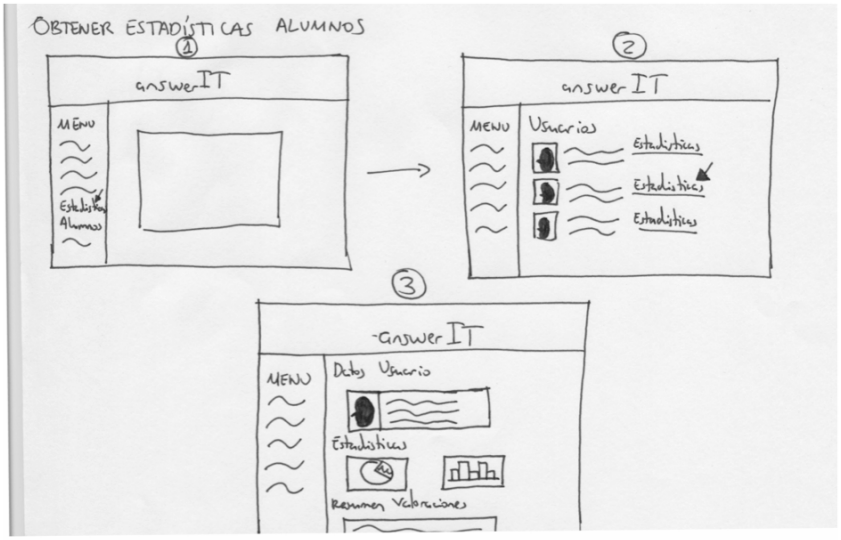


Figura 15: Flujo de Páginas. Obtener estadísticas de alumnos

1. Un usuario de rol profesor podrá acceder a las estadísticas de la actividad de todos sus alumnos mediante la opción “Ver Usuarios” de su menú principal.

2. Al activar dicha opción, aparecerá un listado con todos los usuarios de la aplicación, y para cada uno de los usuarios con el rol alumno, aparecerá un enlace con la palabra “Estadísticas” a la derecha.

3. Presionando dicho enlace para uno de los alumnos, el profesor podrá acceder a las siguientes estadísticas sobre el alumno correspondiente:

   - Resumen de las respuestas recibidas por las preguntas creadas por el alumno, recogiendo tanto si estas respuestas fueron acertadas o fallidas como las valoraciones numéricas otorgadas por los usuarios al responder. Un gráfico de tarta será utilizado para representar los aciertos y fallos y uno de barras para las valoraciones.

   - Resumen de las respuestas que haya realizado el alumno a preguntas de otros alumnos reflejando sus aciertos y sus fallos así como las valoraciones que haya ido otorgando. De nuevo un gráfico de tarta será utilizado para representar los aciertos y fallos y uno de barras para las valoraciones.

   - Resumen completo de las valoraciones y los comentarios asociados que el alumno ha recibido en sus preguntas. En este caso, el profesor tendrá acceso tanto a la puntuación otorgada, como a los comentarios como a la información que identifica al autor de la valoración.

   - Resumen completo de las valoraciones y los comentarios asociados que el alumno ha realizado a preguntas de otros alumnos. El profesor podrá ver a qué pregunta de qué alumno le fue dada la valoración, la puntuación que se le otorgó y los comentarios adicionales que el alumno hizo a la pregunta, en caso de haberlos.

4. DISEÑO INICIAL

El haber elegido *Ruby on Rails* como marco de desarrollo para esta aplicación implica directamente la elección del patrón arquitectónico Modelo-Vista-Controlador como base para el proyecto. De la misma forma, este marco de desarrollo establece una serie de convenciones que serán seguidas en su mayor medida para llevar a cabo la implementación y que por tanto influirán de manera importante en el diseño detallado.

El siguiente diagrama de clases aclara cómo es implementada la arquitectura MVC en Rails (de la cual ya se habló brevemente en el capítulo 2.d):

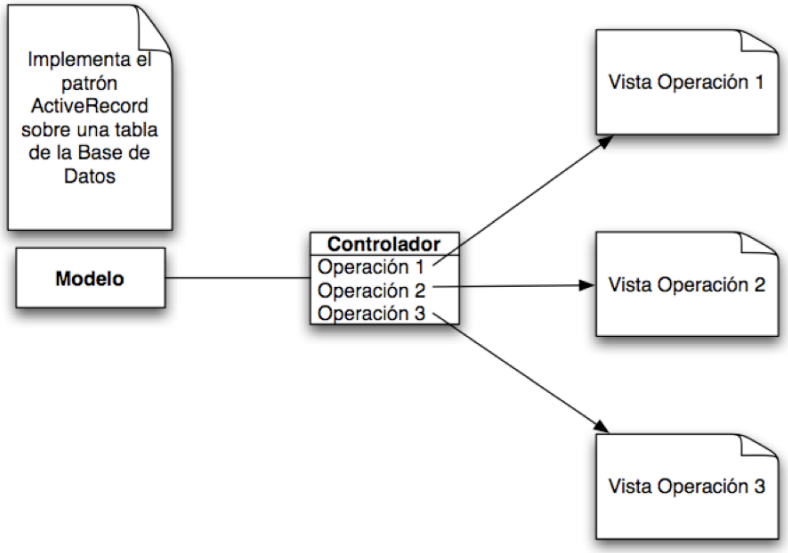


Figura 16: Implementación MVC en Rails

El modelo encapsula una tabla de la base de datos (por convención, el nombre de la clase del modelo es el mismo de la tabla en la base de datos en singular), el controlador contiene una serie de operaciones en los que se emplearán y modificarán los datos presentes en el modelo, y por cada una de estas operaciones, existirá una vista, desde la cual el usuario podrá configurar estas operaciones o recibir información sobre el estado de las mismas.

Sobre esta convención se basa una de las prestaciones más útiles que ofrece *Ruby on Rails*, conocida como ***scaffold***. Esta prestación consiste en un pequeño *script* que, a partir de una especificación de una tabla de una base de datos (nombre de la tabla y campos), genera el modelo para la misma, el controlador con cuatro funciones sobre ella (crear, leer, actualizar, eliminar) y las vistas para cada una de estas cuatro funciones. En definitiva, crea una aplicación web capaz de realizar las cuatro operaciones básicas de una base de datos sobre la tabla especificada. El código generado por ***scaffold*** es una aplicación completamente funcional

que, además, puede ser empleado como esqueleto para la construcción aplicaciones más complejas.

Como sería poco inteligente desperdiciar la potencia de una herramienta como ***scaffold*** para la construcción de una aplicación como la que ataña a este proyecto, es lógico adaptar el diseño de la aplicación a su uso. Esta decisión lleva inevitablemente a empezar a pensar en el diseño de la base de datos de la aplicación, a partir del cual se podrán obtener las tablas sobre las que ejecutar ***scaffold.***

DISEÑO DE LA BASE DE DATOS

Para diseñar la base de datos hay que plantearse qué elementos de los que formarán parte de la aplicación son aquellos que se quieren almacenar.

De lo descrito en el capítulo anterior se obtiene que el elemento principal a almacenar son las **preguntas**, alrededor de ellas gira la aplicación puesto que es el elemento básico de interacción entre los distintos usuarios de la aplicación.

Las preguntas son publicadas por **usuarios**, los cuales además deben de tener la posibilidad de iniciar una sesión en el sistema y de acceder a cierta información correspondiente a otros usuarios, por tanto, los usuarios serán otro elemento a almacenar.

Finalmente, las preguntas se hallan clasificadas en **temas**, que son establecidos previamente por un profesor. Ello lleva a que antes de formularse una pregunta sea necesario conocer los temas en los cuales ésta puede enmarcarse, luego los temas constituyen también conocimiento a almacenar.

Una vez ya se ha determinado que **preguntas, usuarios y temas** son las entidades existentes en la base de datos, el siguiente paso es determinar cómo se hayan estas entidades relacionadas:

- Un usuario publica preguntas
- Un usuario responde preguntas
- Un usuario valora preguntas. Cada pregunta que es valorada por un usuario recibe una determinada puntuación y puede recibir comentarios a modo de *feedback.*
- Un profesor (usuario) crea un tema
- Las preguntas pertenecen a un determinado tema

De estas relaciones existentes, es preciso decidir cuáles de ellas interesa tener contempladas en la base de datos. Dada la naturaleza de la aplicación, se da una mayor importancia a la interacción entre usuarios y, por tanto, a la valoración de las preguntas que a las respuestas

que se dan a las mismas, por ello esta última relación puede ser descartada. Lo mismo sucede con la publicación de temas, es cierto que está especificado que un usuario con el rol de profesor es el único que puede hacerlo, pero esta restricción es ajena al modelo de datos, y por tanto no tiene por qué ser representada en el mismo.

Sin embargo, de acuerdo a lo especificado en el capítulo anterior, el autor de cada una de las preguntas si debe de ser contemplado (pues es información sobre la actividad de un determinado alumno en el sistema que puede resultar de interés para el profesor), y lo mismo con las valoraciones realizadas sobre las preguntas y la temática de cada una de las preguntas.

El siguiente paso es determinar la cardinalidad de dichas relaciones:

- Un usuario puede publicar varias preguntas, mientras que una pregunta sólo puede tener un autor
- Un usuario puede valorar varias preguntas, y una pregunta a la vez puede contener valoraciones pertenecientes a varios usuarios.
- Un tema puede contener varias preguntas, sin embargo una pregunta no puede encuadrarse en varios temas.

A partir de este análisis es posible realizar un modelo entidad-relación que represente conceptualmente el modelo de datos a implementar:

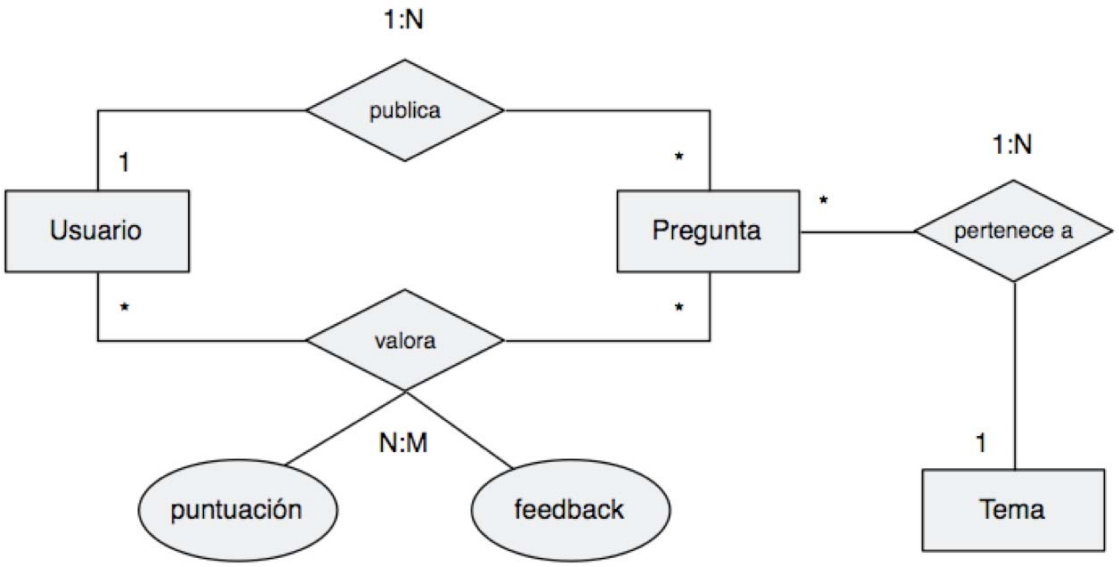


Figura 17: Modelo Entidad/Relación de la aplicación

Una vez determinadas las entidades del modelo de datos y sus relaciones, se tienen las tablas y las claves foráneas de estas que serán cargadas en la base de datos. Sin embargo, aún quedarían por determinar los atributos de cada tabla, que es lo que se hará a continuación.

Preguntas:

Según lo conocido hasta ahora, en una pregunta se debe almacenar lo siguiente:

- En primer lugar, toda pregunta necesita un **enunciado**.
- Las preguntas, tal y como se ha acordado con el cliente serán de tipo test y tendrán **cuatro opciones posibles**: *a*, *b*, *c* y *d.*
- Debe de haber un campo que recoja la **respuesta correcta** con el que se contraste la respuesta introducida por el alumno que decida contestarla.
- Se ha especificado como requisito que una pregunta, además de informar sobre qué respuesta es la correcta, puede aportar una mayor información sobre si misma a modo de **feedback**.
- Además de lo anterior, la tabla preguntas poseerá dos claves foráneas para así poder asociar cada pregunta a un **autor** y a un **tema**.
- Asimismo, Rails añade automáticamente a cada tabla un campo **id** que sirve como identificador y por los cuales los distintos elementos de la aplicación web son identificados y cargados.

Por tanto, una posible representación para la tabla *Preguntas* podría ser la siguiente:

Pregunta (id, enunciado, tema, respuestaA, respuestaB, respuestaC, respuestaD, correcta, feedback, usuario)

Usuarios:

La información que concierna a los usuarios a ser almacenada debe, por un lado, permitir al usuario autenticarse ante la aplicación, y por otro, permitir al usuario compartir información personal básica y de contacto con el resto de usuarios de la aplicación.

- En lo que se refiere a la autenticación de usuarios, cada usuario deberá tener un identificador de usuario (***login***) y una contraseña personales con los cuales podrá iniciar una sesión en el sistema.
- Asimismo debe especificarse el **rol** del usuario, ya que el uso que pueda hacer de la aplicación variará según este campo.
- Por otro lado, el **nombre** y los **apellidos** del usuario facilitan su identificación personal, y el **correo electrónico** puede ser empleado como principal vía de contacto. Además, se puede facilitar a los usuarios la posibilidad de personalizar su perfil con una **fotografía o imagen**, la cual podrán subir al sistema facilitando una URL válida.

Bajo estas premisas, la tabla *Usuarios* quedaría:

Usuario (id, login, password, nombre, apellidos, email, rol, url_foto)

Temas:

Los distintos temas en los cuales pueden ser enmarcadas no requieren de una gran cantidad de información para ser descritos. Un **título** en el cual pueda enunciarse brevemente el área de conocimiento al que el tema se refiera, y una **descripción**, mediante la cual sea posible extender la información que el título pueda proveer resulta suficiente para este fin.

Por tanto, la tabla *Temas* contendría:

Tema (id, titulo, descripcion)

Valoraciones:

Cómo ha podido verse en el anterior diagrama entidad-relación, la relación concerniente a la valoración de las preguntas por parte de los distintos usuarios de la aplicación es la única que posee cardinalidad N:M, además de dos atributos que únicamente existen en el contexto de esta relación.

Cuando un caso así aparece durante el diseño de una base de datos, la solución a la hora de traducir el modelo entidad-relación en un modelo relacional está clara: ha de crearse una nueva tabla que represente esta relación.

Esta nueva tabla tendrá:

- Dos claves foráneas que representarán a la **pregunta** que es valorada y al **usuario** que la valora.
- Los atributos contemplados para la relación en el anterior modelo entidad-relación, **puntuación** y feedback**,** también tendrán que aparecer en esta nueva tabla.

Por ello, la nueva tabla surgida, que recibirá el nombre de valoraciones, estará descrita de la siguiente manera:

Valoraciones (id, pregunta, usuario, puntuacion, feedback)

Modelo Relacional:

Una vez se ha llegado a este punto resulta posible llevar a cabo un modelo relacional que determinará el modo en que los datos serán introducidos en la base de datos de la aplicación así como la implementación de las clases correspondientes al modelo de datos.

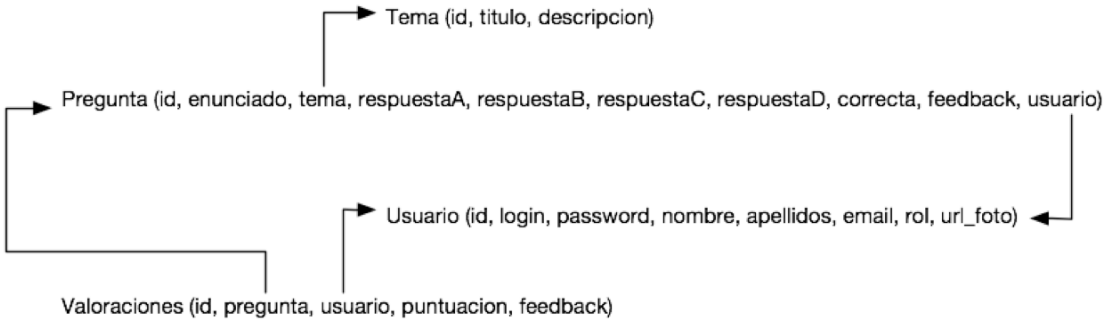


Figura 18: Modelo Relacional de la aplicación

Una vez ya se tiene un modelo relacional para la base de datos de la aplicación es posible empezar a ir generando el esqueleto de la aplicación mediante ***Scaffold***.

CREAR EL ESQUELETO DE LA APLICACIÓN

Crear un proyecto en Ruby on Rails

Antes de poder ejecutar ***Scaffold*** o cualquier funcionalidad implementada en Rails, es preciso especificar que se va a crear un proyecto que va a emplear esta tecnología. Para ello es indispensable tener instalados *Ruby y Rails* así como todos los paquetes que están relacionados con éste último y que por defecto suelen instalarse con él. Este aspecto será tratado con mayor profundidad en el manual de instalación de este proyecto.

Una vez que los paquetes mencionados se hallan instalados correctamente, será posible ejecutar el comando ***rails***, el cual tiene como cometido crear un proyecto que emplee *Ruby on Rails*, con su estructura de directorios predeterminada y una serie de scripts o utilidades que tienen como finalidad facilitar la tarea del desarrollador, entre los cuales se encuentra la mencionada ***Scaffold***.

Por tanto, el proyecto se creará de la siguiente forma:

```
>> rails answerIT
```

dando como resultado la siguiente estructura de directorios:

```
README   app      db       lib      public   test     vendor
Rakefile config   doc      log      script   tmp
```

De los cuales a continuación se explica brevemente su función:

- **app:** Contiene los componentes (modelos, vistas y controladores) de la aplicación. Es donde la mayor parte del proceso de implementación tendrá lugar.
- **config:** Contiene código relativo a la configuración de la aplicación, siendo los aspectos más destacados de esto la configuración de la base de datos (la cual se lleva a cabo a par-

tir del fichero *database.yml*) y la de las rutas y los recursos de la aplicación (fichero *routes.rb).*

- **db:** En este directorio se encuentran las clases que regirán el funcionamiento de la base de datos a nivel físico. Es decir, las clases de este directorio serán las responsables de la creación de tablas, de su eliminación, de la inclusión o eliminación de campos en ellas o de llevar a cabo migraciones en la base de datos de la aplicación fácilmente en caso de que resulten necesarias.

- **doc:** Al igual que Java posee *Javadoc, Ruby* posee *Rubydoc*, una interfaz a partir de la cual es posible generar automáticamente documentación sobre la aplicación a partir de los comentarios recogidos en el código y que se almacenarán en este directorio.

- **lib:** Donde se almacenan las librerías adicionales que sean utilizadas por el proyecto.

- **log:** Contiene las trazas de ejecución, que resultan especialmente útiles para tareas de depuración

- **public:** Donde se almacenarán recursos internos de la aplicación como imágenes, hojas de estilos, javascripts, páginas html, etc.

- **script:** Contiene diversos scripts que facilitarán sin duda la tarea del desarrollo en Rails, y que van desde un servidor ligero que alojará la aplicación web y desde el cual podrán verse sus evoluciones en todo momento (script/server) a la generación de código (script/generate).

- **test:** Donde se almacenarán las correspondientes pruebas unitarias llevadas a cabo sobre el código

- **tmp:** Dedicado al almacenamiento de aquellos archivos temporales que puedan estar relacionados con el funcionamiento de la aplicación.

- **vendor:** Para aquellas librerías externas que extiendan el funcionamiento básico de Rails.

Además de estos directorios, destaca un fichero, **Rakefile,** cuya finalidad es similar al Makefile de GNU y donde se podrán configurar los aspectos orientados a la construcción de la aplicación, su distribución en paquetes o los tests a realizar. Esta configuración será utilizada por el comando ***rake*** (hecho a similitud del ***make*** de GNU).

Cuando se genera un proyecto en Rails, y siguiendo uno de los principios de este framework, *Convention over Configuration*, se genera, además de estos directorios, una serie de ficheros en ellos con el fin de establecer ya una configuración por defecto basado en los estándares de programación web, discurriendo casi toda la totalidad de la fase de desarrollo dentro del directorio **app**. Sin embargo, resulta conveniente conocer esta información ya que en ocasiones alguna de estas convenciones establecidas como configuración debe de ser modificada.

Una vez ya se tiene el proyecto y su estructura de directorios creada, es posible empezar a disfrutar de las distintas utilidades que Rails ofrece.

Scaffolding

Para crear el esqueleto de la aplicación web mediante ***Scaffold*** hay que invocar esta función dentro del script *generate* que cualquier instalación de Rails posee de la siguiente manera:

```
>>script/generate scaffold <nombre_tabla> <campo1>:<tipo> <campo2><tipo>...
```

De esta forma se generarán las distintas tablas que se han extraído del diseño de la base de datos:

```
>> script/generate scaffold Pregunta enunciado:string tema:text respuestaA:string, respuestaB:string respuestaC:string respuestaD:string correcta:string feedback:string usuario:references

>> script/generate scaffold Usuario login:string password:string nombre:string apellidos:string email:string rol:string url_foto:string

>> script/generate scaffold Tema titulo:string descripcion:text

>> script/generate scaffold Valoracion pregunta:references usuario:references puntuacion:integer feedback:text
```

Por cada elemento que representa una tabla en la base de datos, ***Scaffold*** genera:

- Una clase para la creación/migración de la tabla en la base de datos (en **db/migrate**).
- Una clase para el modelo (en **app/models**).
- Una clase para el controlador que implementa las funciones de crear, eliminar y actualizar elementos en la tabla, además de otra para listar todos los elementos de la tabla (en **app/controllers**).
- Cuatro vistas para las cuatro funciones mencionadas del controlador (en **app/views/*<nombre_tabla>***)
- Un estilo para las vistas (**app/views/layouts/*<nombre_tabla>*.html.erb**)
- Una clase de test para la implementación de pruebas unitarias.

En definitiva, que tras ejecutar ***Scaffold*** para los elementos de la aplicación que hasta ahora han sido identificados, la aplicación queda tal y como se muestra en el diagrama de clases de la siguiente página.

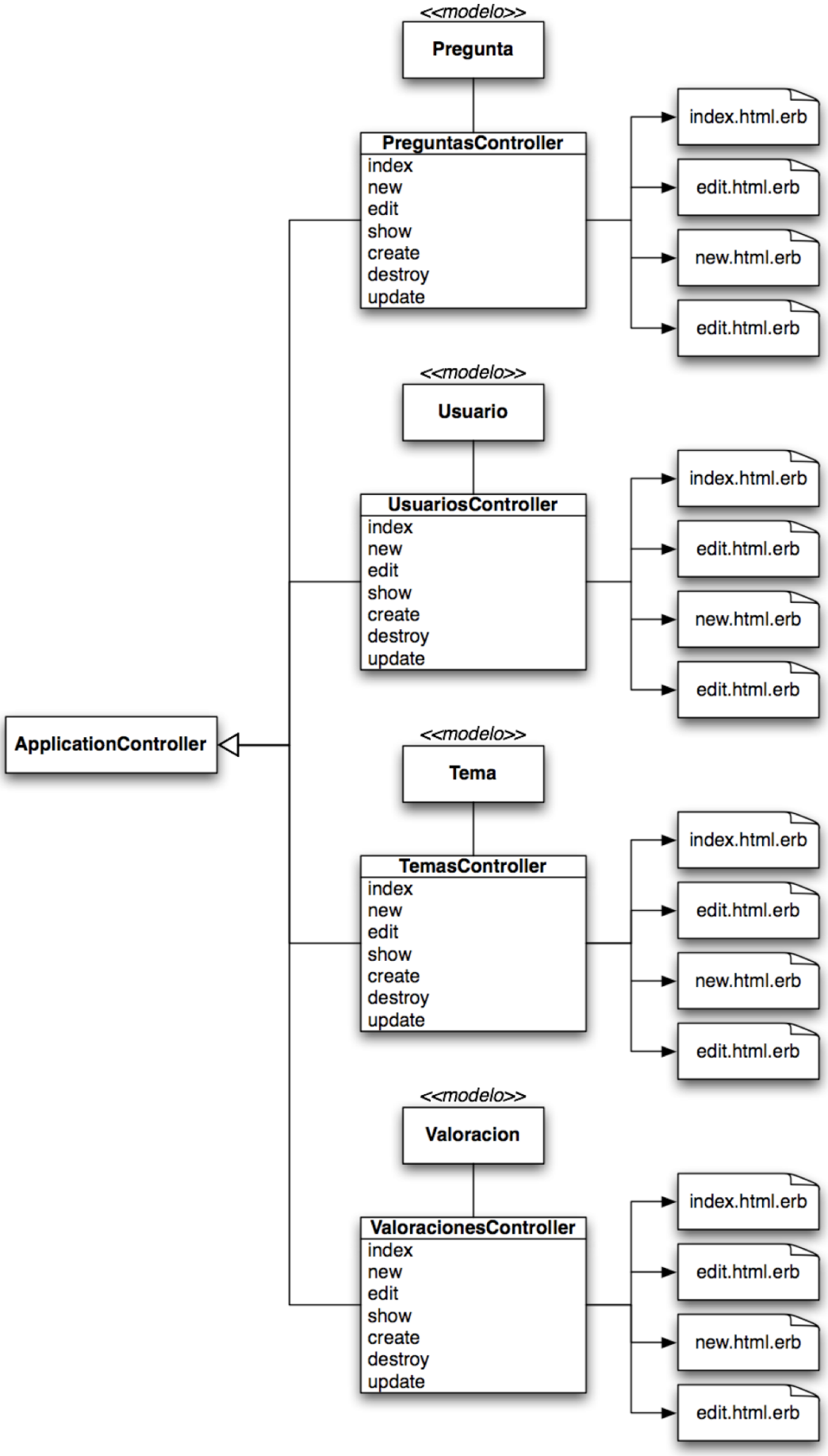


Figura 19: Diagrama de clases tras *scaffolding*

5. IMPLEMENTACIÓN

Como se ha especificado en el capítulo anterior, ***Scaffold*** genera un esqueleto funcional para la aplicación web que se va a desarrollar, generando una interfaz web que permite llevar a cabo las operaciones de creación, obtención, actualización y eliminación sobre cualquier elemento de las tablas que se han especificado. Sin embargo, estas funcionalidades resultan más que insuficientes para cumplir con los requisitos de la aplicación que se pretende diseñar.

La fase de implementación partirá del esqueleto obtenido a partir de la ejecución de ***Scaffold*** y terminará cuando una aplicación web funcional que cumpla con todos los casos de uso expuestos. Dentro de este largo proceso se destacarán una serie de hitos como los puntos más importantes dentro de él, y el desarrollo de los mismos será lo que se comentará en este capítulo.

Estos hitos se referirán a la construcción de una serie de subsistemas que serán decisivos para que la aplicación final cumpla con los requisitos especificados en el capítulo 3, o bien que ayuden a mejorar la usabilidad de la aplicación, ofreciendo de esta forma nuevas herramientas al usuario para poder realizar sus operaciones más cotidianas. Los mencionados hitos serán los siguientes:

- Trasladar completamente el modelo relacional de la base de datos realizado en la fase de diseño a la aplicación.
- Construcción de un subsistema de gestión de usuarios y de sesiones.
- Construcción de un entorno de configuración de la aplicación.
- Construcción de un subsistema que permita responder y valorar una pregunta.
- Construcción de un subsistema de ordenación dinámica por criterio de tablas.
- Construcción de un subsistema de paginación de tablas
- Construcción de un subsistema de generación dinámica de estadísticas.
- Diferenciar los roles de la aplicación, habilitando y restringiendo accesos y visibilidades por roles.
- Tratamiento y recuperación de errores

TRASLADO DEL MODELO RELACIONAL

Scaffold genera, por cada tabla por la cual se ejecuta, un esquema de migración ubicado en el directorio **db/migrate** con el siguiente nombre: ***<fecha y hora de creación>*****create*****<nombre de la tabla>*****.rb**, en el cual se contendrán los métodos necesarios para la creación de la tabla con todos sus campos y su eliminación. Además de este esquema de migración se creará una clase para el modelo (inicialmente vacía), el controlador que implementará las funciones de creación, recuperación, actualización y eliminación, y las cuatro vistas para cada una de estas funciones.

Esto tiene como consecuencia que las tablas no tienen de momento ninguna relación entre sí, y que por tanto resultará necesario trasladar las relaciones entre tablas existentes en el modelo relacional a la aplicación.

En Rails, las relaciones entre tablas se expresan en las clases correspondientes al modelo de cada una de las tablas mediante sencillas e intuitivas instrucciones como ***has_one***, ***belongs***, ***has_many***, etc. A continuación se explica con detalle como se implementaron las relaciones recogidas en el capítulo anterior en la aplicación:

Relación *"Pregunta pertenece a Tema" (N:1)*

Esta relación expresa que una pregunta ha de estar encuadrada en un solo tema, y que un tema puede contener múltiples preguntas. En código Rails se expresará de la siguiente forma.

Dentro de la carpeta **app/models** y en la clase **Pregunta**, deberá añadirse

```
belongs_to :tema
```

mientras que en la clase **Tema** de la misma carpeta habrá que añadir:

```
has_many :preguntas
```

Relación *"Usuario publica Pregunta" (1:N)*

En este caso se establece que una pregunta sólo puede ser publicada por un usuario y que un usuario puede publicar n preguntas. Por tanto y al igual que en el caso anterior:

Dentro de la clase **Usuario** se añade

```
has_many :preguntas
```

y dentro de la clase **Pregunta**

```
belongs_to :usuario
```

Relación *"Usuario valora Pregunta" (M:N)*

Esta es sin duda la relación más compleja. Además en la fase de diseño, el hecho de que las valoraciones además poseyeran atributos propios (puntuación, comentarios), motivó que este caso de relación M:N se resolviera mediante la creación de una nueva tabla llamada **Valoraciones**, luego cabe decir que la relación M:N entre preguntas y usuarios en este caso se expresa a través de la tabla **Valoraciones**.

La implementación en Rails de este caso resulta ser igual de sencilla e intuitiva que las anteriores:

En la clase Pregunta se añadirá:

```
has_many :valoraciones

has_many :usuarios, :through => :valoraciones
```

en la clase Usuario:

```
has_many :valoraciones

has_many :preguntas, :through => :valoraciones
```

y finalmente, en la clase Valoracion

```
belongs_to :pregunta

belongs_to :usuario
```

Eliminación de Registros

Cuando existen relaciones relaciones entre las distintas tablas que componen una base de datos, aparece también un problema a plantearse a la hora de eliminar registros. ¿Qué sucede con los registros que se hallen relacionados con el registro que nos disponemos a eliminar?

Este problema no es trivial, puesto que de no tratarse, puede dar lugar elementos, que faltos de un registro que les servía como referencia, empiecen a causar problemas (frecuentemente excepciones provocadas por referencias a un elemento nulo) en ciertas acciones de la aplicación.

Para evitar problemas de este tipo en la aplicación ha diseñar, se ha optado por marcar unas políticas en la eliminación de registros que irá, en gran medida, orientada a evitar la aparición de este fenómeno cuando algún registro del que sean dependientes otros registros en la aplicación sea eliminado. Dichas políticas serán las siguientes:

- La eliminación de un **tema** involucra la eliminación de las preguntas adscritas a él en cascada.

- La eliminación de un **usuario** involucra la eliminación de las preguntas que haya realizado en cascada así como sus valoraciones.

- La eliminación de una **pregunta** involucra la eliminación de todas aquellas valoraciones que se hayan realizado sobre ella.

Este tipo de políticas se establecen en *Rails* en las instrucciones ***has_many*** de las clases correspondientes al modelo de cada uno de los módulos de la aplicación añadiendo lo siguiente:

```
:dependent => :destroy
```

Por ello, las relaciones que anteriormente se habían declarado quedarían modificadas de la siguiente forma:

En la clase Tema:

```
has_many :preguntas, :dependent => :destroy
```

En la clase Usuario:

```
has_many :valoraciones, :dependent => :destroy
has_many :preguntas, :through => :valoraciones
has_many :preguntas, :dependent => :destroy
```

En la clase Pregunta:

```
has_many :valoraciones, :dependent => :destroy
has_many :usuarios, :through => :valoraciones
```

GESTIÓN DE USUARIOS Y SESIONES

En los casos de uso descritos en el capítulo 3 puede verse que uno de ellos concernía al acceso a la aplicación y que todos los demás tienen como precondición el haber iniciado una sesión previamente en la aplicación para ser ejecutados. Por tanto, el posibilitar a los usuarios de la aplicación el poder autenticarse e iniciar una sesión en ellas supone uno de los requisitos fundamentales del desarrollo de la aplicación.

Creación de un Esquema de Autenticación

Cómo la aplicación a desarrollar no es de carácter crítico, no resultará necesaria la implementación de un esquema de autenticación demasiado complicado, por tanto, el esquema tan abundante en la red en foros, redes sociales, etc. de utilizar un identificador único de usuario y una palabra clave o contraseña será el empleado también en la aplicación a desarrollar.

Este hecho ya se había previsto cuando se diseñó para la base de datos la tabla correspondiente a usuarios, que como puede recordarse, incluía los siguientes campos:

Usuario (id, login, password, nombre, apellidos, email, rol, url_foto)

Según ese esquema pensado entonces, al ser registrado un usuario se le almacenarían también en la base de datos un identificador de usuario (login) y una contraseña (password) con los que podría acceder al sistema. Cada vez que un usuario deseara entrar se contrastaría la información que introdujese en la pantalla de acceso con la almacenada en la base de datos, y de ocurrir esta comprobación con éxito, el usuario podría acceder a la aplicación y al resto de operaciones que ésta ofrece.

Sin embargo, la política de almacenar una información que se supone secreta como una contraseña en claro en la base de datos no resulta para nada segura, aún en aplicaciones poco críticas como la que se está desarrollando. Una simple consulta a la base de datos sería más que suficiente para poder obtener la contraseña de cualquier usuario y, por tanto, para poder iniciar una sesión suplantando a cualquier usuario.

En conclusión, la primera cuestión en la que pensar es un método por el cual la contraseña de un usuario pueda ser almacenada para ser contrastada cada vez que este desee acceder al sistema pero que evite que dicha contraseña pueda ser leída en claro en cualquier consulta realizada a la base de datos.

La respuesta a esta cuestión está en la criptografía, y en concreto en el uso de **funciones resumen o funciones hash.** Las funciones hash son funciones unidireccionales que convierten una cadena de datos de cualquier longitud en otra cadena de datos completamente diferente y de una longitud predeterminada, características que las hacen ideales para resolver el problema que se está tratando. El hecho de que una función hash sea unidireccional, implica que no es posible volver a obtener la cadena de datos inicial a partir de su valor hash, lo que hace de este valor un descriptor ideal de una contraseña para ser almacenada en una base de datos evitando así el almacenar la contraseña en claro. Bajo esta condición, cada vez que un usuario deseara iniciar sesión en el sistema, se computaría el valor hash de la contraseña que haya introducido y se contrastaría éste con el disponible en la base de datos. La conclusión es que el mismo proceso proceso de autenticación podría hacerse de forma prácticamente idéntica a la que fue mencionada antes sin necesidad de almacenar información secreta de los usuarios en la base de datos.

Para realizar este cambio serán necesarios una serie de cambios en el código inicial generado por ***Scaffold*** que serán descritos a continuación:

Lo primero de todo será modificar la tabla Usuarios previamente diseñada:

Usuario (id, login, nombre, apellidos, email, rol, url_foto, hashed_password, salt)

Como puede verse, además del campo correspondiente a la función hash de la contraseña, **hashed password**, se ha incluido un nuevo campo llamado **salt**. Este nuevo campo se refiere a una extensión de caracteres aleatoria que se añade a la contraseña antes de calcular su función hash con un doble fin, el de evitar que contraseñas idénticas generen valores hash idénticos y el de prevenir ataques mediante *tablas rainbow* (tablas que contienen los valores hash utilizando algunas de las funciones más conocidas de un gran número de posibles contraseñas en claro).

Para llevar a cabo esta modificación en lo que hay construido de aplicación, en primer lugar habrá que acceder al fichero donde está guardado el esquema de la tabla Usuarios, en el directorio **db/migrate** y en el fichero que contiene **"create_usuarios.rb"** (los dígitos que suelen aparecer antes en estos ficheros son referidos a la fecha de creación).

En ella se ve lo siguiente:

```
class CreateUsuarios < ActiveRecord::Migration
  def self.up
    create_table :usuarios do |t|
      t.string :login
      t.string :nombre
      t.string :apellidos
      t.string :email
      t.string :rol
      t.string :url_foto
      t.string :password

      t.timestamps
    end
  end

  def self.down
    drop_table :usuarios
  end
end
```

Cuando se pretende cambiar el esquema de una tabla en la base de datos, las siguientes situaciones pueden darse:

- Que el script de migración de la base de datos no se haya ejecutado, lo que significa que las tablas ni se han creado en la base de datos.

- Que el script de migración se haya ejecutado pero que los datos que se hayan ido almacenando no resulten relevantes, no teniendo entonces problema en poder eliminar todo el contenido de la base de datos pudiéndola crear de nuevo desde cero según el nuevo esquema.

- Que el script de migración se haya ejecutado y que los datos disponibles en ella resulten relevantes y no se desee que se pierden, además de poder cambiar el esquema de una o varias tablas.

En el primer caso, bastará con modificar los ficheros **create*<nombre_tabla>*.rb** y ejecutar el script de migración de la siguiente forma

```
>> rake db:migrate
```

En el segundo caso, lo más sencillo es reiniciar la base de datos y volver a construirla, esto se consigue en primer lugar llevando a la base de datos a su versión 0 ejecutando el script de migración de la siguiente forma:

```
>> rake db:migrate VERSION=0
```

A continuación se realizan los cambios deseados en los ficheros **create*<nombre_tabla>*.rb** y se ejecuta de nuevo el script de migración

```
>> rake db:migrate
```

Finalmente, en el tercer caso, el más frecuente en casos de modificación o actualización de una aplicación que ya esté funcionando, la solución pasa por la creación de una nueva clase dentro del directorio **db/migrate** que herede de la clase **ActiveRecord::Migration** y especificar dentro del método **self_up** los cambios que se desean hacer (normalmente relacionados con la adición o eliminación de algún campo). Tras finalizar la clase correspondiente a los cambios se ha de volver a ejecutar el script de migración para hacerlos efectivos:

```
>> rake db:migrate
```

En el caso que se está tratando, aún no se había ejecutado el script de migración, luego bastará con eliminar dentro del método **self_up** de la clase **create_usuarios.rb** la línea

```
t.string :password
```

añadiendo las dos siguientes:

```
t.string :hashed_password

t.string :salt
```

y a continuación se ejecutará el script de migración:

```
>> rake db:migrate
```

Con esto finalizan los cambios dentro del esquema de migración y, tras ejecutar el script de migración, el diseño que se había hecho para cada una de las tablas queda copiado en la base de datos de la aplicación.

El siguiente paso será modificar el modelo para la tabla **Usuarios**, donde el nuevo esquema de autenticación empezará a tomar forma.

En el modelo de la tabla **Usuarios** es donde se tomará la contraseña introducida por un usuario al acceder a la aplicación, se extraerá el resultado de la función hash de esta y se contrastará con el valor hash de la contraseña disponible para ese usuario en la base de datos.

Para empezar a realizar estas operaciones, lo primero será establecer e importar la función hash que se va a utilizar. Dentro del lenguaje *Ruby* existe una librería llamada ***digest*** encargada del cómputo de funciones hash, implementando en concreto las funciones MD5, SHA1 y SHA2.

El primer paso por tanto será importar esta librería en el modelo de la tabla **Usuarios (app/models/usuario.rb)**.

```
require 'digest/sha1'
```

Una vez hecho esto resultará sencillo encriptar una contraseña utilizando la función escogida:

```
def self.encriptar(pass, salt)
  Digest::SHA1.hexdigest(pass+salt)
end


def password=(pass)
  @password=pass
  self.salt = Usuario.generarCadenaAleatoria(10) if !self.salt?
  self.hashed_password = Usuario.encriptar(@password, self.salt)
end
```

y autenticar un usuario aplicando la función hash a la contraseña con la que ha intentado acceder el sistema y contrastándola con el código hash disponible en la base de datos:

```
def self.autenticar(login, pass)
  u=find(:first, :conditions=>["login = ?", login])
  return nil if u.nil?
  return u if Usuario.encriptar(pass, u.salt)==u.hashed_password
```

```
  nil

end
```

Creación de un Esquema de Registro

Para que el esquema de autenticación anteriormente descrito pueda funcionar, es preciso que los usuarios se hallen inicialmente registrados en el sistema.

Como answerIT está concebida como una herramienta a ser utilizada dentro del contexto de una clase o un curso, resulta obvio que el registro a la misma no será abierto, sino que se encontrará restringido a los profesores y alumnos que formen parte del curso para el cual se cree.

Por ello, y tal y como está reflejado en los casos de uso en el capítulo 3, serán los profesores los encargados de registrar en la aplicación a los alumnos o a los otros profesores que formen parte de la asignatura o el curso para el cual se haya habilitado la aplicación.

No obstante, el hecho de que los profesores sean responsables del registro de los alumnos abre dos problemas que será preciso resolver para poder implementar el esquema de registro mencionado.

- ¿Cómo sabe un usuario que ha sido registrado en la aplicación?
- Si un usuario necesita una contraseña para entrar pero el que se registra no es él, ¿cómo puede acceder a la aplicación tras el registro?

Es preciso, por tanto, establecer un medio de comunicación entre la aplicación y los usuarios que han sido registrados para que a estos últimos se les notifique su registro y el cómo pueden acceder a la aplicación.

El medio que resultará ideal para la realización de esta tarea será el correo electrónico. En las universidades modernas, los alumnos disponen de un correo electrónico propio y la mayoría de los profesores suelen disponer de listados sobre sus alumnos en los cuales es posible encontrar sus direcciones de correo electrónico de la universidad.

Por tanto, y en estas circunstancias, el esquema de registro tendría estas dos fases:

1. Un profesor con perfil creado en la aplicación registra a un nuevo usuario
2. El nuevo usuario registrado recibe un e-mail con sus datos de acceso (identificador y contraseña).

Generación de contraseñas aleatorias

Por razones de seguridad, es muy preferible que las contraseñas que les sean enviadas a cada uno de los distintos usuarios sean distintas entre si, por ello resultará necesario que las contraseñas que se generen por cada nuevo usuario sean aleatorias. Para este fin puede utilizarse también el método de generación de cadenas de caracteres aleatorias que se emplea para crear las extensiones aleatorias **salt** de cada perfil. Para habilitar esta nueva función será preciso escribir las siguientes líneas de código en la clase correspondiente al modelo para los **usuarios (app/models/usuario.rb)**

```
def registrar
   pass = Usuario.generarCadenaAleatoria(8)
   self.password = self.password_confirmation = pass
   self.save
end
```

Al invocar a este método que se ha creado, se le asignará a un usuario una contraseña aleatoria de 8 caracteres, y de paso se compatibilizará este método de registro con el más tradicional según el cual el usuario al inscribirse ha de introducir la que será su contraseña y confirmarla en un formulario. Este registro tradicional se llevará a cabo únicamente en la fase de puesta a punto y configuración de la aplicación para dar de alta al primer usuario de la misma, que será el profesor encargado de realizar esta configuración inicial.

Una vez ya que el profesor puede ya registrar a un nuevo usuario y generar una contraseña aleatoria para él, queda pendiente la implementación del sistema de notificaciones por correo electrónico, que será lo que a continuación se explique. El cómo únicamente los usuarios de rol **Profesor** podrán ejecutar esta función se explicará en el apartado *g* de este mismo capítulo, “Diferenciación por roles”.

Notificaciones por correo electrónico

En la instalación por defecto de *Rails* existe una librería llamada ***ActionMailer***, cuya función es la de implementar el envío desde el framework de correos electrónicos de una manera sencilla. Con esta base, crear un *mailer* para una aplicación *Rails* resulta bastante sencillo, pues bastará con introducir la siguiente instrucción en la línea de comandos dentro del directorio de la aplicación:

```
script/generate mailer <clase Mailer> <métodos mailer>*
```

En esta instrucción, *clase Mailer* será una clase que se creará junto al resto de modelos de la aplicación en la ruta **app/models** que heredará de la clase principal de la librería ***ActionMailer*** y donde se codificarán las distintas plantillas de las distintas notificaciones que se deseen enviar por correo electrónico. Cada una de las notificaciones que sea implementada dentro de esta clase tendrá su correspondiente método dentro de la misma, y podrán ser declaradas

cuando se cree el *mailer* (*métodos mailer*), o bien a mano una vez el *mailer* haya sido creado. En el primer caso, Rails creará una vista para cada uno de los métodos del *mailer* creado, en el segundo, será preciso crear el fichero de dicha vista a mano.

Una vez ya se ha conocido con detalle como crear un *mailer* en *Rails,* se creará el que es necesario para la aplicación:

```
script/generate mailer Registro envio_password
```

Una vez hecho esto solo queda editar el método correspondiente (**envio_password**) de la clase responsable del envío de los correos electrónicos (**Registro**) y su vista para dar a la notificación el aspecto deseado. A continuación se muestra cómo se ha hecho para **envio_password**:

En **app/models/registro.rb**

```
class Registro < ActionMailer::Base

  def envio_password(destinatario, login, password, sent_at = Time.now)

    subject    'AnswerIT - Registro en la aplicación'

    recipients destinatario

    from       'AnswerIT'

    sent_on    sent_at

    body       :username => login, :pass => password, :url => inicio

  end

end
```

En **app/views/registro/envio_password.erb**

```
Se ha creado una cuenta para usted en la aplicación AnswerIT con los siguientes datos

Usuario: <%= @username %>
Contraseña: <%= @pass %>

Puede acceder a AnswerIT en la siguiente dirección: <%= @url %>


¡Saludos!
```

Con esto se ha personalizado la notificación a enviar, pero la librería *Mailer* no ha sido aún configurada, para ello será necesario añadir las siguientes líneas al fichero **config/environment.rb**

```
ActionMailer::Base.smtp_settings = {
```

```
    :tls => true,
    :address => "<dirección servidor correo>",
    :port => "puerto",
    :authentication => :plain,
    :user_name => "<usuario>"
    :password => "<password>"
}
```

La línea en la que puede leerse **tls => true**, no corresponde propiamente a la librería ***ActionMailer*** de *Rails*, sino a un plugin llamado ***action_mailer_optional_tls*** que extiende la funcionalidad de ***ActionMailer*** a aquellos servidores de correo que corran bajo SSL, el cual se ha optado por añadir a la aplicación y que se encuentra almacenado en el directorio **plugins** de la misma.

Sesiones

Un elemento imprescindible que va unido a todo esquema de autenticación es la gestión de sesiones. El uso de sesiones permitirá registrar los detalles que se deseen de la actividad de un determinado usuario dentro de la aplicación. En la aplicación desarrollada, la política de gestión de sesiones a utilizar será la que *Rails* emplea por defecto, es decir, el manejo de sesiones mediante *cookies*. La sesión correspondiente a un determinado usuario se identificará por tanto mediante una cadena de caracteres aleatoria que el navegador almacenará de forma temporal en el ordenador del usuario una vez éste se haya autenticado ante la aplicación.

ENTORNO DE CONFIGURACIÓN DE LA APLICACIÓN

La gestión de usuarios descrita en el apartado anterior ha dejado dos puntos pendientes que deben de ser solucionados:

- Los usuarios de la aplicación son inscritos en esta únicamente por usuarios de rol profesor, sin embargo, ¿cómo se inscribe en ella el primer profesor que desee utilizarla? Al no existir ningún usuario inscrito no existirá ningún profesor que pueda inscribirle y por tanto ningún otro usuario podría ser inscrito.

- Cuando un usuario es inscrito en la aplicación recibe un correo electrónico. Dicho correo electrónico se envía desde un servidor SMTP cuyos datos, según se ha mencionado, forman parte del fichero **config/environment.rb**, el fichero de configuración de toda la aplicación *Rails*. ¿Existe la posibilidad de evitar que cualquier usuario que quiera instalar la aplicación se vea en la obligación de manipular un fichero sensible, como es uno de configuración, en un lenguaje que quizás no conozca?

Ambos problemas se intentarán resolver en este apartado.

Inscripción del primer usuario

Lo más corriente en la aplicación que está siendo desarrollada es que el primer usuario de la misma sea un profesor que haya decidido usarla y por lo tanto haya tomado la determinación de instalarla y desplegarla en un servidor.

Una vez llegado a este punto, la aplicación debería proveer al usuario de algún tipo de interfaz por la cual este pueda registrarse en la misma, y de esta forma, empezar a utilizarla.

Se ha optado por tanto por crear una página con un formulario dentro de la aplicación para este caso, en la cual el usuario podrá introducir sus datos (nombre de usuario, contraseña, nombre, apellidos, e-mail y despacho) y que se comunicará con el controlador de usuarios para crear un nuevo usuario en la base de datos.

Una vez el usuario ha completado el formulario sin errores estará inscrito y podrá disfrutar de todas las funciones de la aplicación acordes con su rol de profesor.

Configuración del servidor de correo electrónico

Para configurar el servidor de correo electrónico sin que el usuario tuviera que acceder al fichero de configuración **config/environment.rb**, la primera opción que se barajó fue la de aportar al usuario una interfaz a través de la cual pueda establecer los parámetros de su servidor de correo y que la lógica de esta interfaz manipulara el fichero de configuración en cuestión.

Sin embargo, esta solución resulta increíblemente compleja y peligrosa, pues se supone que la lógica de la interfaz debería generar código *Rails* dinámicamente con el riesgo de poder generar en alguna ejecución fallida fragmentos de código erróneo en el fichero **config/environment.rb**, lo que provoca que la aplicación no pueda ni siquiera ejecutarse.

Para evitar estos problemas, se buscó como solución alternativa el llevar la configuración del gestor de correo a un fichero fuera de **config/environment.rb**. *Rails* ofrece también la posibilidad de crear otros ficheros de configuración para cambiar detalles puntuales de la misma, sin embargo su uso tampoco soluciona los problemas anteriormente mencionados (también irían escritos en código *Ruby* que debería ser generado dinámicamente y un error en este código afectaría también de forma grave al funcionamiento de la aplicación).

La solución pasará por analizar más a fondo cómo *Rails* establece dichos parámetros de configuración, cómo se ha explicado en el apartado anterior, el servidor de correo queda configurado en la siguiente instrucción:

```
ActionMailer::Base.smtp_settings = {
    :tls => true,
    :address => "<dirección servidor correo>",
```

```
    :port => “puerto”,
    :authentication => :plain,
    :user_name => “<usuario>”
    :password => “<password>”
}
```

Puede verse como un objeto de tipo ActionMailer::Base.smtp_settings contiene todos y cada uno de los parámetros de configuración del servidor de correo. Por lo tanto, si el contenido de este objeto pudiera ser exportado a un fichero, y desde el fichero **config/environment.rb** se pudieran reconstruir las propiedades que han sido exportadas, el problema estaría solucionado, y este problema es justo el que resuelve la **serialización**.

La serialización consiste en la conversión de un objeto en una colección de bytes o a un lenguaje de marcado determinado (por ejemplo XML) ya sea para enviarlo a través de una red o para guardarlo temporalmente en disco y luego recuperarlo.

Esta estrategia es ya utilizada por *Rails* para la configuración de otros elementos del entorno de desarrollo, como sucede con la base de datos (fichero **database.yml**). Este fichero posee la extensión *yml*, que caracteriza a aquellos ficheros propios del lenguaje de serialización YAML.

YAML (YAML Ain’t Another Markup Language), es un lenguaje de serialización de datos legible por humanos y bastante sencillo cuyo fin es el de centrarse en los datos en vez de en el marcado de documentos. Es además un formato ampliamente utilizado por *Rails*, no solo para la configuración del entorno de desarrollo, sino también para la realización de pruebas, para las cuales pueden utilizarse objetos definidos mediante YAML situados en la carpeta **test/fixtures**. Debido por tanto a las librerías que *Ruby* y *Rails* disponen para la manipulación de este lenguaje de serialización, los parámetros de configuración del servidor de correo electrónico se emplazarán en un fichero YAML.

Para hacer efectivo dicho cambio será preciso que, primero, la lógica detrás del formulario en el cual el usuario especifica los detalles de su servidor de correo convierta dicha información a formato YAML, lo cual es muy sencillo e intuitivo, pues bastará que la información que se escriba en el fichero posea la siguiente estructura:

```
:user_name: <valor>
:password: <valor>
:address: <valor>
:port: <valor>
:authentication: <valor>
:tls: <valor>
```

Y finalmente, lograr que esta información sea interpretada dentro del fichero **/config/environment.rb**, lo que se consigue cambiando las líneas que contenían la información del servidor de correo por las siguientes:

```
mailer_config = File.open("#{RAILS_ROOT}/config/mailer.yml")
mailer_options = YAML.load(mailer_config)
ActionMailer::Base.smtp_settings = mailer_options
```

Funcionamiento conjunto del entorno de configuración

Una vez se han creado ya las dos páginas que permitirán configurar la aplicación, el siguiente paso es definir una ruta para ellas. Para ello se utilizará la raíz **setup**. Cuando dicha raíz sea invocada desde el navegador, de no existir ningún usuario de rol profesor en la aplicación, se llamará al formulario de inscripción del primer usuario, y de existir ya este usuario, la redirección se hará al formulario de configuración del servidor de correo.

Resulta altamente recomendable dejar la raíz **setup** como únicamente accesible localmente cuando se despliegue la aplicación, dada la alta sensibilidad de las páginas que alberga.

RESPUESTAS Y VALORACIONES A PREGUNTAS

Mediante el esqueleto de la aplicación que fue automáticamente generado por ***Scaffold***, cuando se recupera una pregunta de la base de datos (acción ***show***), automáticamente aparecen mostrados por pantalla todos los campos de la tabla **Preguntas** almacenada en la base de datos (id, enunciado, tema, respuestaA, respuestaB, respuestaC, respuestaD, correcta, feedback, usuario) sin permitir al usuario ninguna interacción con el contenido.

Para poder habilitar la opción de que un usuario pueda escoger entre las cuatro posibles respuestas de la pregunta, obtener tras esto la respuesta correcta y finalmente poder valorar la pregunta, resultará necesario modificar el código existente.

Elección entre Respuestas

Lo primero será permitir al usuario poder elegir entre las cuatro posibles respuestas a la pregunta.

Este problema viene dado principalmente por la forma en la que una pregunta se le es mostrada a un usuario, luego las modificaciones a elegir estarán en la vista de la aplicación, pudiendo dejar tanto el modelo como el controlador de Preguntas tal y como están.

Accediendo a la vista de la acción *show*, que es la encargada de recuperar y mostrar una pregunta seleccionada de la base de datos, se ve un fragmento de código sencillo en lenguaje *Ruby embebido* (HTML + sentencias *Ruby*). Este código se encarga únicamente de mostrar por pantalla todos los campos del registro de la tabla Preguntas que ha sido seleccionado.

En el código existente, se actuará de la siguiente forma:

- Se dejará tal cual la parte correspondiente a mostrar el **enunciado** de la pregunta.
- Se alterará levemente la correspondiente al **tema** para que se muestre el título del tema y no la referencia física al lugar de la base de datos donde se encuentra.
- Se eliminará la referencia al **autor de la pregunta**, pues es una información que es preferible que el alumno que va a responder la pregunta no conozca, ya que esto puede alterar su objetividad a la hora de valorar la pregunta.
- Se eliminarán igualmente las referencias a la **respuesta correcta** y al **feedback** de la pregunta, pues no interesan en este punto.
- Cada una de las **cuatro respuestas** se incluirán en un elemento *submit*. Este elemento consiste en un botón que al ser pulsado envía una cierta información a otro componente de la aplicación. Como solo es posible incluir un elemento *submit* por un formulario, será necesario implementar un formulario por respuesta. Cada uno de estos formularios enviará la letra correspondiente a la respuesta que tienen asociada.
- Finalmente el uso de algunas cabeceras y de CSS le darán a esta vista un aspecto más claro y elegante.

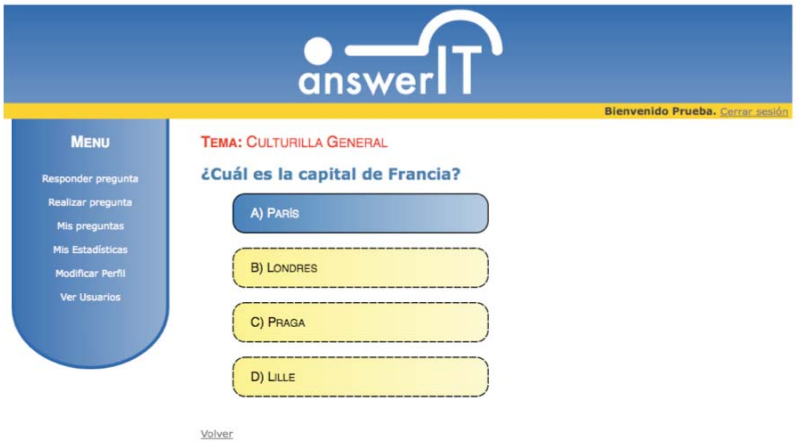


Figura 20: Captura de pantalla de la elección de respuestas

Al llegar a este punto, el usuario ya puede hacer clic en una de las respuestas y escogerla, lo siguiente será implementar la manera en la que la aplicación responderá a esta acción.

Verificación de la Respuesta

Tras responder, según los requisitos, se ha de comprobar si la respuesta del alumno ha sido correcta o no, informarle de la respuesta correcta en caso de que no lo sea, y mostrarle el feedback adicional a la pregunta que el autor haya dejado.

El primer objetivo es comprobar si la pregunta ha sido respondida correctamente. Para ello basta con vincular cada uno de los cuatro botones de envío que representan una respuesta con una acción en la cual se compruebe la veracidad de la respuesta. Como la ejecución de esta acción involucra la ejecución de una operación que se encuentra dentro del marco de la lógica de la aplicación, resultará necesario crear esta acción en el controlador de Preguntas.

La acción a crear tomará la respuesta dada y la contrastará con la respuesta correcta almacenada para la pregunta en la base de datos:

```
def verificar
  @pregunta = Pregunta.find(params[:id])
  @respuesta = params[:respuesta]
  if @respuesta==@pregunta.correcta
    @acierto = true
  else
    @acierto = false
  end
end
```

Una vez hecha la comprobación, deberá reflejarse su resultado en pantalla, para que el usuario conozca si ha respondido correctamente o no. Además, deberá aparecer también el feedback o conocimiento adicional a la pregunta introducida por el autor de la misma. Para hacer de esto algo vistoso y elegante se hará uso de JavaScript.

Al igual que en Rails existen plantillas de html con código *Ruby embebido* (*.rhtml o .html.erb)* es posible crear plantillas de JavaScript con código *Ruby embebido* (*.rjs)*. La idea será que en cuanto el usuario escoja una respuesta, se resalten inmediatamente la respuesta escogida y la respuesta correcta (en caso de que la respuesta escogida sea fallida) y aparezca un recuadro conteniendo el feedback del autor. Por tanto, se está hablando de una serie de elementos activados por un evento concreto, lo que encaja perfectamente con JavaScript.

Mediante JavaScript se reemplazará el código HTML correspondiente a cada una de las respuestas, cambiando la clase del elemento *submit*, para que su aspecto pueda ser controlado mediante CSS y desactivándolos, para conseguir que una vez respondida una pregunta, no pueda ser respondida de nuevo. Igualmente se dejará de ocultar el fragmento de código HTML que muestra el feedback del autor.

Figura 21: Captura de pantalla tras responder una pregunta

Valoraciones

Al mismo tiempo que al usuario se le muestra el resultado de su respuesta y el feedback añadido a la pregunta, ha de presentársele la posibilidad de valorar la pregunta con una puntuación entre 0 y 5 y de realizar un comentario a modo de aporte u opinión a la pregunta.

El formulario en el cual el usuario que ha respondido podrá realizar su valoración aparecerá bajo el feedback aportado por el autor de la pregunta. El botón de enviar de dicho formulario se enlazará con la acción *crear* del controlador correspondiente a las valoraciones, que se encargará de introducir una nueva valoración en la base de datos.

Figura 22: Captura de pantalla de una valoración

ORDENACIÓN DINÁMICA

Los siguientes supuestos no son difíciles de imaginar una vez que la aplicación haya sido desarrollada de acuerdo a sus requisitos, algunos usuarios se hayan registrado en ella y hayan empezado a dejar sus preguntas para ser respondidas:

- Un usuario puede estar interesado en encontrar una pregunta en concreto por su enunciado.
- Un usuario puede estar interesado en ver las preguntas separadas por temas.
- Un profesor puede estar interesado en ver las preguntas que publicó un usuario en concreto, o en visualizar todas las preguntas utilizando el usuario que las publicó como criterio de agrupación.
- Un profesor puede tener interés por conocer cuáles son las preguntas que mejor han sido valoradas o desear ordenar las preguntas de acuerdo a su valoración.

Todas las mencionadas son utilidades que, a medida que la aplicación fuese creciendo en volumen de preguntas, tomarían una importancia cada vez más decisiva, pues su influencia en el tiempo en el que tardaría un usuario en encontrar una pregunta que se ajustara a sus criterios sería cada vez mayor.

Por estos motivos, para las tablas de preguntas presentes en las opciones “Realizar Preguntas” (rol alumno) y “Estadísticas Preguntas” (rol profesor) se ha decidido implementar la opción de poder ordenarlas dinámicamente, ya sea en orden ascendente o descendente, por cada uno de los distintos campos que contengan.

Modificaciones en el Controlador

Para poder llevar a cabo esta funcionalidad, lo primero será modificar el controlador de Preguntas, que es donde se extraerán las preguntas de la base de datos que luego serán mostradas por pantalla. Estas preguntas, como cualquier elemento que se desee extraer de una base de datos en *Rails*, son recopiladas mediante el método *find*, cuya función es la de ejecutar consultas en la base de datos según los criterios especificados en sus parámetros y almacenar los resultados de esta consulta en un array. Para que estos resultados se almacenen ya ordenados en el array bastará con añadir el modificador **:order => <campo>** (u **:order => <campo> DESC** si lo que desea es ordenación descendente) como parámetro del método *find.*

Como el campo a partir del cual se ordenarán los registros vendrá dado por el usuario, y , por tanto, por la vista de la aplicación, se hará depender el modificador **:order** de una variable

donde se almacenará el campo escogido por el usuario en la interfaz para llevar a cabo la ordenación.

Todas estas modificaciones se llevarán a cabo en la acción de la interfaz que muestra todas las preguntas, esto es, la acción *index*, y por tanto en el método que lleva ese nombre en el controlador de Preguntas.

Una vez hechos los cambios necesarios en el controlador faltaría modificar la vista correspondiente a la acción *index* dentro de las vistas contenidas en el directorio *Preguntas* para permitir que el usuario pueda señalar un criterio de ordenación para la tabla de preguntas que se le muestra.

Modificaciones en la Vista

Lo primero que habrá que hacer será escoger el método por el cual el usuario podrá elegir el campo según el cual pretende ordenar los campos de la tabla que se le muestran. Una solución bastante recurrida y que, además, resulta la más intuitiva en el caso que se da, es la de crear un vínculo en cada una de las cabeceras de las tablas. Las cabeceras de las tablas no son otra cosa que los distintos campos que posee la tabla, por tanto, resulta bastante intuitivo que al hacer clic en ellas, los registros se ordenen según el campo de la cabecera seleccionada.

Como la ordenación puede ser tanto de carácter ascendente como descendente, se establecerá que cada vez que el usuario haga clic por primera vez en una cabecera, el orden será el estimado por defecto para ella (ascendente en todos los casos excepto para las valoraciones, en las que será descendente para dar preponderancia a las valoraciones altas). Luego, cada vez que el usuario haga clic sobre la misma cabecera, el orden se invertirá.

Para poder implementar esto se emplearán los helpers habilitados para las preguntas. Los helpers contienen normalmente funciones que pueden ser invocadas desde cualquier lugar de la aplicación (ya sea modelo, vista o controlador). Su finalidad principalmente es la de proporcionar un lugar común en el cual puedan encontrarse aquellas funciones que contienen código que de otra forma se repetiría varias veces en distintos puntos de la aplicación. En definitiva, su existencia tiene sentido dentro de la premisa de *Rails* de *Don't Repeat Yourself*, que intenta proporcionar herramientas para que el programador repita fragmentos de código lo menos posible.

Por tanto, en el fichero **app/helpers/preguntas_helper** se creará una función en la cual se creará el enlace de la cabecera y que irá alternando el parámetro que indique el orden (ascendente o descendente) según las veces que este enlace sea pulsado.

Esta funcionalidad se podrá ejecutar tanto utilizando AJAX y evitando el volver a cargar la página; como sin utilizar esta técnica, recargándose entonces la página y utilizando los parámetros que le lleguen por un método de petición GET para ejecutarse.

PAGINACIÓN

Tratando el mismo supuesto que ya fue comentado en el apartado anterior, el de que la aplicación vaya llenándose de un número alto de preguntas planteadas por los usuarios, surge otro inconveniente relacionado con la usabilidad de la aplicación. A pesar de poder ordenar las preguntas por el criterio que deseara, el usuario debería enfrentarse a una larga lista de preguntas, lo que por un lado ralentizaría la carga de la página y, por otro, obligaría al usuario a hacer un uso excesivo de la barra de desplazamiento del navegador o de la rueda de su ratón.

La paginación es un recurso al que suelen acudir aquellas páginas que esperan recibir gran cantidad de contenidos para resolver problemas como el mencionado. Consiste en partir el contenido existente en una serie de páginas con un límite razonable de contenido, lo cual permite que al usuario se le muestre una cantidad no demasiado grande de contenido que no sobrecargue la página ni que resulte abrumador a la hora de buscar una entrada en concreto.

Figura 23: Ejemplo de paginación [www.google.com]

Además, la paginación permite el poder saltarse aquellas páginas que contengan entradas no relevantes a la hora de buscar una entrada de la tabla en concreto.

Las versiones anteriores a *Ruby on Rails* 2.0 tenían incluido ya un componente dedicado a la paginación, pero a partir de esta versión este componente desapareció dejando la paginación a manos de paquetes y plug-ins externos al nucleo de *Ruby on Rails*. Algunos de estos plug-ins son ***classic_pagination***, ***paginator*** *o* ***will_paginate***.

Cómo este proyecto está utilizando Rails 2.2.2, será necesario hacerse con uno de estos plug-ins para poder incluir esta funcionalidad. ***Will_paginate*** ha sido el plugin escogido para esta función.

Lo primero será añadir el repositorio donde se encuentra ***will_paginate*** al gestor de paquetes de *Ruby* (*Rubygems*)

```
>> gem sources -a http://gems.github.com (como superusuario)
```

y a continuación proceder a la instalación del paquete o “gema” (como se conocen a los paquetes gestionados por *Rubygems*).

```
>> gem install mislav-will_paginate (como superusuario)
```

Con estas dos instrucciones se habrá instalado el paquete ***will_paginate*** en el ordenador, y el siguiente paso será invocarlo desde la aplicación web que está siendo desarrollada. Para habilitarlo habrá que introducir las siguientes líneas en el fichero **config/environment.rb** dentro del bloque ***Rails::Initializer.run do |config|***

```
Rails::Initializer.run do |config|
...
  config.gem 'mislav-will_paginate',
    :lib => 'will_paginate',
    :source => 'http://gems.github.com'
...
end
```

Tras hacer esto, las librerías correspondientes al plugin ***will_paginate*** podrán ser usadas en la aplicación.

Para funcionar correctamente, ***will_paginate*** deberá ser invocado desde el controlador, donde se especificará el contenido que se quiere paginar, y desde la vista, donde se escogerá el lugar que se desea que ocupe el selector de páginas.

Para los controladores, ***will_paginate*** incorpora un método llamado **paginate*,*** cuya función es esencialmente la del método **find** ya existente en *Rails*, es decir, realizar una consulta determinada a la base de datos y devolver los resultados en un array, con el cambio de que el array devuelto correspondería únicamente al contenido de una página. Para poder hacer esto, el método **paginate** utiliza dos parámetros adicionales; **:page**, para el número de página solicitada, y **:per_page**, en el cual se indicarán el número de entradas que se desea que aparezcan por página.

A continuación se muestran algunos ejemplos extraídos del código de la aplicación del uso de **paginate** en el controlador:

```
@page_results = @preguntas.paginate(:page => params[:page], :per_page => 15)

@v = Valoracion.paginate :all, :include => :pregunta, :conditions => ["pre-
guntas.usuario_id = ?", params[:id]], :page => params[:page],  :per_page => 5

@alumnos  =  Usuario.paginate_all_by_rol("Alumno",  :page  =>  params[:page],
:per_page => 10)
```

Como puede verse, ***will_paginate*** admite los mismos modificadores que el método **find** original de *Rails* para permitir búsquedas por un campo concreto y además puede también actuar sobre arrays de objetos ya predefinidos sin interactuar con la base de datos.

Una vez efectuada la parte lógica del proceso de paginación, queda acudir a la vista para escoger el lugar que ocupará en la página el selector de páginas que permitirá al usuario moverse de una página a otra. Para esto bastará con incluir la siguiente instrucción en el lugar deseado de la página html.erb:

```
<%= will_paginate <variable> %>
```

Donde *<variable>* señala a la variable en la que se guardaron los resultados de la paginación, por tanto, para los ejemplos anteriores, se incluiría en las correspondientes vistas:

```
@page_results = @preguntas.paginate(:page => params[:page], :per_page => 15)

@v = Valoracion.paginate :all, :include => :pregunta, :conditions => ["preguntas.usuario_id = ?", params[:id]], :page => params[:page], :per_page => 5

@alumnos = Usuario.paginate_all_by_rol("Alumno", :page => params[:page], :per_page => 10)
```

Esta instrucción generará el siguiente selector en la página correspondiente:

« Previous 1 2 3 4 5 6 7 8 Next »

Figura 24: Paginador por defecto WillPaginate

Selector que puede personalizarse. Por ejemplo, ya que la aplicación se encuentra en lenguaje castellano, los indicadores de “Previous” y “Next” en inglés en el selector desentonan, sin embargo, esto es algo que por suerte se puede corregir incluyendo las siguientes líneas tras el bloque ***Rails::Initializer.run do |config|*** en el fichero ***config/environment.rb***:

```
WillPaginate::ViewHelpers.pagination_options[:prev_label] = '&laquo; Anterior'

WillPaginate::ViewHelpers.pagination_options[:next_label] = 'Siguiente &raquo;'
```

Además ***will_paginate*** provee al selector de un identificador HTML por el cual su aspecto puede resultar además personalizable mediante el uso de CSS, consiguiendo resultados como los siguientes, que son los que finalmente figurarán en la aplicación:

« Anterior 1 2 3 4 5 6 7 8 9 Siguiente »

« Anterior 1 2 3 4 5 6 7 8 Siguiente »

Figura 25: Paginadores de la aplicación

GENERACIÓN DE ESTADÍSTICAS

Los gráficos basados en estadísticas resultan de gran ayuda para tareas de extracción y análisis de datos, y en esta aplicación existe un rol en concreto, el de profesor, cuya principal tarea es ésta, la extracción de información y el análisis de datos basados en la actividad que los alumnos desarrollen dentro de la aplicación.

El rol que tomará el profesor lleva implícitamente a la creación de un entorno en el cual en análisis de datos sea una tarea cómoda y sencilla, y la inclusión de representaciones gráficas de los datos que permitan obtener una percepción rápida de la naturaleza de los datos sin duda ayudará en este cometido.

Como en el caso anterior, no existe en el paquete estándar de *Ruby on Rails* ninguna librería que permita la generación dinámica de gráficos; no obstante, para este caso existen también plugins que realizan este cometido como ***Scruffy*** o ***Gruff***.

Finalmente, el *plugin* que se ha escogido para la generación dinámica de estadísticas ha sido ***Gruff*** por su mayor documentación. Al igual que en el caso anterior, habrá que instalar el paquete correspondiente al *plugin* e importar sus librerías en la aplicación antes de empezar a utilizarlo.

Antes de pensar en instalar la gema correspondiente de ***Gruff***, es preciso comprobar la existencia del paquete **ImageMagick** en el ordenador. **ImageMagick** es un programa de conversión de gráficos vectoriales svg a otros formatos de imagen y es utilizada por ***Gruff*** para la conversión de sus gráficos que son inicialmente generados en formato svg. **ImageMagick** puede encontrarse en los repositorios de serie de las distribuciones Linux más conocidas. En caso de utilizarse otros sistemas operativos como Windows o Mac OSX, resulta también posible encontrar paquetes de instalación para estos sistemas operativos en la página web del producto http://www.imagemagick.org

Una vez se ha comprobado que **ImageMagick** está instalado, el siguiente paso es instalar la gema correspondiente a la interfaz entre **ImageMagick** y el lenguaje *Ruby*, **RMagick**:

```
>> gem install rmagick (como superusuario)
```

Tras esto podrá instalarse la gema correspondiente a ***Gruff:***

```
>> gem install gruff (como superusuario)
```

Y ya instalado *Gruff* importar sus librerías en el código de la aplicación, para ello habrá que acudir al fichero config/environment.rb e introducir la siguiente línea tras el bloque ***Rails::Initializer.run do |config|***

```
require 'gruff'
```

Una vez ya se ha llegado a este punto, *Gruff* puede empezar a utilizarse para generar gráficos dentro de la aplicación.

Para crear un gráfico, el primer paso será crear una instancia de tipo *Gruff*, especificar como subinstancia el tipo de gráfico que se desea y asignar una variable a esta instancia:

```
<variable> = Gruff::<tipo_grafico>.new(<tamaño_píxeles_del_gráfico>)
```

Para la aplicación solo se han utilizado gráficos de tarta (para la estadística de respuestas acertadas y erróneas) y de barras (para representar las puntuaciones otorgadas o recibidas en las valoraciones).

Una vez creado el gráfico, el siguiente paso será introducir los datos para el mismo, lo cual se consigue mediante el método **data** sobre la variable que anteriormente se ha escogido para instanciar el gráfico.

```
<variable>.data (<titulo dato>, <dato numérico>)
```

A continuación puede verse un ejemplo extraído de la aplicación sobre la creación de un gráfico de tarta en el cual se muestran la proporción de aciertos y de fallos ya sea para una pregunta, o para el conjunto de preguntas realizadas por un determinado usuario.

```
g = Gruff::Pie.new(350)
g.data ('ACIERTOS', @aciertos)
g.data ('FALLOS', @respuestas-@aciertos)
```

Una vez se han introducido los datos, faltaría únicamente generar la imagen correspondiente al gráfico y vincularla a la página de la vista en la que se pretende mostrar. Para esto, se creará una referencia a la imagen del gráfico utilizando el método *to_blob* que aporta la librería **RMagick**:

```
send_data(g.to_blob, :disposition => 'inline', :type => 'image/png',
:filename => "question-stats.png")
```

Y finalmente desde la vista correspondiente basta con invocar a la acción del controlador en la cual se crea el gráfico y su referencia para que el gráfico pueda ser mostrado por pantalla.

```
<%= image_tag(url_for :controller => 'preguntas', :action => 'estadisti-
cas_aciertos', :id => @pregunta.id)%>
```

Por último cabe mencionar que las librerías de ***Gruff*** poseen múltiples métodos relacionados con la personalización y la presentación de los gráficos a generar, pudiendo de esta manera adaptarse el modo en el que se muestran los gráficos al resto de la interfaz de la aplicación.

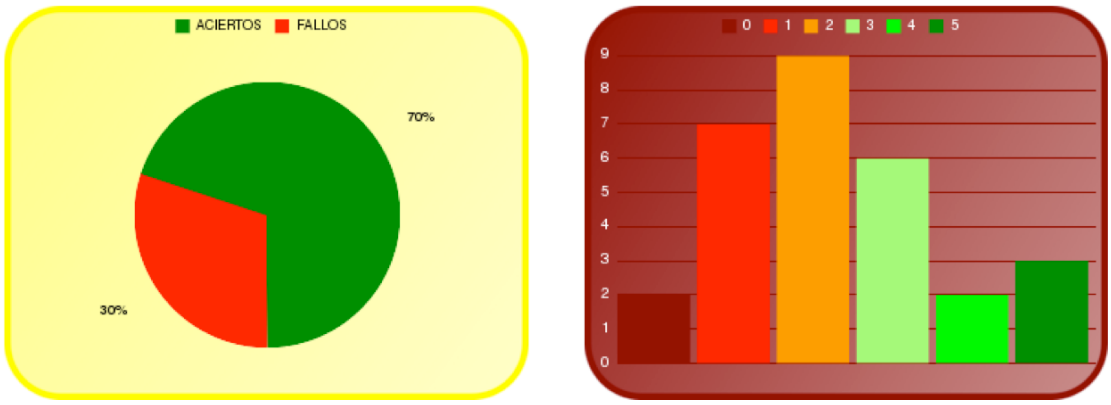


Figura 26: Ejemplos de los gráficos de estadísticas de la aplicación

DIFERENCIACIÓN POR ROLES

Otro aspecto importante en el proceso de implementación de la aplicación es el relativo a la diferenciación de roles.

Como se ha comentado en capítulos y apartados anteriores, formarán parte de la aplicación dos clases de usuarios, alumnos y profesores, y el uso que un usuario vaya a dar a la aplicación dependerá fundamentalmente de a cual de estas dos clases pertenezca; pues los alumnos tendrán como principal objetivo publicar, responder y valorar preguntas; mientras que el profesor actuará principalmente como un observador de la actividad de sus alumnos en el sistema, esperando obtener detalles sobre qué preguntas son las más valoradas, cómo se valoran sus alumnos entre sí, qué alumnos son los que publican las preguntas más interesantes a juicio de sus compañeros, etc.

Todo esto lleva, como ya se ha visto anteriormente, a que haya funcionalidades de la aplicación que correspondan exclusivamente a un rol o que haya otras funcionalidades que ambos roles compartan, pero que los usuarios de un rol determinado tengan mas restrcciones a la hora de ejecutarlas. Los detalles sobre estas funcionalidades ya han sido explicados con detalle anteriormente, y especialmente en el capítulo 3, por tanto este apartado se limitará a centrarse en cómo han sido tratadas las distintas vistas implementadas en la aplicación para adaptarse a las necesidades de uno u otro rol.

Las siguientes tablas contendrán las distintas vistas que forman parte de la aplicación, indicando el caso de uso con el cual estarán relacionadas, los roles que pueden ejecutarlas y aquellos detalles que resulten de especial interés respecto a cómo se mostrarán las distintas vistas.

Vista:	**views/acceso/login.html.erb**
Caso de Uso Relacionado:	Acceder a la aplicación
Roles:	Profesor, Alumno
Más Detalles:	

Tabla 13: Propiedades Vista "acceso/login"

Vista:	**views/preguntas/edit.html.erb**
Caso de Uso Relacionado:	Formular Preguntas
Roles:	Alumnos
Más Detalles:	El **alumno** edita una pregunta propia ya existente

Tabla 14: Propiedades Vista "preguntas/edit"

Vista:	**views/preguntas/estadisticas.html.erb**
Caso de Uso Relacionado:	Obtener Estadísticas Preguntas (Profesor) Obtener Estadísticas Personales (Alumno)
Roles:	Profesor, Alumno
Más Detalles:	El profesor puede obtener estadísticas de cualquier pregunta existente en la aplicación. El alumno solo puede obtener estadísticas de las preguntas formuladas por si mismo, y al mostrar las valoraciones ajenas a sus preguntas, el autor de las mismas no se mostrará.

Tabla 15: Propiedades Vista "preguntas/estadisticas"

Vista:	**views/preguntas/index.html.erb**
Caso de Uso Relacionado:	Responder Preguntas
Roles:	Alumno
Más Detalles:	El alumno verá listadas únicamente aquellas preguntas realizadas por sus compañeros que no haya contestado ya.

Tabla 16: Propiedades Vista "preguntas/index"

Vista:	**views/preguntas/mispreguntas.html.erb**
Caso de Uso Relacionado:	Obtener Estadísticas Personales
Roles:	Alumno
Más Detalles:	El alumno verá listadas únicamente sus propias preguntas junto a la valoración media que estas hayan recibido por parte de sus compañeros.

Tabla 17: Propiedades Vista "preguntas/mispreguntas"

Vista:	**views/preguntas/new.html.erb**
Caso de Uso Relacionado:	Formular Preguntas
Roles:	Alumno
Más Detalles:	El alumno formula una pregunta desde cero.

Tabla 18: Propiedades Vista "preguntas/new"

Vista:	**views/preguntas/show.html.erb**
Caso de Uso Relacionado:	Responder Preguntas
Roles:	Alumno
Más Detalles:	El alumno puede seleccionar una de las cuatro respuestas y responder así a las preguntas.

Tabla 19: Propiedades Vista "preguntas/show"

Vista:	**views/preguntas/verificar.js.rjs**
Caso de Uso Relacionado:	Responder Preguntas
Roles:	Alumno
Más Detalles:	Aporta los componentes *JavaScript* necesarios para verificar de forma dinámica la validez de la elección del alumno.

Tabla 20: Propiedades Vista "preguntas/verificar"

Vista:	**views/temas/edit.html.erb**
Caso de Uso Relacionado:	Seleccionar Temas Preguntas
Roles:	Profesor
Más Detalles:	Permite al profesor editar un tema ya creado

Tabla 21: Propiedades Vista "temas/edit"

Vista:	**views/temas/index.html.erb**
Caso de Uso Relacionado:	Seleccionar Temas Preguntas
Roles:	Profesor
Más Detalles:	Lista los temas que han sido creados

Tabla 22: Propiedades Vista "temas/index"

Vista:	**views/temas/new.html.erb**
Caso de Uso Relacionado:	Seleccionar Temas Preguntas
Roles:	Profesor
Más Detalles:	Permite al profesor crear un nuevo tema

Tabla 23: Propiedades Vista "temas/new"

Vista:	**views/temas/show.html.erb**
Caso de Uso Relacionado:	Seleccionar Temas Preguntas
Roles:	Profesor
Más Detalles:	Muestra información sobre un tema en concreto.

Tabla 24: Propiedades Vista "temas/show"

Vista:	**views/usuarios/cambiar_password.html.erb**
Caso de Uso Relacionado:	Editar Datos Personales
Roles:	Profesor, Alumno
Más Detalles:	Una pantalla especial donde únicamente se puede modificar la contraseña.

Tabla 25: Propiedades Vista "usuarios/cambiar_password"

Vista:	**views/usuarios/edit.html.erb**
Caso de Uso Relacionado:	Editar Datos Personales
Roles:	Profesor, Alumno
Más Detalles:	

Tabla 26: Propiedades Vista "usuarios/edit"

Vista	**views/usuarios/index.html.erb**
Caso de Uso Relacionado	Obtener Información Usuarios
Roles	Profesor, Alumno
Más Detalles:	El profesor posee enlaces para acceder a las estadísticas de cualquier usuario alumno o bien para eliminar un usuario existente. Para si mismo dispondrá de un enlace mediante el cual podrá editar su información personal. El alumno solo puede visualizar la información de contacto de otros usuarios, solo disponiendo de enlaces para ello. Para si mismo tiene otro enlace mediante el cual puede editar su propio perfil.

Tabla 27: Propiedades Vista "usuarios/index"

Vista	**views/usuarios/new.html.erb**
Caso de Uso Relacionado	Inscribir Alumnos
Roles	Profesor
Más Detalles:	

Tabla 28: Propiedades Vista "usuarios/new"

Vista:	**views/usuarios/show.html.erb**
Caso de Uso Relacionado:	Obtener Información Usuarios (Alumno, Profesor) Obtener Estadísticas Alumnos (Profesor)
Roles:	Profesor, Alumno
Más Detalles:	Para ambos roles se muestra la información de contacto del usuario seleccionado. El profesor verá además las estadísticas de la actividad del usuario en la aplicación, si el usuario del que se desea extraer la información es un alumno. El alumno podrá ver sus propias estadísticas de actividad en la aplicación si se selecciona a si mismo en esta opción. No podrá ver en cambio las estadísticas del resto de sus compañeros.

Tabla 29: Propiedades Vista "usuarios/show"

Por último, cabe mencionar también que para hacer de la distinción de roles algo intuitivo a simple vista, se ha optado por introducir esquemas de colores propios para cada rol; de acuerdo a esto, un esquema de colores rojo y negro será el predominante cuando el usuario que haya iniciado la sesión tenga el rol de profesor, y uno amarillo y azul será el utilizado cuando el usuario que esté utilizando la aplicación sea un alumno.

Figura 27: Ejemplos de las dos interfaces de la aplicación

TRATAMIENTO Y RECUPERACIÓN DE ERRORES

Los errores que pueden producirse mientras un usuario utiliza la aplicación pueden clasificarse en las siguientes categorías:

- Errores en la introducción de datos en formularios, lo que puede ocasionar que se archiven registros irregulares en la base de datos.
- Ejecución por error por parte de un profesor de alguna de las acciones relacionadas con la administración de la aplicación.
- Intento por parte de un usuario de acceder a funciones o páginas ajenas a su rol.

En el primer caso, puede especificarse en las clases correspondientes al modelo de los distintos módulos las restricciones que han de cumplir los campos de un registro para que este pueda llegar a ser almacenado en la base de datos. Además, adicionalmente a estas restricciones existe un atributo ***message*** el cual permite especificar un mensaje que será mostrado por la vista en el caso de que se especifique algún tratamiento de errores en este componente.

A continuación se muestra un ejemplo que se puede encontrar en la aplicación, correspondiente al modelo del módulo de preguntas.

```
validates_presence_of :enunciado, :respuestaA, :respuestaB, :respuestaC,
:respuestaD, :correcta, :message => "debe ser rellenado"
```

Estas instrucciones garantizan que para que una pregunta se almacene o actualice, sus campos para el enunciado, las cuatro respuestas y la respuesta correcta deben tener algún valor.

Por último, para que el mensaje de error pueda mostrarse de alguna forma cuando un formulario sea rellenado incumpliendo algunas de las restricciones mencionadas hace falta especificarlo en la vista correspondiente (normalmente aquellas que ejecutan las acciones de creación y edición) mediante la siguiente línea.

```
<%= error_messages_for :pregunta, :header_message => "Se han encontrado erro-
res en el formulario", :message => "Existen problemas en los siguientes cam-
pos:" %>
```

Con estos cambios, cada vez que un usuario cometa un error rellenando un formulario recibirá una notificación como la siguiente:

Figura 28: Ejemplo de error en un formulario

Para el segundo caso comentado; en el cual un profesor ejecutaría alguna de sus funciones como administrador de forma errónea, tal como crear un usuario con el rol equivocado, introducir mal los datos del servidor de correo o eliminar por error un tema o un usuario; la aplicación se ha construido de forma tal que todas estas operaciones sean **reversibles** de alguna forma. Es decir, que si un profesor da por error a un alumno privilegios de profesor al inscribirle, puede eliminarle y crearle de nuevo correctamente; si ha introducido mal los datos del servidor de correo, puede volver a introducirlos hasta hacerlo correctamente; y, por último, el caso más sensible, la eliminación por error de un usuario o un tema, que acarrearía la eliminación en cadena de las preguntas y valoraciones relacionadas, se ha resuelto añadiendo una ventana de confirmación que se le mostrará al profesor cada vez que decida ejecutar esta acción, avisándole de las consecuencias de la misma y disminuyendo significativamente de esta forma la probabilidad de que sea ejecutada por error.

Figura 29: Ejemplo de alerta al eliminar un registro

Por último, para aquellos casos en los que un usuario decida acceder a una acción o página para la cual no posea permisos, se ha optado por detectar estos casos y automáticamente redirigir al usuario a una página o una acción por defecto del dominio que le corresponda.

6. PRUEBAS REALIZADAS

Las distintas pruebas llevadas a cabo durante el desarrollo de este proyecto han tenido dos finalidades:

- Dar soporte y guiar la fase de desarrollo e implementación
- Dar una garantía del correcto funcionamiento de las distintas funcionalidades implementadas, para poder así asegurar la calidad del producto final entregado.

Como se ha comentado ya varias veces a lo largo del documento, la aplicación web a desarrollar está basada en una arquitectura Modelo-Vista-Controlador, con cada uno de estos componentes siendo independientes entre si. Este hecho trasladado a la fase de pruebas acarrea como consecuencia el que algunas de las pruebas a realizar se centren en aspectos cuya funcionalidad resida principalmente en el modelo, en el controlador o en la vista; y que según donde resida dicha funcionalidad, las pruebas necesiten un enfoque o unas herramientas concretas.

La primera prioridad de este capítulo pasa, por tanto, por dar un enfoque y determinar que herramientas se emplearán para las pruebas que atañen a modelo, vista y controlador respectivamente.

- Las pruebas concernientes al **modelo** son aquellas que se encuentran relacionadas con el modelo de datos de la aplicación. Todo modelo de datos incluye una serie de normas, excepciones y relaciones que lo adaptan al problema a resolver. Por tanto, las pruebas realizadas en el modelo irán orientadas a comprobar el correcto cumplimiento de estas normas, excepciones y relaciones establecidas en el modelo de datos de la aplicación tratando que ningún tipo de información que las contravenga sea almacenada en la base de datos. Este tipo de pruebas pueden ser perfectamente automatizadas mediante el uso de **pruebas unitarias**, siendo las mismas además de gran ayuda para el proceso de implementación, puesto que en ellas pueden establecerse los resultados finales deseados para cada operación que se lleve a cabo sobre el modelo de datos.
- Las clases y funciones involucradas en el controlador contienen la mayor parte de la funcionalidad que se espera de la aplicación. Según la operación a realizar, la información que se obtenga del modelo de datos será transformada y preparada para ser mostrada al usuario por pantalla. De nuevo, las **pruebas unitarias** pueden ser de gran ayuda para garantizar que todas estas operaciones concernientes a la lógica de la aplicación se realizan correc-

tamente y que, por tanto, la información que posteriormente será mostrada por pantalla es la correcta.

- Finalmente, existen diferencias en el lado de la **vista** para los usuarios de rol profesor y de rol alumno, como ya se ha comentado anteriormente, el contenido de una página en la cual se encuentre un usuario puede variar según el rol que este tenga. Las pruebas unitarias no pueden ser de ayuda en este último caso, pues a partir de ellas no es posible determinar qué debe de mostrar el navegador en cada momento, por tanto, las pruebas de esta funcionalidad únicamente pueden ser realizadas ejecutándose por un usuario en un navegador. Como el navegador fue ya empleado como elemento de depuración mientras dicha funcionalidad fue desarrollada, este capítulo se centrará únicamente en las pruebas unitarias realizadas para garantizar el correcto funcionamiento de modelo y controlador.

REALIZACIÓN DE PRUEBAS UNITARIAS EN *RAILS*

Rails posee en su árbol de directorios una carpeta **test** orientada a la implementación de pruebas sobre el código existente. Dichas pruebas a realizar serán pruebas unitarias, es decir, irán destinadas a probar el correcto funcionamiento de un módulo de la aplicación en concreto.

Dicha carpeta posee los siguientes subdirectorios concernientes a la implementación de pruebas unitarias:

fixtures: Donde se incluirán una serie de ficheros *YAML* en los cuales, y siguiendo el mencionado formato de serialización, será posible crear las instancias que serán empleadas en las distintas pruebas de forma sencilla e intuitiva.

functional: En esta carpeta tendrán lugar las pruebas unitarias relativas a la funcionalidad de cada uno de los módulos a probar, en otras palabras, en esta carpeta se codificarán las pruebas encargadas de garantizar el correcto funcionamiento de los distintos controladores de la aplicación.

unit: En la cual se almacenarán las distintas pruebas unitarias que se lleven a cabo sobre los modelos de datos de los distintos módulos a probar.

Además de pruebas unitarias, *Rails* permite llevar a cabo otra serie de pruebas sobre el código desarrollado, siendo también posible implementar tests de integración, para probar el funcionamiento de varios módulos a la vez o de la aplicación en su totalidad; o de rendimiento, permitiendo realizar pruebas sobre aspectos como la rapidez en la ejecución o el uso de memoria de la aplicación con el fin de ayudar a su optimización.

El uso de algunos de los scripts incluídos en *Rails* como ***generate Scaffold***, ***generate Mailer*** o ***generate Controller*** tienen como consecuencia la creación automática de ficheros en los

cuales el programador puede empezar a programar directamente las pruebas unitarias que desee utilizar así como a crear las instancias que tomarán parte en las mismas.

- Cuando ***generate Scaffold*** es invocado se creará un fichero *YAML* en **test/fixtures** donde podrán crearse las instancias de prueba para el módulo que se acabe de crear, una clase *Ruby* en la carpeta **test/unit** donde realizar pruebas unitarias sobre el modelo de datos y otra clase *Ruby* en la carpeta **test/functional** donde se implementarán las pruebas sobre el controlador

- Cuando ***generate Mailer*** es el *script* empleado, se creará una clase en la que será posible llevar a cabo pruebas unitarias sobre la instancia del *mailer* creada en **test/unit**.

- Si un controlador es generado mediante el script ***generate Controller***, únicamente una clase *Ruby* para la realización de pruebas unitarias será generada en la carpeta **test/functional**.

Por ello, dado que anteriormente se había invocado ***generate Scaffold*** en cuatro ocasiones

```
>> script/generate scaffold Pregunta enunciado:string tema:text respues-
taA:string, respuestaB:string respuestaC:string respuestaD:string correc-
ta:string feedback:string usuario:references

>> script/generate scaffold Usuario login:string password:string nom-
bre:string apellidos:string email:string rol:string url_foto:string

>> script/generate scaffold Tema titulo:string descripcion:text

>> script/generate scaffold Valoracion pregunta:references usuario:references
puntuacion:integer feedback:text
```

generate Controller en dos:

```
>> script/generate controller Acceso

>> script/generate controller Setup
```

y ***Mailer*** en una:

```
>> script/generate mailer Registro envio_password
```

Se generarán automáticamente los siguientes ficheros en la carpeta test:

```
fixtures/preguntas.yml

fixtures/temas.yml

fixtures/usuarios.yml

fixtures/valoraciones.yml

functional/acceso_controller_test.rb

functional/preguntas_controller_test.rb

functional/setup_controller_test.rb
```

```
functional/temas_controller_test.rb
functional/usuarios_controller_test.rb
functional/valoraciones_controller_test.rb
unit/pregunta_test.rb
unit/registro_test.rb
unit/tema_test.rb
unit/usuario_test.rb
unit/valoracion_test.rb
```

RESUMEN DE LAS PRUEBAS UNITARIAS REALIZADAS

Modelos

El objetivo de las pruebas unitarias sobre el modelo de datos de los distintos módulos será el de garantizar el cumplimiento de las condiciones de validación impuestas por dicho modelo de datos para aquellos registros que vayan a ser almacenados en la base de datos.

Modelo Preguntas (unit/pregunta_test.rb)

Test nº	1	**Nombre test:**	Prueba Creación
Descripción:			
Creación de un registro para ser almacenado en la tabla **Preguntas**			
Aserciones:			
El registro introducido, sin haber incumplido ninguna de las condiciones de validación, es válido			

Tabla 30: Test Modelo Preguntas. Creación

Test nº	2	**Nombre test:**	Prueba Comprobar No Enunciado
Descripción:			
Se intenta probar la creación de una pregunta para la cual no se ha especificado el enunciado.			
Aserciones:			
El registro a crear no es válido			

Tabla 31: Test Modelo Preguntas. No Enunciado

Test nº	3	**Nombre test:**	Prueba Comprobar No Respuesta A
Descripción:			
Se intenta probar la creación de una pregunta para la cual no se ha especificado la primera de las posibles respuestas.			
Aserciones:			
El registro a crear no es válido			

Tabla 32: Test Modelo Preguntas. No Respuesta A

Test nº	4	**Nombre test:**	Prueba Comprobar No Respuesta B
Descripción:			
Se intenta probar la creación de una pregunta para la cual no se ha especificado la segunda de las posibles respuestas.			
Aserciones:			
El registro a crear no es válido			

Tabla 33: Test Modelo Preguntas. No Respuesta B

Test nº	5	**Nombre test:**	Prueba Comprobar No Respuesta C
Descripción:			
Se intenta probar la creación de una pregunta para la cual no se ha especificado la tercera de las posibles respuestas.			
Aserciones:			
El registro a crear no es válido			

Tabla 34: Test Modelo Preguntas. No Respuesta C

Test nº	6	**Nombre test:**	Prueba Comprobar No Respuesta D
Descripción:			
Se intenta probar la creación de una pregunta para la cual no se ha especificado la cuarta de las posibles respuestas.			
Aserciones:			
El registro a crear no es válido			

Tabla 35: Test Modelo Preguntas. No Respuesta D

Test nº	7	**Nombre test:**	Prueba Comprobar No Respuesta Correcta
Descripción:			
Se intenta probar la creación de una pregunta para la cual no se ha especificado cuál de las cuatro posibles respuestas es la correcta			
Aserciones:			
El registro a crear no es válido			

Tabla 36: Test Modelo Preguntas. No Respuesta Correcta

Resultados de los tests:

```
>> ruby test/unit/pregunta_test.rb

Loaded suite test/unit/pregunta_test

Started

.......

Finished in 0.151868 seconds.

7 tests, 7 assertions, 0 failures, 0 errors
```

Modelo Temas (unit/tema_test.rb)

Test nº	1	**Nombre test:**	Prueba Creación
Descripción:			
Se intenta crear un tema que cumple con todas las condiciones de validación			
Aserciones:			
El registro introducido, sin haber incumplido ninguna de las condiciones de validación, es válido			

Tabla 37: Test Modelo Temas. Creación

Test nº	2	**Nombre test:**	Prueba Tema Repetido
Descripción:			
Se intenta crear un tema que ya existe en la base de datos.			
Aserciones:			
Al no poder haber dos temas con el mismo título, el registro es inválido.			

Tabla 38: Test Modelo Temas. Tema Repetido

Test nº	3	**Nombre test:**	Prueba Tema Sin Título
Descripción:			
Se intenta crear un tema para el cual el campo *título* se ha dejado en blanco.			
Aserciones:			
Al ser obligatorio introducir un título para cada tema que se vaya a crear, el registro es inválido.			

Tabla 39: Test Modelo Temas. Tema Sin Título

Resultados de los tests:

```
>> ruby test/unit/tema_test.rb

Loaded suite test/unit/tema_test

Started

...

Finished in 0.090777 seconds.


3 tests, 3 assertions, 0 failures, 0 errors
```

Modelo Usuarios (unit/usuario_test.rb)

Test nº	1	**Nombre test:**	Prueba Creación
Descripción:			
Se intenta crear un usuario que cumple todas las condiciones de validación.			
Aserciones:			
El registro introducido, sin haber incumplido ninguna de las condiciones de validación, es válido			

Tabla 40: Test Modelo Usuarios. Creación

Test nº	2	**Nombre test:**	Prueba Login Repetido
Descripción:			
Se intenta crear un usuario cuyo nombre de usuario (*login*) es uno ya existente en la base de datos.			
Aserciones:			
Al ser el nombre de usuario un identificador único empleado para acceder a la aplicación, el registro no es válido.			

Tabla 41: Test Modelo Usuarios. *Login* Repetido

Test nº	3	**Nombre test:**	Prueba No Login
Descripción:			
Se intenta crear un usuario habiendo dejado vacío el campo correspondiente a su nombre de usuario.			
Aserciones:			
Al ser el campo mencionado necesario para que el usuario pueda acceder a la aplicación, el registro introducido no es válido			

Tabla 42: Test Modelo Usuarios. No *Login*

Test nº	4	**Nombre test:**	Prueba No E-mail
Descripción:			
Se intenta crear un usuario sin especificar el campo correspondiente a su dirección de correo electrónico.			
Aserciones:			
Como el correo electrónico es el medio por el cual los usuarios recibirán los datos que necesiten para acceder a la aplicación, no puede admitirse la creación de un usuario sin dirección de correo electrónico y el registro, por tanto, no es válido.			

Tabla 43: Test Modelo Usuarios. No E-mail

Test nº	5	**Nombre test:**	Prueba E-mail Repetido
Descripción:			
Se intenta crear un usuario cuya dirección de correo electrónico es la misma de otro usuario ya existente.			
Aserciones:			
Al correo electrónico llega información privada, como las contraseñas de los usuarios, por tanto no pueden permitirse dos registros con la misma dirección de correo. Se comprueba que el registro introducido no es válido.			

Tabla 44:Test Modelo Usuarios. E-mail Repetido

Test nº	6	**Nombre test:**	Prueba Login Corto
Descripción:			
Se intenta crear un usuario cuyo nombre de usuario es menor de 3 caracteres.			
Aserciones:			
Emplear nombres de usuario de muy pequeña longitud o a la inversa, puede llevar a confusiones mientras se usa la aplicación. Tanto en un caso como en otro, el registro será no válido.			

Tabla 45: Test Modelo Usuarios. Login Corto

Test nº	7	**Nombre test:**	Prueba Password Insegura
Descripción:			
Se intenta crear un usuario con una contraseña de tamaño menor a 6 caracteres.			
Aserciones:			
Contraseñas tan cortas resultan mucho más vulnerables a ataques, por cuestiones de seguridad se considerarán los registros con tales contraseñas como no válidos.			

Tabla 46: Test Modelo Usuarios. *Password* Insegura

Test nº	8	**Nombre test:**	Prueba Password Mal Confirmada
Descripción:			
Se simula la creación de un usuario que ha introducido mal el campo de confirmación de contraseña.			
Aserciones:			
Se considera el registro no válido y no se almacena, pues el usuario podría estar introduciendo una contraseña no deseada por error.			

Tabla 47: Test Modelo Usuarios. *Password* Mal Confirmada

Test nº	9	**Nombre test:**	Prueba E-mail No Válido
Descripción:			
Se intenta crear un usuario que ha introducido una dirección de correo electrónico no válida (que no cumple con el formato de una dirección de correo)			
Aserciones:			
El registro introducido no es válido.			

Tabla 48: Test Modelo Usuarios. E-mail No Válido

Test nº	10	**Nombre test:**	Prueba Autenticación
Descripción:			
Se simula la autenticación de un usuario ante el sistema mediante su *login* y contraseña.			
Aserciones:			
Como el *login* y contraseña introducidos son correctos, se comprueba que los datos del usuario son enviados al controlador.			

Tabla 49: Test Modelo Usuarios. Prueba Autenticación

Test nº	11	**Nombre test:**	Prueba Autenticación Fallida
Descripción:			
Se simula la autenticación de un usuario ante el sistema mediante su *login* y contraseña, siendo uno de estos campos erróneos.			
Aserciones:			
Se comprueba que no se envía ningún tipo de información al controlador, pues el proceso de autenticación ha fallado			

Tabla 50: Test Modelo Usuarios. Autenticación Fallida

Resultados de los tests:

```
>> ruby test/unit/usuario_test.rb
Loaded suite test/unit/usuario_test
Started
...........
Finished in 0.235488 seconds.

11 tests, 11 assertions, 0 failures, 0 errors
```

Modelo Valoraciones (unit/valoracion_test.rb)

Test nº	1	**Nombre test:**	Prueba Creación
Descripción:			
Se simula la creación de una valoración para una pregunta por parte de un usuario.			
Aserciones:			
Se comprueba el correcto almacenamiento de una valoración cuyos datos son válidos.			

Tabla 51: Test Modelo Valoraciones. Creación

Resultados de los tests:

```
>> ruby test/unit/valoracion_test.rb

Loaded suite test/unit/valoracion_test

Started

.

Finished in 0.079314 seconds.



1 tests, 1 assertions, 0 failures, 0 errors
```

Controladores

Controlador Acceso (functional/acceso_controller_test.rb)

Test nº	1	**Nombre test:**	Prueba Login
Descripción:			
Se prueba el inicio de una sesión en la aplicación por parte de un usuario tras autenticarse introduciendo su nombre de usuario y contraseña.			
Aserciones:			
1. Se confirma que el usuario es redirigido desde la página de acceso a su página de inicio. 2. Se confirma que se ha creado una sesión para el usuario mencionado.			

Tabla 52: Test Controlador Acceso. *Login*

Test nº	2	**Nombre test:**	Prueba Login Fallido
Descripción:			
Se prueba el inicio de una sesión en la aplicación por parte de un usuario después de que el proceso de autenticación de éste haya fallado por el nombre de usuario.			
Aserciones:			
Se confirma que no se ha creado ninguna sesión.			

Tabla 53: Test Controlador Acceso. *Login* Fallido

Test nº	3	**Nombre test:**	Prueba Password Fallida
Descripción:			
Se prueba el inicio de una sesión en la aplicación por parte de un usuario después de que el proceso de autenticación de éste haya fallado por la introducción de una contraseña errónea.			
Aserciones:			
Se confirma que no se ha creado ninguna sesión.			

Tabla 54: Test Controlador Acceso. Password *Fallida*

Test nº	4	**Nombre test:**	Prueba Logout
Descripción:			
Se prueba el cierre de sesión por parte de un usuario que estaba activo en la aplicación..			
Aserciones:			
1. Se comprueba que el usuario abandona la página donde se encontraba. 2. Se comprueba que la sesión se ha cerrado. 3. Se comprueba que el usuario ha sido redirigido a la página de acceso.			

Tabla 55: Prueba Controlador Acceso. *Logout*

Test nº	5	**Nombre test:**	Prueba Login-Logout
Descripción:			
Se prueba en primer lugar el inicio de sesión de un usuario en la aplicación tras una autenticación correcta, y posteriormente se cierra la sesión iniciada por este mismo usuario.			
Aserciones:			
1. Se comprueba que el usuario abandona la página de acceso y es redirigido a su página de inicio. 2. Se comprueba que se ha creado una sesión para el usuario correctamente tras el inicio. 3. Se comprueba que tras realizarse el cierre de sesión el usuario abandona la página en la que se encontraba. 4. Se comprueba que la sesión se ha cerrado correctamente tras el cierre. 5. Se comprueba que el usuario ha sido redirigido de nuevo a la página de acceso.			

Tabla 56: Test Controlador Acceso. *Login-Logout*

Resultados de los tests:

```
>> ruby test/functional/acceso_controller_test
Loaded suite test/functional/acceso_controller_test
Started
.....
Finished in 0.116886 seconds.


5 tests, 12 assertions, 0 failures, 0 errors
```

Controlador Configuración (functional/setup_controller_test.rb)

Test nº	1	**Nombre test:**	Prueba Configurar Administrador
Descripción:			
Se simula la carga del formulario de configuración del usuario administrador (o el usuario que inicializa la aplicación).			
Aserciones:			
Se comprueba la correcta carga de la página que contiene dicho formulario.			

Tabla 57: Prueba Controlador Configuración. Configurar Administrador

Test nº	2	**Nombre test:**	Prueba Configurar Mailer
Descripción:			
Se simula la carga del formulario de configuración del gestor de correo electrónico de la aplicación.			
Aserciones:			
Se comprueba la correcta carga de la página que contiene dicho formulario.			

Tabla 58: Prueba Controlador Configuración. Configurar *Mailer*

Resultados de los tests:

```
>> ruby test/functional/setup_controller_test
Loaded suite test/functional/setup_controller_test
Started
..
Finished in 0.089948 seconds.


2 tests, 2 assertions, 0 failures, 0 errors
```

Controlador Preguntas (functional/preguntas_controller_test.rb)

Test nº	1	**Nombre test:**	Prueba Listado
Descripción:			
Se simula la llamada a la página de listado de preguntas.			
Aserciones:			
1. Se comprueba que la llamada se ha llevado a cabo con éxito. 2. Se comprueba que la estructura de datos que contiene las preguntas a listar existe.			

Tabla 59: Prueba Controlador Preguntas. Listado

Test nº	2	**Nombre test:**	Prueba Nueva Pregunta
Descripción:			
Se simula la llamada a la página de creación de preguntas.			
Aserciones:			
Se comprueba que la llamada se ha llevado a cabo con éxito.			

Tabla 60: Prueba Controlador Preguntas. Nueva Pregunta

Test nº	3	**Nombre test:**	Prueba Crear Pregunta
Descripción:			
Se simula la creación de una pregunta desde el controlador en la aplicación.			
Aserciones:			
1. Se comprueba que la pregunta se ha creado (existe una pregunta más que antes en la base de datos). 2. Se comprueba que tras la creación el usuario es redirigido correctamente a la página donde puede ver todas las preguntas que ha realizado.			

Tabla 61:Prueba Controlador Preguntas. Crear Pregunta

Test nº	4	**Nombre test:**	Prueba Mostrar Pregunta
Descripción:			
Se simula la recuperación de una pregunta de la base de datos y la llamada a la página que la muestra por pantalla.			
Aserciones:			
Se comprueba el éxito del resultado de la operación.			

Tabla 62: Prueba Controlador Preguntas. Mostrar Pregunta

Test nº	5	**Nombre test:**	Prueba Editar Pregunta
Descripción:			
Se simula la recuperación de una pregunta de la base de datos y la llamada a la página que contiene el formulario para editarla.			
Aserciones:			
Se comprueba el éxito del resultado de la operación			

Tabla 63: Prueba Controlador Preguntas. Editar Pregunta

Test nº	6	**Nombre test:**	Prueba Actualizar Pregunta
Descripción:			
Se simula la actualización de una pregunta desde el controlador.			
Aserciones:			
Se comprueba el éxito de la operación y que el usuario es redirigido correctamente a la página en la que puede ver la pregunta que acaba de modificar.			

Tabla 64: Prueba Controlador Preguntas. Actualizar Pregunta

Test nº	7	**Nombre test:**	Prueba Eliminar Pregunta
Descripción:			
Se simula la eliminación de una pregunta desde el controlador.			
Aserciones:			
1. Se comprueba el éxito de la operación (existe una pregunta menos que antes en la base de datos. 2. Se comprueba que tras eliminar la pregunta el usuario es redirigido correctamente a la página donde puede visualizar sus preguntas publicadas.			

Tabla 65: Test Controlador Preguntas. Eliminar Pregunta

Resultados de los tests:

```
>> ruby test/functional/preguntas_controller_test
Loaded suite test/functional/preguntas_controller_test
Started
.......
Finished in 0.158789 seconds.


7 tests, 10 assertions, 0 failures, 0 errors
```

Controlador Temas (functional/temas_controller_test.rb)

Test nº	1	**Nombre test:**	Prueba Listado Temas
Descripción:			
Se simula la llamada a la página que lista los temas disponibles en la aplicación			
Aserciones:			
1. Se confirma el éxito de la operación mencionada. 2. Se comprueba que la estructura de datos que contiene las preguntas a listar existe.			

Tabla 66: Test Controlador Temas. Listado

Test nº	2	**Nombre test:**	Prueba Nuevo Tema
Descripción:			
Se simula la carga de la página que contiene el formulario que permite la creación de un nuevo tema.			
Aserciones:			
Se verifica el éxito de la operación.			

Tabla 67: Test Controlador Temas. Nuevo Tema

Test nº	3	**Nombre test:**	Prueba Nuevo Tema Por Alumno
Descripción:			
Se simula la carga de la página que contiene el formulario que permite la creación de un nuevo tema por parte de un usuario de rol alumno, al cual no debería estar permitido ejecutarla.			
Aserciones:			
Se verifica que la operación no puede llevarse a cabo.			

Tabla 68: Test Controlador Temas. Nuevo Tema Por Alumno

Test nº	4	**Nombre test:**	Prueba Crear Tema
Descripción:			
Se simula la creación de un nuevo tema en la base de datos desde el controlador.			
Aserciones:			
1. Se comprueba que el nuevo tema se ha creado (existe un tema más que antes en la base de datos). 2. Se comprueba que tras la creación el usuario es redirigido correctamente a la página donde puede ver todos los temas que han sido publicados hasta el momento.			

Tabla 69: Test Controlador Temas. Crear Tema

Test nº	5	**Nombre test:**	Prueba Crear Tema Por Alumno
Descripción:			
Se simula la creación de un nuevo tema en la base de datos desde el controlador por parte de un alumno, a cuyo rol no le está permitido realizar esta operación			
Aserciones:			
1. Se comprueba que el nuevo tema no se ha creado (existe el mismo numero de temas que antes en la base de datos). 2. Se comprueba que tras la creación el usuario es redirigido correctamente a una página correspondiente a su rol y a su sesión (la que contiene el listado de preguntas al que puede responder).			

Tabla 70: Test Controlador Temas. Crear Tema Por Alumno

Test nº	6	**Nombre test:**	Prueba Mostrar Tema
Descripción:			
Simulación de la llamada a la página que muestra un tema en concreto por pantalla.			
Aserciones:			
Se verifica el éxito de la operación mencionada.			

Tabla 71: Test Controlador Temas. Mostrar Tema

Test nº	7	**Nombre test:**	Prueba Editar Tema
Descripción:			
Simulación de la llamada a la página que contiene el formulario que permite editar un tema ya existente.			
Aserciones:			
Se verifica el éxito de la operación.			

Tabla 72: Test Controlador Temas. Editar Tema

Test nº	8	**Nombre test:**	Prueba Editar Tema Por Alumno
Descripción:			
Simulación de la llamada a la página que contiene el formulario que permite editar un tema ya existente por parte de un alumno, al cual no le está permitido realizar dicha operación.			
Aserciones:			
Se verifica que la operación mencionada no puede llevarse a cabo.			

Tabla 73: Test Controlador Temas. Editar Tema Por Alumno

Test nº	9	**Nombre test:**	Prueba Actualizar Tema
Descripción:			
Simulación que simula la modificación de un tema ya existente en la base de datos realizada desde el controlador.			
Aserciones:			
Se verifica el éxito de la operación y que el usuario es redirigido correctamente a la página en la que puede ver el tema que acaba de modificar.			

Tabla 74: Test Controlador Temas. Actualizar Tema

Test nº	10	**Nombre test:**	Prueba Eliminar Tema
Descripción:			
Simulación de la eliminación de un tema de la base de datos realizada desde el controlador.			
Aserciones:			
1. Se verifica el éxito de la operación comprobando que el existe un tema menos en la base de datos. 2. Se verifica que tras la eliminación, el usuario es redirigido correctamente a la página donde se listan todos los temas creados.			

Tabla 75: Test Controlador Temas. Eliminar Tema

Test nº	11	**Nombre test:**	Prueba Eliminar Tema Por Alumno
Descripción:			
Simulación de la eliminación de un tema de la base de datos realizada desde el controlador y por un usuario de rol alumno. Los usuarios de este rol no pueden realizar esta operación.			
Aserciones:			
1. Se verifica que la operación no se ha podido llevar a cabo, comprobando que el número de temas existentes en la base de datos no ha variado. 2. Se comprueba que tras eliminar la pregunta el usuario es redirigido correctamente a la página donde puede visualizar sus preguntas publicadas.			

Tabla 76: Test Controlador Temas. Eliminar Tema Por Alumno

Resultados de los tests:

```
>> ruby test/functional/temas_controller_test
Loaded suite test/functional/temas_controller_test
Started
..........
Finished in 0.173635 seconds.

11 tests, 14 assertions, 0 failures, 0 errors
```

Controlador Usuarios (functional/usuarios_controller_test.rb)

Test nº	1	**Nombre test:**	Prueba Listado
Descripción:			
Se simulará la llamada a la página en la cual será posible consultar el listado de usuarios de la aplicación.			
Aserciones:			
1. Se verificará el correcto funcionamiento de esta operación. 2. Se comprobará que la estructura de datos que ha de contener los usuarios a mostrar existe.			

Tabla 77: Test Controlador Usuarios. Listado

Test nº	2	**Nombre test:**	Prueba Nuevo Usuario
Descripción:			
Se simulará la llamada a la página que contendrá el formulario de creación de un nuevo usuario.			
Aserciones:			
Se verificará que dicha llamada se ejecuta correctamente.			

Tabla 78: Test Controlador Usuarios. Nuevo Usuario

Test nº	3	**Nombre test:**	Prueba Nuevo Usuario Por Alumno
Descripción:			
Se simulará la llamada a la página que contendrá el formulario de creación de un nuevo usuario por parte de un alumno. Los alumnos no tienen permisos para inscribir nuevos usuarios en la aplicación.			
Aserciones:			
Se verifica que el alumno es redirigido a una página de su dominio cuando intenta ejecutar esta operación			

Tabla 79: Test Controlador Usuarios. Nuevo Usuario Por Alumno

Test nº	4	**Nombre test:**	Prueba Crear Usuario
Descripción:			
Se simulará la creación de un usuario en la base de datos realizada desde el controlador.			
Aserciones:			
1. Verificar que el usuario se ha creado correctamente comprobando que en la base de datos existe un usuario más que antes de que se creara. 2. Redirigir al usuario que ha realizado la operación a la pantalla en la cual se lista a los usuarios de la aplicación.			

Tabla 80: Test Controlador Usuarios. Crear Usuario

Test nº	5	**Nombre test:**	Prueba Crear Usuario Inicio
Descripción:			
Se simulará la creación del que será el primer usuario del sistema, la cual tendrá que realizarse, por tanto, cuando ningún usuario haya iniciado una sesión en el sistema.			
Aserciones:			
1. Verificar que el usuario se ha creado correctamente comprobando que en la base de datos existe un usuario más que antes de que se creara. 2. Verificar que el usuario ha sido redirigido a la siguiente página de configuración de la aplicación.			

Tabla 81: Test Controlador Usuarios. Crear Usuario Inicio

Test nº	6	**Nombre test:**	Prueba Mostrar Usuario
Descripción:			
Se simulará la recuperación de un usuario de la base de datos y la llamada a la página que lo muestra por pantalla.			
Aserciones:			
Verificar que la operación mencionada se llevó a cabo correctamente.			

Tabla 82: Test Controlador Usuarios. Mostrar Usuario

Test nº	7	**Nombre test:**	Prueba Editar Usuario
Descripción:			
Se simulará la recuperación de un usuario de la base de datos para su edición.			
Aserciones:			
Verificar que la operación mencionada se llevó correctamente.			

Tabla 83: Test Controlador Usuarios. Editar Usuario

Test nº	8	**Nombre test:**	Prueba Actualizar Usuario
Descripción:			
Se simulará la modificación de los datos de uno de los usuarios existentes en la base de datos.			
Aserciones:			
1. Comprobar que la operación se ha llevado a cabo con éxito. 2. Comprobar que el usuario es redirigido a la página correcta una vez ha finalizado la actualización.			

Tabla 84: Test Controlador Usuarios. Actualizar Usuario

Test nº	9	**Nombre test:**	Prueba Eliminar Usuario
Descripción:			
Se simulará la eliminación de un usuario existente en la base de datos.			
Aserciones:			
1. Comprobar que la operación se ha llevado a cabo con éxito comprobando que el número de usuarios existentes en la base de datos se ha reducido en una unidad. 2. Comprobar que el usuario es redirigido a la página correcta una vez ha finalizado la actualización.			

Tabla 85: Test Controlador Usuarios. Eliminar Usuario

Test nº	10	**Nombre test:**	Prueba Eliminar Usuario Por Alumno
Descripción:			
Se simulará la eliminación de un usuario existente en la base de datos realizada por un usuario de rol alumno, rol que no posee permisos para llevar a cabo esta acción.			
Aserciones:			
1. Comprobar que no se ha eliminado ningún usuario de la base de datos verificando que el número de usuarios almacenados en ella es el mismo. 2. Comprobar que el usuario de rol alumno es redirigido a una página de su dominio establecida por defecto.			

Tabla 86: Test Controlador Usuarios. Eliminar Usuario Por Alumno

Resultados de los tests:

```
>> ruby test/functional/usuarios_controller_test
Loaded suite test/functional/usuarios_controller_test
Started
..........
Finished in 0.193786 seconds.


10 tests, 14 assertions, 0 failures, 0 errors
```

Controlador Valoraciones (functional/valoraciones_controller_test.rb)

Test nº	1	**Nombre test:**	Prueba Crear Valoración
Descripción:			
Creación de una valoración sobre una pregunta por parte de un usuario llevada a cabo desde el controlador.			
Aserciones:			
1. Verificar que la valoración se ha creado correctamente. 2. Verificar que el usuario es redirigido a la página correcta tras realizar la valoración.			

Tabla 87: Test Controlador Valoraciones. Crear Valoración

Resultados de los tests:

```
>> ruby test/functional/valoraciones_controller_test

Loaded suite test/functional/valoraciones_controller_test

Started

.

Finished in 0.081711 seconds.


1 tests, 2 assertions, 0 failures, 0 errors
```

7. CONCLUSIONES Y TRABAJOS FUTUROS

CONCLUSIONES

El trabajo realizado puede analizarse desde dos frentes:

Por un lado cabe analizar la aplicación desarrollada y la funcionalidad de la misma. Sobre este aspecto se puede decir que, desde que la aplicación fue concebida, su principal objetivo fue el de promover que sus usuarios compartan sus conocimientos y fomentar de esta forma el aprendizaje general y no que los usuarios simplemente demuestren sus conocimientos respondiendo correctamente a las preguntas.

Este objetivo principal se ve reflejado en la distinta importancia que toman ciertos elementos como la existencia de cuadros de texto para aportar algún conocimiento adicional tanto cuando se hace una pregunta como cuando se responde, el papel del profesor (siendo fundamentalmente observador de la actividad en la aplicación y únicamente participando de forma activa en la proposición y creación de temas para las preguntas, y la no aparición de ningún tipo de calificación o medida de rendimiento similar relacionada con las respuestas que un usuario vaya dando a las distintas preguntas (ya que únicamente un porcentaje de preguntas acertadas y falladas por cada usuario es lo que aparece y lo hace por haberse considerado que es un dato estadístico que puede influir en las distintas valoraciones que un usuario haya tomado).

Por otro lado, la idea de realizar una aplicación en la cual los alumnos puedan intercambiar distintos conocimientos involucra que cuanto más se utilice la aplicación, más cumplirá esta con su objetivo, y es por esta razón por la cual se le ha dado especial importancia a un aspecto como el diseño de la interfaz de la aplicación, intentando conseguir como resultado una aplicación web de aspecto moderno, juvenil y vistoso que pudiera resultarle atractivo a los usuarios a los que va dirigida esta aplicación (estudiantes); y que además resulte sencilla de utilizar con accesos rápidos a las funciones más comunes, permitiendo llevar estas a cabo en pocos *clics* de ratón.

Finalmente, el otro frente desde el cuál puede analizarse este proyecto es el que concierne a la tecnología empleada (*Ruby on Rails*) así como a las metodologías de desarrollo que esta tecnología involucra (metodologías ágiles). Esta elección supuso posiblemente el factor de mayor riesgo en el desarrollo del proyecto, pues mi experiencia con *Ruby on Rails* era bastante escasa reduciéndose al desarrollo de una pequeña aplicación durante un curso intensivo en esta materia de dos semanas y media. Es por esta razón por la que solo puedo destacar lo cómodo y lo sencillo que resulta el adaptarse a este *framework*, el cual, en mi opinión, facilita y acelera enormemente la función del desarrollador mediante unas configuraciones de inicio muy efectivas, una estructura muy clara y, sobre todo, el uso de un lenguaje que resulta real-

mente intuitivo y funcional como es *Ruby*. Estas razones hacen posible que un desarrollador pueda empezar a trabajar en *Ruby On Rails* bastante pronto y con una gran libertad, no siendo necesario leer APIs kilométricas o comprender el funcionamiento de un gran número de librerías para empezar a utilizarlo con cierto criterio.

De este *framework* cabe destacar también una comunidad de usuarios creciente además de muy participativa, lo que resulta de gran ayuda para la resolución de problemas concretos en la fase de desarrollo.

TRABAJOS FUTUROS

A continuación se exponen una serie de posibles mejoras y extensiones que podrían ayudar en un futuro a la mejora de la aplicación:

- **Ampliar la variedad en los tipos de preguntas y respuestas:** En la actualidad, la aplicación solo soporta preguntas tipo test con una opción correcta entre cuatro posibles. El uso de preguntas de distinto tipo por tanto aumentaría los medios por los cuales los usuarios podrían intercambiar sus conocimientos además de aportar un mayor dinamismo y creatividad a la creación y resolución de preguntas. Unos ejemplos de estos nuevos tipos de pregunta que se podrían añadir son los siguientes:
  - **Preguntas de respuesta simple**: En las cuales el usuario pueda responder en pocas palabras.
  - **Preguntas de desarrollo:** Aquí podrían incluirse aquellas preguntas que posean una respuesta más compleja o incluso una discusión sobre un aspecto en concreto de la materia que se vaya a tratar. Las respuestas a estas preguntas no se considerarían como acierto o como fallo y solo estarían sujetas a la valoración de los usuarios de la aplicación.
- **Posibilitar la creación de tests o de conjuntos de preguntas:** Que los alumnos en un momento dado pudieran contestar a un conjunto de preguntas en vez de tener que seleccionar las preguntas que desean responder una a una en el índice de preguntas. Estos tests podrían ser:
  - **Predefinidos:** Tests que el profesor pudiera aportar a sus alumnos para que estos puedan entrenar sus conocimientos en un tema en concreto o en la materia globalmente.
  - **'Retos'**: Tests que un alumno propondría a otro en concreto a partir de las preguntas disponibles en la aplicación en ese momento. La aplicación podría resultar más divertida de usar con esta funcionalidad además de incentivar una sana competitividad de la cual ambos alumnos involucrados en el reto aprenderían conocimientos nuevos.

  - **Aleatorios:** Dejar que la propia aplicación genere dinámicamente un test a partir de las preguntas existentes.

- **Publicación de artículos y enlaces de interés**: Tanto los profesores como los alumnos tendrían la oportunidad de publicar y divisar desde la aplicación artículos que pudieran resultar de interés para la asignatura, además de poder discutir sobre ellos.

- **Método rápido de creación de usuarios:** Buscar métodos para que el proceso de añadir de usuarios sea menos tedioso para el profesor haciendo posible la creación de varios usuarios a la vez introduciendo la mínima información posible para ello (para esto pueden buscarse patrones dentro de la información de registro de los usuarios (números de identificación de alumno, dominio de las direcciones de correo electrónico propias de la universidad, etc.).

- **Comunicación con otras aplicaciones:** Abrir la posibilidad de que las preguntas de la aplicación así como cualquier otra información que pueda resultar de interés puedan ser exportadas a un formato como XML, con el objetivo de que puedan ser importadas por alguna aplicación similar que tenga intención de hacer uso de ellas.

- **Empleo de técnicas de análisis de datos:** Permitir que el profesor pueda realizar análisis e interpretaciones más profundos de la información que obtenga a partir de las estadísticas de la aplicación. Investigar el uso de librerías de análisis de datos o intentar exportar la información de las estadísticas a aplicaciones que se dediquen a esta función.

8. BIBLIOGRAFÍA

«10 Steps to deploy your RubyonRails Application on Windows Vista/2008 using Apache and Mongrel cluster.» (acceso el 30/6/2009). http://nlakkakula.wordpress.com/2008/11/24/10-steps-for-deploying-your-ruby-on-rails-application-on-a-windows-server-2008-apache-mongrel-cluster/.

«Agile Manifesto.» http://www.agilemanifesto.org (acceso el 30/6/2009).

Antti Tarvainen, Carsten Bormann. «Agile Web Development slides.» *Agile Web Development Intensive Course.* Tampere University of Technology, 2008.

«Basic User Authentication in Rails.» (acceso el 30/6/2009). http://www.aidanf.net/rails_user_authentication_tutorial.

Dave Thomas and David Heinemeier Hansson, with Leon Breedt, Mike Clark, James Duncan Davidson, Justin Gehtland, and Andreas Schwarz. *Agile Web Development with Rails.* 3rd Edition. The Pragmatic Bookshelf, 2008.

Galera, José Lozano. «El triángulo del e-learning.» http://www.telepolis.com/cgi-bin/web/DISTRITODOCVIEW?url=/1589/doc/Reflexiones/triangulo.htm (acceso el 1/12/2008).

«Gruff Graphs for Ruby.» http://nubyonrails.com/pages/gruff (acceso el 30/6/2009).

«How to paginate, sort and search a table with Ajax and Rails.» http://dev.nozav.org/rails_ajax_table.html (accessed 30 de 6 de 2009).
ImageMagick. http://www.imagemagick.org/script/index.php (acceso el 30/6/2009).

«Instalar Ruby on Rails en Windows.» http://wiki.rubyonrails.org/es/getting-started/installation/windows (acceso el 30/6/2009).

«Rails 2.0 and Scaffolding Step by Step.» http://fairleads.blogspot.com/2007/12/rails-20-and-scaffolding-step-by-step.html (acceso el 30/6/2009).

Rails Framework Documentation. http://api.rubyonrails.org/ (acceso el 30/6/2009).

RMagick Download Page. http://rmagick.rubyforge.org/ (acceso el 30/6/2009).

«Sorta_Nested_Layouts.» http://mattmccray.com/archive/2007/02/19/Sorta_Nested_Layouts/ (acceso el 30/6/2009).

Will_paginate library home page. http://wiki.github.com/mislav/will_paginate (acceso el 30/6/2009).

«Wikipedia - "Agile Software Development".» http://en.wikipedia.org/wiki/Agile_software_development (acceso el 1/12/2008)..

«Wikipedia - "Agile Web Development".» http://en.wikipedia.org/wiki/Agile_web_development (acceso el 1/12/2008)..

«Wikipedia - "E-Learning".» http://es.wikipedia.org/wiki/E-learning (acceso el 1/12/2008)..

«Wikipedia - "Model-View-Controller".» http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (acceso el 1/12/2008)..

«Wikipedia - "Ruby on Rails".» http://en.wikipedia.org/wiki/Ruby_on_rails (acceso el 1/12/2008)..

«Wikipedia - "Ruby".» http://es.wikipedia.org/wiki/Ruby (acceso el 1/12/2008)..

«Wikipedia - "Web 2.0".» http://es.wikipedia.org/wiki/Web_2.0 (acceso el 1/12/2008)..

ANEXO I – MANUAL DE INSTALACIÓN

Introducción:

La función de este manual es la de explicar como instalar *Ruby On Rails*, y una vez llegados a este punto, explicar como instalar, configurar y desplegar *AnswerIT* sobre un servidor Apache en una máquina que emplee **Microsoft Windows**.

Al final de este manual se incluirá también un apartado relativo a la instalación y el despliegue de *AnswerIT* sobre Apache en Linux (mucho más sencilla).

Requisitos:

Antes de comenzar con el proceso de instalación, ha de verificarse que se dispone del siguiente software:

- Apache 2.x
- MySQL 5.x o SQLite 3.6.x

En caso de no disponerse de alguno de ellos puede recurrirse a sus páginas oficiales para descargarlos gratuitamente (http://www.apache.org/, http://www.mysql.com/ o http://www.sqlite.org)

Instalación de Ruby:

Instalar *Ruby* a partir de su *one-click installer* para Windows, para ello puede descargarse gratuitamente su última versión de http://rubyforge.org/frs/?group_id=167, o bien utilizar la copia disponible en el CD de la aplicación (versión 1.8.6-26).

Seguir los pasos del instalador automático y esperar a que termine su función.

Instalación de Inkscape:

Inkscape es una librería de tratamiento y conversión de gráficos vectoriales (SVG) que será utilizado para la renderización de los gráficos de estadísticas, por ello, su instalación resulta también necesaria.

El fichero de instalación de dicha librería también se encuentra en el CD *ImageMagick-6.4.8-6-Q8-windows-dll.exe*, y para instalarlo será de nuevo preciso seguir los pasos del instalador automático hasta su finalización.

Una vez terminada la instalación de Inkscape, conviene que reinicie su ordenador, pues tanto esta instalación como la de *Ruby* habrán creado nuevas variables de entorno en el sistema operativo.

Instalación de las *gemas* necesarias para la aplicación:

Se conoce como *gem* (gema) a aquellos paquetes o librerías que se han creado en *Ruby*. Una vez se ha instalado *Ruby* se instalará también *RubyGems*, que es el gestor de estos paquetes y librerías. Tanto *Rails* como otros paquetes y librerías que serán empleadas por la aplicación desarrollada deberán ser instaladas utilizando *RubyGems*.

Utilizar *RubyGems* para instalar un paquete *Ruby* es sencillo. Bastará con la siguiente línea:

```
>>  gem install <nombre gema>
```

Por ello el siguiente paso será instalar las *gemas* necesarias para que *AnswerIT* pueda correr:

El primer paso será actualizar la versión existente de *Rubygems*, lo cual se consigue entrando, en la unidad de cd, en el directorio **rubygems-1.3.4**:

```
>> cd rubygems-1.3.4
```

Para a continuación ejecutar el script de instalación que actualizará todo lo relacionado con *Rubygems*, y por tanto, con la futura instalación de paquetes.

```
>> ruby setup.rb
```

Una vez la actualización de *RubyGems* haya finalizado, solo quedará volver al directorio raíz de la unidad de CD, entrar en el directorio **install**, y, desde allí, ejecutar el fichero de instalación **install.bat**

```
>> cd ..

>> cd install

>> install
```

Con esto se instalará *Rails* además de todos los componentes y librerías necesarios para que *AnswerIT* funcione.

Una vez se ha llegado a este punto, *AnswerIT* puede empezar a configurarse, luego el siguiente paso será descomprimir el fichero **answerit.rar** que contiene la aplicación al directorio deseado del disco duro en el que se desee instalar.

Configuración de la base de datos:

El siguiente paso será configurar la base de datos, si MySQL es la utilizada (SQLite no requiere de este paso previo) creando una base de datos donde la información de AnswerIT se almacenará y, de desearse, un usuario con capacidad de manipular esta base de datos.

Una vez realizado este paso será posible configurar la base de datos de la aplicación, tarea que se realiza en el fichero **config/database.yml** dentro de la ruta en la que previamente se descomprimió la aplicación. Como la base de datos que se va a emplear en este punto no estará dedicada a tareas de test ni de desarrollo, sino de producción, se modificarán los siguientes parámetros:

```
production:

   adapter: mysql

   database: <nombre que se le ha dado a la base de datos>

   username: <nombre del usuario de MySQL que manejará la base de datos>

   password: <contraseña del usuario introducido en el parámetro anterior>

   host: localhost (o dirección IP del ordenador en que se encuentre la base
de datos)
```

Ya en este punto puede cargarse el modelo de datos definido en la aplicación en la base de datos mediante la siguiente instrucción:

```
>> rake db:schema:load RAILS_ENV=production
```

Configurar el servidor

Lo siguiente será configurar Apache para que pueda alojar *AnswerIT*. Para ello se necesita retirar los comentarios de las siguientes líneas en el fichero **conf/http.conf**:

```
LoadModule rewrite_module modules/mod_rewrite.so

LoadModule proxy_module modules/mod_proxy.so

LoadModule proxy_balancer_module modules/mod_proxy_balancer.so

LoadModule proxy_http_module modules/mod_proxy_http.so
```

Estos cambios habilitarán el que sea posible redirigir mediante una serie de proxies las instancias de *AnswerIT* que estarán funcionando bajo el servidor de *Ruby* ***Mongrel*** hacia el servidor **Apache**, para lo cual habrá que configurar un *host* virtual.

Este *host* virtual se especificará en el fichero **conf/extra/httpd-vhosts.conf** donde habrá que añadir las siguientes líneas:

```
NameVirtualHost *:80

#Proxy balancer section (create one for each ruby app cluster)
```

```
<Proxy balancer://Myapp_cluster>
       BalancerMember http://127.0.0.1:3001
       BalancerMember http://127.0.0.1:3002
       BalancerMember http://127.0.0.1:3003
</Proxy>
<VirtualHost *:80>
       DocumentRoot C:/www/PFC/public
       ErrorLog logs/Myapp-error.log
       CustomLog logs/Myapp-access.log common
       <Directory C:/www/PFC/public/ >
              Options Indexes FollowSymLinks MultiViews
              AllowOverride All
              Order allow,deny
              allow from all
       </Directory>
       RewriteEngine On
       # Rewrite index to check for static files
       RewriteRule ^/$ /index.html [QSA]
       # Rewrite to check for Rails cached pages
       RewriteRule ^([^.]+)$ $1.html [QSA]
       # Redirect all non-static requests to cluster
       RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
       RewriteRule ^/(.*)$ balancer://Myapp_cluster%{REQUEST_URI} [P,QSA,L]
</VirtualHost>
```

Una vez hechos estos cambios habrá que reiniciar Apache.

Arrancar la aplicación:

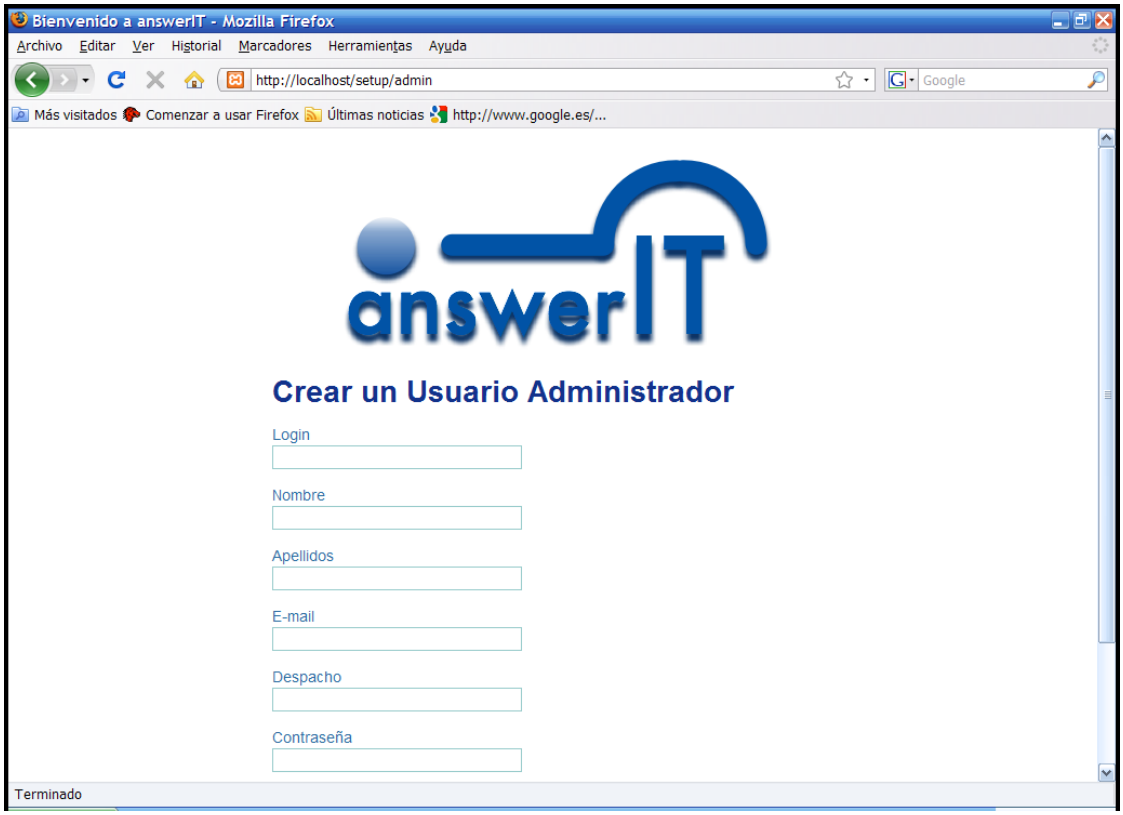
Únicamente faltará ya declarar varias instancias de Mongrel, que funcionarán a modo de *cluster* gracias al paquete mongrel_services anteriormente instalado mediante el fichero *batch* que se facilita a tal efecto:

```
>> deploy
```

Y finalmente arrancar dichas instancias mediante el fichero *start.bat*

```
>> start-answerit
```

Llegados a este punto, en el navegador bastará con introducir en la barra de direcciones http://localhost o http://localhost/<ruta> si el host virtual se montó sobre una ruta en concreto del Puerto 80 para ver *AnswerIT* funcionando y su primer formulario de configuración.



Al rellenarlo, el usuario sera automáticamente el primer usuario registrado en la aplicación y lo hará con el rol de professor, pudiendo así ejercer labores administrativas y dar de alta a alumnos o a otros profesores. Asimismo, aparecerá otro formulario en el cual el profesor deberá rellenar los datos correspondientes al servidor de correo electronico (SMTP) desde el cual *AnswerIT* deberá enviar sus notificaciones.

Detener la aplicación:

```
>> stop-answerit
```

Instalación en Linux/Mac OSX:

AnswerIT puede instalarse y desplegarse también en sistemas UNIX como Linux o Mac OSX. En estos casos habrá que instalar igualmente *Ruby* e *Inkscape* en primer lugar (bien descargándolos gratuitamente de su web oficial o de los paquetes incluídos en algunas distribuciones de *Linux).*

Tras ello, habrá que igualmente instalar las *gemas* que se instalaban para *Windows* a excepción de *mongrel y mongrel_service*.

La base de datos, de resultar necesario, se configuraría de la misma forma; y es en el proceso de despliegue de la aplicación en *Apache* en el cual la instalación presenta la mayor diferencia, pues bajo sistemas *UNIX*, es posible utilizar *Phusion Passenger* para esta función. *Phusion Passenger* es un módulo de *Apache* para el despliegue de aplicaciones web que empleen el lenguaje *Ruby*, haciendo de esta función un proceso bastante más rápido y automático además de incrementar la eficiencia de la aplicación mientras está corriendo en el servidor (http://www.modrails.com/index.html).

Posibles Problemas:

En caso de aparecer el siguiente mensaje de error al migrar la base de datos o al tratar de arrancar la aplicación: `undefined method 'each' for #<Mysql:0xffffff>`, la solución viene copiando el fichero **libmysql.dll** de una instalación funcional de PHP al directorio **bin** dentro del directorio donde se instaló *Ruby.* Existe una copia de esta librería también en el CD.

ANEXO II – MANUAL DE USUARIO

Índice

Introducción:

El objetivo de este documento es el de familiarizar al usuario, ya sea usuario o profesor, con las distintas funcionalidades que *AnswerIT* le ofrece, mostrándole también como llevar a cabo cada una de ellas.

Por ello, este manual ofrecerá un formato *HOWTO*, tratando de responder de la forma más eficiente y clara a las necesidades tipo que pudieran surgir en los usuarios de la aplicación. Primero se dará respuesta a aquellas cuestiones más generales, y, posteriormente, a aquellas que dependan de un rol concreto.

Cuestiones Generales

¿Cómo puedo acceder a AnswerIT?

Para poder acceder a *AnswerIT* es preciso estar registrado en ella, para ello debe haberse recibido el e-mail correspondiente. En este correo electrónico aparecerá tu nombre de usuario y tu contraseña en la aplicación, así como un enlace a la *URL* donde *AnswerIT* se encuentre localizada.

Al acceder a la *URL* de *AnswerIT* la siguiente página aparecerá:

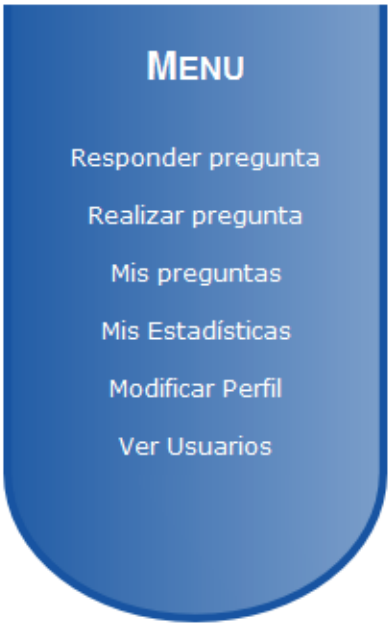
E introduciendo el nombre de usuario y la contraseña que has recibido podrás entrar.

Se que en mi curso se está utilizando AnswerIT pero no estoy registrado en ella aún, ¿puedo solicitar mi registro?

Sí, en la página inicial de la aplicación, debajo del cuadro donde se introducen los datos de acceso, existe un enlace que permite enviar un e-mail a uno de los profesores responsables de la aplicación, que podrá registrarte.

He conseguido entrar ya, ¿cómo me muevo por la aplicación?

La mejor herramienta para moverse entre distintas funcionalidades de *AnswerIT* es el menú principal, el cual se encuentra a tu izquierda en la pantalla. En él se mostraran las funciones principales propias de tu rol en *AnswerIT*.



He terminado mi actividad en AnswerIT, ¿cómo cierro mi sesión en la aplicación?

Solo hay que pulsar el enlace “Cerrar Sesión” que aparece bajo la cabecera con el logo de la aplicación y al lado del mensaje de saludo.

Si soy profesor...

¿Cómo agrego alumnos u otros profesores a la aplicación?

Selecciona la opción “Añadir Usuario” en el menú principal e inmediatamente el formulario para añadir un nuevo usuario aparecerá.

Introduce el e-mail del usuario que desees añadir así como el nombre de usuario que desees que le identifique en la aplicación. En el selector “Rol” especifica si el usuario a ser añadido es alumno o profesor (alumno, al ser la opción mayoritaria, es la seleccionada por defecto en este selector). Una vez la información introducida sea la deseada pulsa el botón de “Añadir”.

Tras esta operación aparecerá justo debajo de la cabecera un mensaje informando del éxito de la operación y en pantalla el listado de usuarios de la aplicación, donde el usuario recién añadido debería ya aparecer de haber ido todo correctamente.

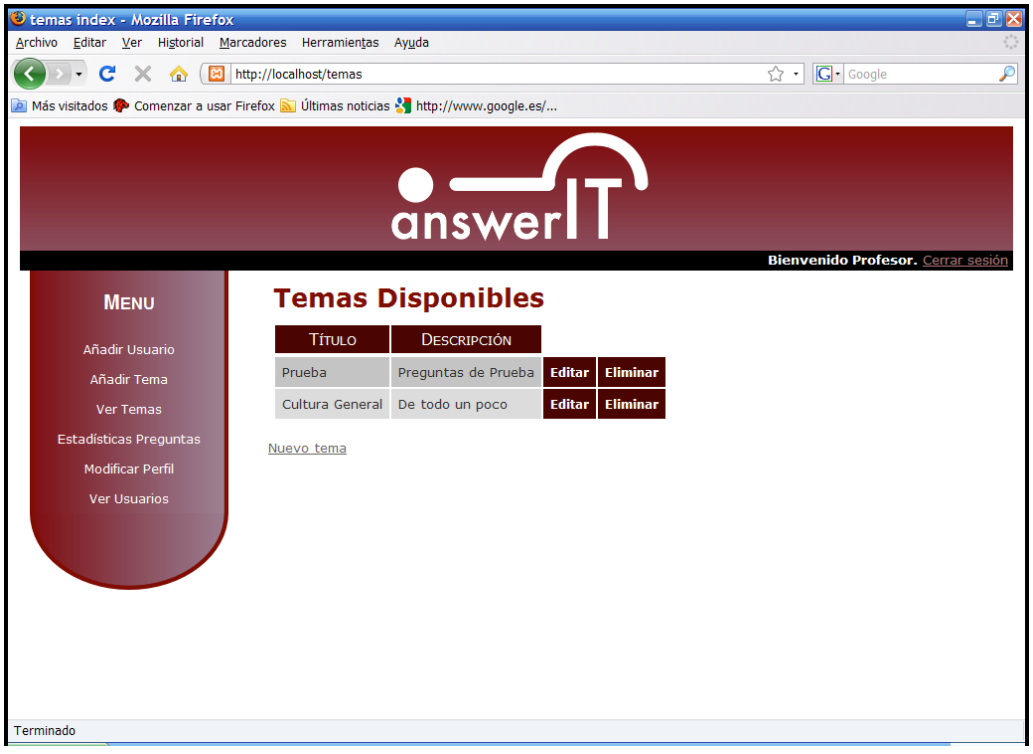
¿Cómo propongo los temas sobre los cuales mis alumnos pueden preguntar?

Selecciona la opción “Añadir Tema” en el menú principal e inmediatamente aparecerá el formulario para añadir un nuevo tema en pantalla. Especifica un título para el tema, y, si lo deseas, una descripción del mismo. Una vez hayas terminado pulsa el botón de “Añadir”.

Tras realizar la operación aparecerá en pantalla la información del tema que acabas de crear.

¿Cómo se los temas que hasta ahora han sido creados?

Seleccionando en el menú principal “Ver Temas”. Inmediatamente aparecerá un listado con los temas existentes en la aplicación.



¿Cómo modifico la información de un tema ya existente?

Accediendo al listado de temas mediante la opción “Ver Temas” de la que se habla en la pregunta anterior, y pulsando sobre el enlace de “Editar” que aparezca al lado del tema que desees modificar.

¿Cómo elimino la información de un tema ya existente?

Pulsando el enlace de “Eliminar” de nuevo en el listado de temas accesible desde la opción “Ver Temas” del menú principal. Deberás confirmar esta operación en una ventana emergente para poder llevarla a cabo.

¿Cómo afecta la modificación de un tema a las preguntas existentes en él?

Si el tema al que pertenecen se ha actualizado, las modificaciones correspondientes a esta actualización aparecerán también en las preguntas.

¿Cómo afecta la eliminación de un tema a las preguntas existentes en él?

Las preguntas existentes en un tema serán eliminadas si el tema al que pertenecen es eliminado.

¿Cómo puedo ver las preguntas que mis alumnos han publicado?

Mediante la opción “Estadísticas Preguntas” del menú principal. Inmediatamente aparecerá un listado con las preguntas que han sido publicadas hasta ese momento ordenadas de forma aleatoria. Esta opción será la que aparecerá por defecto cada vez que inicies sesión.

¿Cómo puedo buscar una pregunta concreta de una forma sencilla en el listado, si éste está ordenado aleatoriamente?

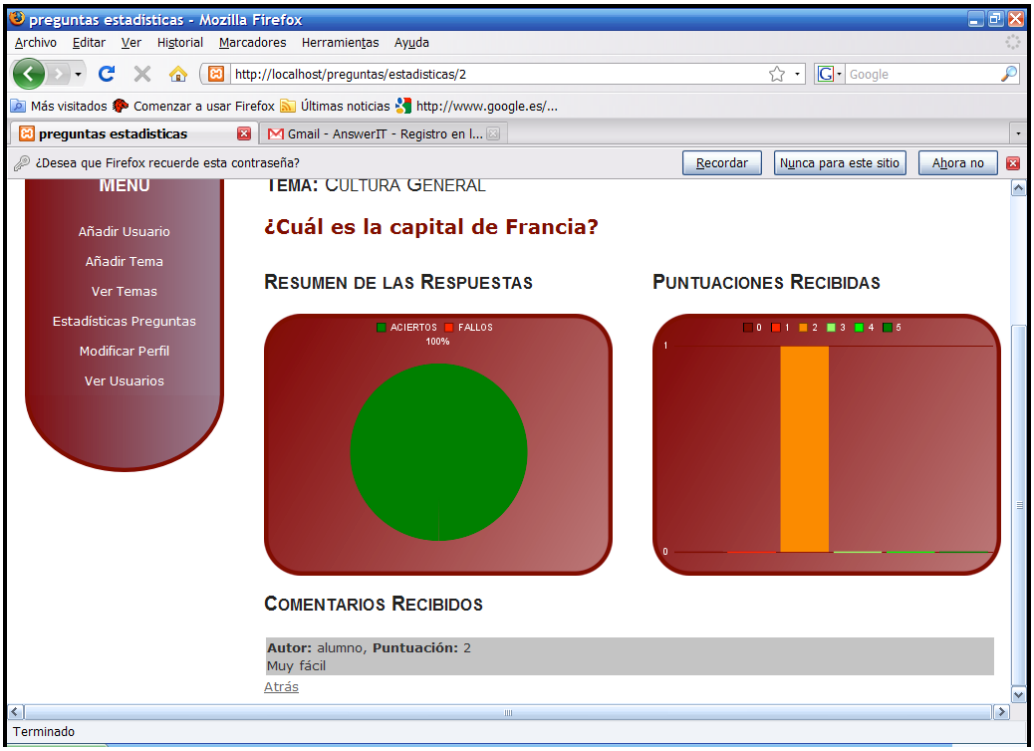
Puedes ordenar las preguntas tanto en orden ascendente como descendente utilizando como criterio su tema, su enunciado, su autor y su valoración. Para ello solo necesitas pulsar el enlace correspondiente en las cabeceras de la tabla de listado.

En el listado solo puedo ver el enunciado de las preguntas ¿Puedo ver también sus posibles respuestas?

Sí, para ello hace falta pinchar sobre el enlace “Ver” que aparece al lado de cada pregunta en el listado de preguntas.

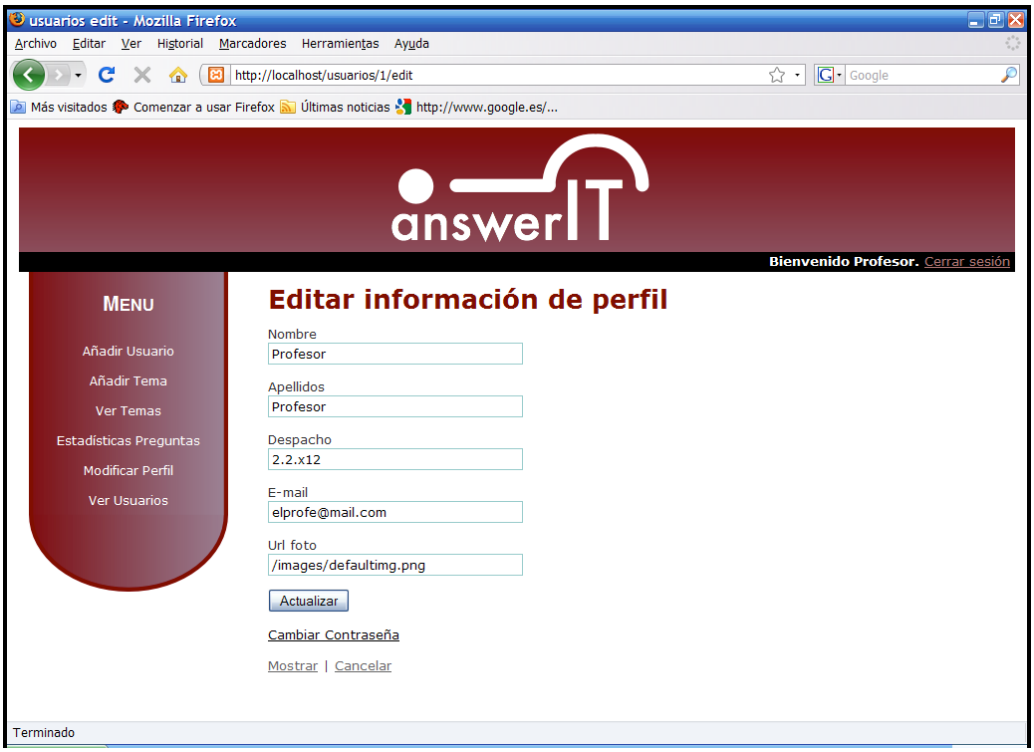
¿Cómo puedo ver las estadísticas relacionadas con una pregunta en concreto y de qué información dispongo en este apartado?

Pinchando sobre el enlace “Estadísticas” que aparece para cada pregunta en el listado. Esta pantalla ofrece dos gráficos, dando a conocer el porcentaje de aciertos de los alumnos al responder esta pregunta así como un resumen de las puntuaciones recibidas. Debajo, será posible leer los distintos comentarios de *feedback* que los alumnos han ido dejando al responder la pregunta.



¿Cómo modifico información propia de mi perfil o mi contraseña?

Seleccionando en el menú principal “Modificar Perfil”. Nada más hacerlo automáticamente se cargará un formulario en el que verás la información actual de tu perfil pudiendo modificarla.



Para modificar tu contraseña accede al enlace “Cambiar Contraseña” que aparece justo debajo del formulario y un nuevo formulario preguntándote por tu nueva contraseña y su confirmación aparecerá.

¿Cómo puedo acceder a la información y a las estadísticas de mis alumnos?

A partir de la opción “Ver Usuarios” del menú principal se obtiene un listado completo de los usuarios que forman parte de la aplicación. En el cuadro correspondiente a cada usuario aparecerá un enlace “Mostrar” que al ser pulsado te permitirá acceder a la información personal de un usuario en concreto, y si este usuario es un alumno, a sus estadísticas.

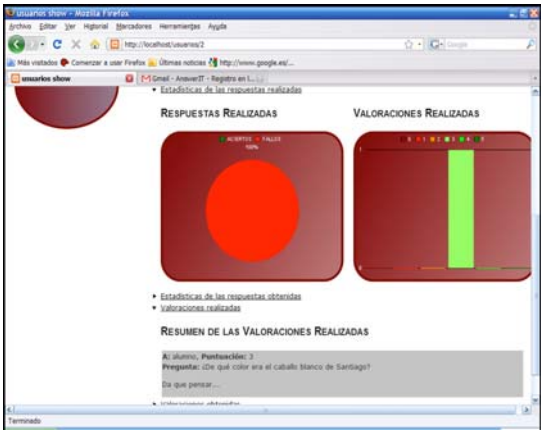
Las estadísticas de un alumno estarán divididas en cuatro apartados:

- **Estadísticas de las preguntas realizadas:** Mostrará dos gráficos. El primero que informará el acierto con el cual el alumno seleccionado ha respondido las preguntas de sus compañeros y el segundo mostrará un resumen de sus valoraciones a las distintas preguntas de sus compañeros que haya respondido.

- **Estadísticas de las respuestas obtenidas:** Mostrando dos gráficos, el primero referente al acierto con el que las preguntas que ha formulado este alumno han sido respondidas, y el segundo mostrando como han valorado sus preguntas el resto de alumnos de la aplicación.

- **Valoraciones Realizadas:** Mostrará una a una las valoraciones realizadas por el alumno seleccionado junto a los comentarios de feedback que ha aportado e información sobre el autor de la pregunta que ha sido valorada.

- **Valoraciones Obtenidas:** Mostrará una a una las valoraciones que el alumno seleccionado ha recibido por parte de sus compañeros junto con los comentarios de feedback de las mismas y el identificador del autor de dichas valoraciones.



¿Cómo elimino un usuario de la aplicación?

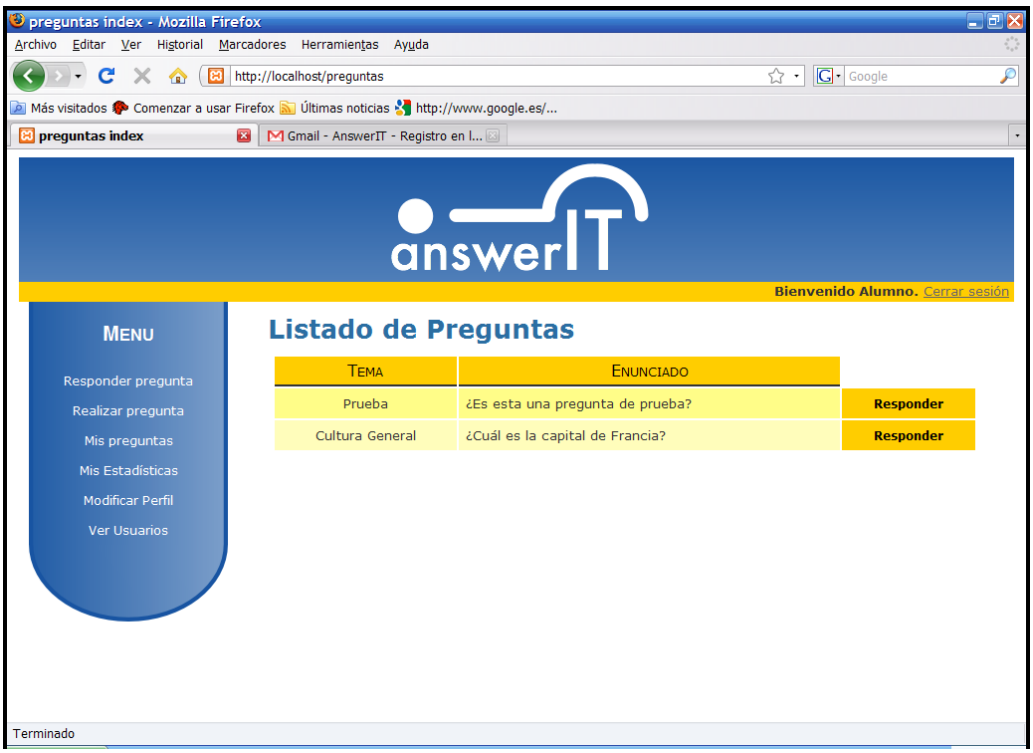
Entrando al listado de alumnos mediante la opción “Ver Usuarios” del menú principal y seleccionando el enlace “Eliminar” correspondiente al usuario que deseas eliminar.

Cuando un usuario es eliminado se eliminan con él las preguntas y las valoraciones que haya realizado, por tanto, para evitar las pérdidas de información que la ejecución de esta operación por error pueda ocasionar, será necesario confirmar la eliminación en una ventana emergente.

Si soy alumno...

¿Cómo veo las preguntas a las que puedo responder?

Seleccionando en el menú principal la opción “Responder Pregunta”. Inmediatamente aparecerá un listado con las preguntas que puedes responder. Esta opción será también la que aparecerá por defecto cada vez que inicies sesión.

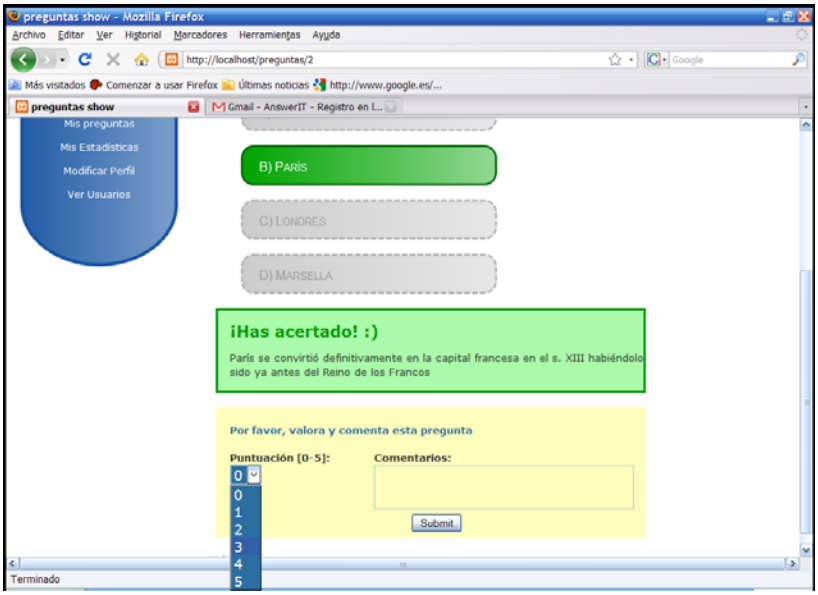


¿Cómo respondo a una pregunta?

Seleccionando el enlace “Responder” para la pregunta que se desee contestar en el listado de preguntas. Inmediatamente podrás ver las cuatro posibles respuestas a la pregunta y seleccionar la que creas correcta.

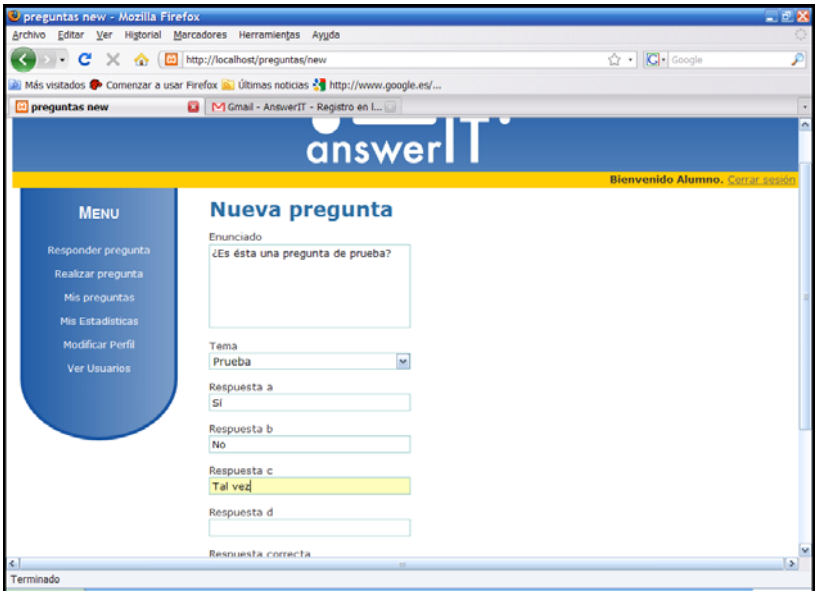
¿Cómo valoro una pregunta?

Respondiéndola un primer lugar. Tras hacerlo, inmediatamente aparecerá el formulario para realizar una nueva valoración justo debajo de las respuestas que acabas de responder. Aquí podrás puntuar la pregunta que acabas de responder e introducir un comentario a modo de feedback sobre la pregunta.



¿Cómo publico mis propias preguntas?

Seleccionando la opción “Realizar Pregunta” en el menú principal. Tras pinchar en ella, el formulario para la creación de una nueva pregunta aparecerá, en él rellena los campos correspondientes al enunciado, las cuatro respuestas posibles y la respuesta correcta, y especifica un tema de los existentes para publicarla. Si lo deseas, también puedes añadir algo de conocimiento adicional sobre el elemento tratado por la pregunta a modo de “feedback”, que les será mostrado a los alumnos que respondan a tu pregunta.

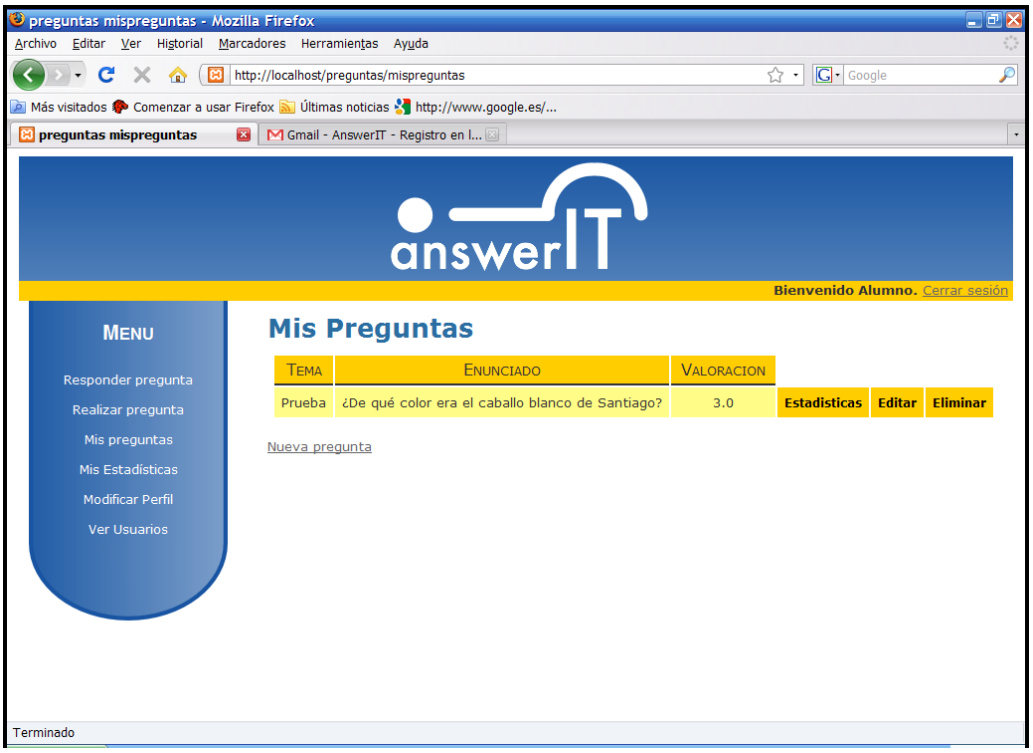


Cuando hayas terminado, puedes publicarla en la aplicación pulsando el botón de “Crear”.

Una vez publicada tu pregunta serás notificado de ello y redirigido automáticamente a la opción “Mis Preguntas” donde se podrá ver la pregunta que acabas de publicar junto con las demás que hayas publicado.

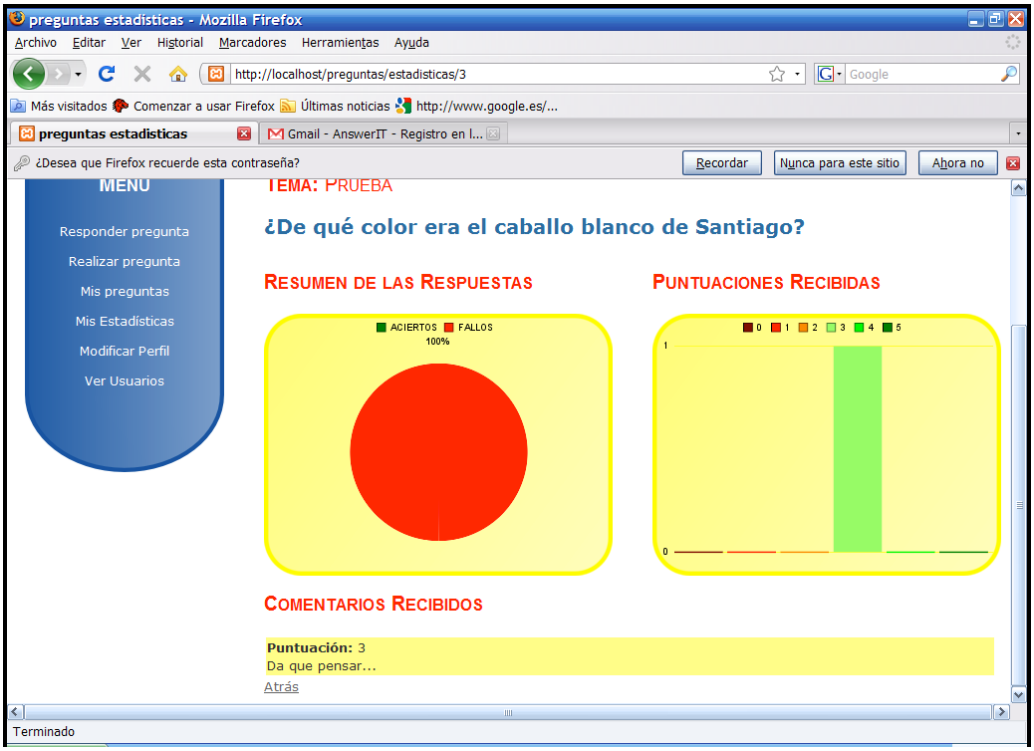
¿Cómo veo las preguntas que he publicado?

Seleccionando la opción “Mis Preguntas” en el menú principal. Tras pinchar en ella, un listado con las preguntas que hayas publicado aparecerá por pantalla.



¿Puedo consultar estadísticas relacionadas con mis preguntas?

Sí, en el listado de la opción “Mis Preguntas” aparece un enlace al lado de cada pregunta a través del cual es posible acceder a las estadísticas de la pregunta correspondiente. Las estadísticas consistirán en dos gráficos, dando a conocer el porcentaje de aciertos de los alumnos al responder esta pregunta así como un resumen de las puntuaciones recibidas. Debajo, será posible leer los distintos comentarios de *feedback* que tus compañeros han ido dejando al responder la pregunta.



¿Puedo modificar una pregunta que ya he publicado?

Sí, igual que en el caso anterior, en el listado de la opción “MisPreguntas” aparece un enlace “Editar”, el cual si es pulsado provocará la aparición en pantalla de un formulario en el cual aparecerán y podrán ser editados los distintos elementos que conforman la pregunta.

¿Cómo elimino una pregunta que haya publicado?

De nuevo y al igual en que en los casos anteriores, habrá que acudir a la opción “Mis Preguntas” y en el listado seleccionar el enlace “Eliminar” correspondiente a la pregunta que desees suprimir. Eliminando una pregunta eliminas también las valoraciones que se hayan realizado sobre ella, luego se te pedirá una confirmación para esta operación en una ventana emergente.

¿Cómo veo las estadísticas de mi actividad en la aplicación?

Mediante la opción “Mis estadísticas” del menú principal. Las estadísticas que podrán ser consultadas son las siguientes:

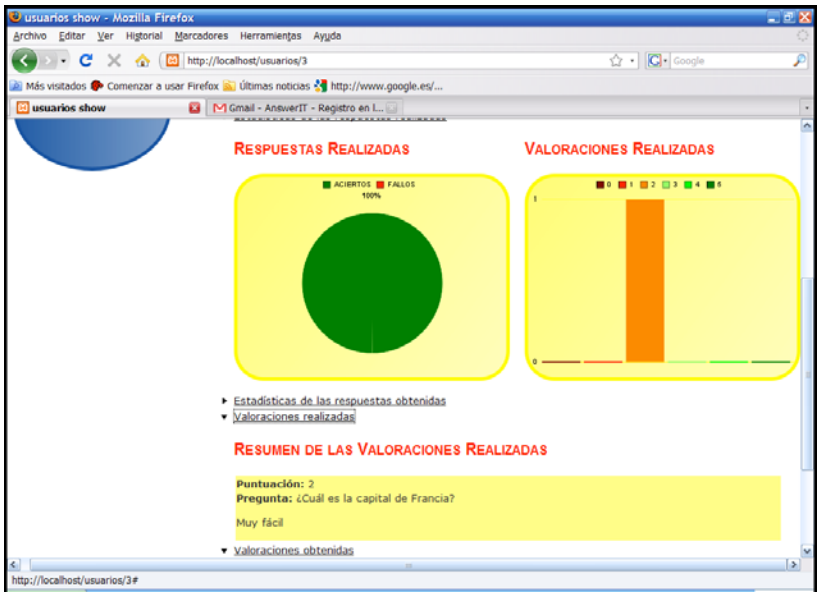
- **Estadísticas de las preguntas realizadas:** Mostrará dos gráficos. El primero te informará del acierto con el cual has respondido las preguntas de sus compañeros y el segundo mostrará un resumen de tus valoraciones a las distintas preguntas de tus compañeros que hayas respondido.

- **Estadísticas de las respuestas obtenidas:** Mostrando dos gráficos, el primero referente al acierto con el que las preguntas has publicado han sido respondidas, y el se-

gundo mostrando como han valorado tus preguntas el resto de alumnos de la aplicación.

- **Valoraciones Realizadas:** Mostrará una a una las valoraciones que hayas realizado junto con los comentarios de feedback que hayas decidido añadir.

- **Valoraciones Obtenidas:** Mostrará una a una las valoraciones que tus preguntas han recibido por los distintos alumnos que las hayan respondido.



¿Cómo modifico información propia de mi perfil o mi contraseña?

Seleccionando en el menú principal “Modificar Perfil”. Nada más hacerlo automáticamente se cargará un formulario en el que verás la información actual de tu perfil pudiendo modificarla.

Para modificar tu contraseña accede al enlace “Cambiar Contraseña” que aparece justo debajo del formulario y un nuevo formulario preguntándote por tu nueva contraseña y su confirmación aparecerá.

¿Cómo accedo a la información de contacto de mis profesores o mis compañeros?

Mediante la opción “Ver Usuarios” del menú principal. Una vez seleccionada aparecerá un listado con los distintos alumnos de la aplicación separados por rol. Pulsando el enlace de “Mostrar” para un usuario podrás acceder a información más detallada acerca de este usuario.