Universidad
Rey Juan Carlos

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2011-2012

Proyecto Fin de Carrera

Aplicación web de edición y explotación de encuestas online

Autor: Carlos Pérez Camarasa

Tutor: Pedro de Las Heras Quirós

Agradecimientos

En primer lugar, quiero agradecer a mi tutor, Pedro de las Herás Quirós, su esfuerzo y dedicación para ayudarme a completar este proyecto final de carrera.

Por otra parte, quiero agradecer a mis familiares el apoyo incondicional que me han dado desde siempre, animándome a seguir adelante, y facilitándome los medios necesarios para alcanzar mis propósitos. Son la base donde se sustenta todo lo que soy a día de hoy.

Finalmente, pero no por ello menos importantes, quiero agradecer a todos mis amigos el haber estado siempre conmigo, para darme fuerzas. En especial, a Patricia, Xabe, María, Lara y Nair, mis cinco ángeles, sin las cuales posiblemente nunca hubiese llegado hasta aquí.

Resumen

La realización de encuestas supone una de las fuentes de información más importantes para la estadística moderna. Sus ámbitos de uso se extienden a prácticamente cualquier actividad humana, sea comercial, académica o lúdica, donde se pretenda conocer la opinión de las personas que han estado involucradas. En el caso de nuestra Universidad, también se realizan numerosas encuestas con la intención de evaluar la calidad de las clases impartidas. Sin embargo, el método utilizado actualmente es muy rudimentario, necesitando gran cantidad de trabajo manual, entre otros muchos inconvenientes.

El presente proyecto representa una alternativa viable a estas necesidades, mediante una aplicación web. El objetivo fundamental es proveer una herramienta que pueda gestionar de forma autónoma gran parte de la cadena de procesado de una encuesta. Esto abarca desde su concepción primigenia hasta su puesta en producción y recogida de resultados. También nos interesa ofrecer a los usuarios una interfaz que sea cómoda de utilizar, sin que ello suponga renunciar a las tecnologías de desarrollo más actuales, con animaciones y efectos interesantes y dinámicos.

Básicamente, nuestra aplicación será capaz de generar encuestas, ya sea completamente nuevas, o utilizando otras ya existentes como plantillas. A estas encuestas podremos insertarle preguntas, que podrán estar almacenadas en una base de datos, o bien podremos crearlas nosotros mismos. Una vez tengamos nuestra encuesta final, la aplicación nos permitirá hacerla pública para que cualquier usuario conteste a nuestras preguntas. Y, finalmente, los datos obtenidos se pondrán a disposición del creador de la encuesta para que pueda revisarlos.

La aplicación ha sido desarrollada íntegramente por el alumno, utilizando el *framework* Ruby on Rails, que es capaz de abarcar la interfaz gráfica, la lógica computacional y la gestión de datos, apoyándose en herramientas como las librerías AJAX para desarrollo de páginas dinámicas, o MySQL para la gestión de bases de datos.

Índice

1. Introducción **1**

1.1. Estadística y encuestas 1

1.2. Procesado de la información 2

1.3. Funcionamiento en la Universidad 3

1.4. Estado del arte 5

2. Objetivos **7**

2.1. Descripción del problema 7

2.2. Requisitos 8

2.3. Metodología y plan de trabajo 10

3. Entorno y plataforma de desarrollo **13**

3.1. IDE XCode 13

3.2. Lenguaje Ruby 14

3.3. Framework Rails 16

3.4. MySQL 18

4. Descripción informática **19**

4.1. Contenido de la sección 19

4.2. Análisis funcional 20

4.2.1. Rol de usuario registrado 20

4.2.2. Rol de usuario no registrado 24
4.3. Diagrama MVC 25
4.4. Diseño de BBDD 28
4.5. Edición de encuestas 29
4.5.1. Nueva encuesta: selección del punto de partida 31
4.5.2. Editar el título 33
4.5.3. Crear y añadir una pregunta 34
4.5.4. Listado de preguntas de la encuesta temporal 35
4.5.5. Borrado de preguntas temporales 37
4.5.6. Listado de preguntas de la base de datos 38
4.5.7. Guardar la encuesta 39
4.5.8. Publicar la encuesta 42
4.5.9. Cancelar la edición 43
4.6. Rellenado de encuestas y obtención de resultados 43
4.6.1. Formulario de las encuestas 43
4.6.2. Recolección de las respuestas 46
4.6.3. Mostrando los resultados obtenidos 53

5. Pruebas **55**

6. Conclusiones y trabajos futuros **57**

6.1. Conclusiones 57

6.2. Trabajos futuros 59

6.3. Esfuerzo dedicado 60

Bibliografía **63**

Anexos **65**

A. Estado del arte: Google Docs **65**

B. Estado del arte: dudle **69**

C. Entidades del modelo relacional **73**

D. Evolución de la BBDD **78**

1. Introducción

En este primer capítulo introductor, hablaremos brevemente sobre estadística, sus orígenes, metodologías y necesidades, así como del papel que desempeña la tecnología en dicho ámbito. Concretamente, nos interesaremos por los métodos de obtención de datos basados en encuestas, que es el cometido del software desarrollado en el presente proyecto fin de carrera.

1.1. Estadística y encuestas

La estadística es una rama de la ciencia matemática de una relevancia innegable para el ser humano. Desde los antiguos egipcios ya se manifestaba la necesidad de controlar el número de habitantes y la producción de las cosechas, aunque con métodos rudimentarios y muy alejados de la estadística que conocemos hoy en día, basada en conteo.

La terminología moderna surgió durante la revolución industrial; con la invención de la producción en cadena, que llevó más adelante a la producción en masa, se hizo imprescindible controlar los índices de productividad de las fábricas, de modo que pudieran cuantificarse los beneficios y los inventarios. Una vez que los niveles de producción consiguieron satisfacer las necesidades de la sociedad, el nivel de vida aumentó exponencialmente, y el mayor poder adquisitivo de la población propició el comienzo de la economía de consumo, donde las personas pueden obtener recursos que no son directamente necesarios para su vida, pero sí pueden ser gratificantes. Este tipo de comercio planteó nuevos retos para las empresas: buscar necesidades no cubiertas de los consumidores, ofrecerles un producto que las satisfaga, y hacerlo mejor que la competencia para aumentar los beneficios. Fue en este contexto donde las encuestas empezaron a tener una gran importancia.

El uso de este sistema de obtención de información tiene una serie de importantes ventajas. Probablemente la más interesante de todas ellas está relacionada con la naturaleza misma de las encuestas; el hecho de contar con información, tanto subjetiva como objetiva, de un conjunto de personas, ofrece una visión más realista cercana a la realidad acerca del campo de estudio. Esto repercute en un mayor impacto a los datos obtenidos, amplificando el obtenido con la interpretación de los mismos que se ha dado al estudio. También en relación al proceso de encuesta propiamente dicho, esta práctica permite acotar una población para realizar un estudio sin que se vea necesariamente afectada la veracidad de los resultados; en

cambio, sí que se ve reflejado en un menor coste del estudio, tanto temporal como económico. Sin embargo, es importante remarcar que todas estas ventajas podrían resultar contraproducentes si se aplican de forma desmedida.

1.2. Procesado de la información

En el proceso de recopilado de la información, inicialmente el único modo era la entrevista personal, donde es necesario contar con entrevistadores que realiza la recogida de datos directamente con las personas de la muestra. Evidentemente, este sistema implica contar con personal dedicado a obtener la información directamente, con el coste económico y temporal para formación que ello supone. Otro de los métodos utilizados es el correo postal, mediante el cual el entrevistado puede realizarlo desde su propio domicilio. Evitamos de esta manera la necesidad de contar con personas especializadas, aunque surgen otro tipo de problemas: el índice de respuestas suele ser bastante bajo, y los datos recogidos pueden ser poco fiables en cuanto a pertenecientes a una muestra representativa.

El avance progresivo de las tecnologías ha permitido perfeccionar y mejorar las técnicas utilizadas para estos fines. Por ejemplo, la posibilidad de realizar entrevistas telefónicas se ha convertido en una herramienta más rápida que realizarlas en persona, aunque sigue siendo necesario contar con operadores especializados y mejor preparados. La aparición de las máquinas IVR (Interactive Voice Response) resuelve este último problema, realizando el mismo trabajo de un modo mucho más eficaz generalmente. Aresco Instant Research sería un ejemplo de empresa especializada en ofrecer este servicio.

Claramente se puede deducir que el uso de las tecnologías en este campo resulta muy beneficioso para obtener mejores resultados, y sobre todo para reducir costes. No obstante, aún existe un problema que no hemos comentado; la necesidad de tener un contacto directo y "síncrono" entre el entrevistador y el encuestado. Parece algo trivial, pero no deja de ser un *handicap* muy importante el encontrar el mejor momento para realizar una encuesta, y debe ser también considerado a la hora de buscar mejoras. Se logró solventar con el uso de las encuestas vía correo postal, aunque implica algunas limitaciones, comentadas anteriormente. Con el auge de internet y las nuevas tecnologías de comunicación, se abrió la puerta a realizar encuestas a través de correo electrónico y aplicaciones web.

No todo son ventajas en los escenarios mencionados hasta ahora. El hecho de realizar el proceso de una forma más autónoma conlleva un importante problema;

la deshumanización de las encuestas. A una persona, el hecho de tener que tramitar con una máquina, le resulta más molesto que interactuar con una persona directamente. Esto se ve agravado conforme independizamos en mayor grado las interacciones, siendo el IVR el máximo referente, traduciéndose en menores índices de respuestas a las encuestas.

1.3. Funcionamiento en la Universidad

En esta sección abordaremos el ámbito de nuestra universidad, y de qué manera se realizan las encuestas en la misma. En la Universidad Rey Juan Carlos se ha delegado toda esta responsabilidad en el Centro Universitario de Estudios Sociales Apilcados (CUESA[1] a partir de ahora). Se trata de un Centro Universitario dedicado a realizar diversos estudios económicos y sociales para múltiples empresas y entidades públicas, que además se encarga del procesado de las encuestas de la Universidad.

Actualmente, el proceso de recogida de información se lleva a cabo de dos formas diferentes. Se cuenta, por un lado, con una plataforma online, dedicada especialmente a las titulaciones que se imparten en la red y no cuentan con clases presenciales. Para ello se utiliza una aplicación online de similares características a la presentada en este proyecto. El profesor enviará una invitación a sus alumnos para que accedan a la encuesta y puedan rellenarla. La aplicación permite acceder sin introducir ningún tipo de acreditación. No obstante, en los cuestionarios se suele incluir un campo donde el alumno deberá indicar su clave de acceso. Esta información se utiliza con la finalidad de evitar duplicidades y falseo de información, y no llegará al profesor cuando se le muestren los resultados. Este método únicamente consigue un 10 % de participación del alumnado.

Por otra parte, también se cuenta con un medio más tradicional basado en encuestas presenciales. Se concreta un calendario de fechas con los profesores de todas las asignaturas, y se realizan los cuestionarios directamente en un aula. Para ello, se contrata personal de una empresa externa, que son los encargados de llevar las hojas de preguntas y respuestas, las recogen una vez cumplimentadas, y las llevan de vuelta al CUESA para su procesado. En la figura 1 puede verse una hoja de respuestas utilizada en estas encuestas presenciales. Se basa en la ubicación precisa de las casillas donde se marca la opción a escoger, de modo que una máquina específica es capaz de digitalizar la información. Este proceso tarda

[1] http://www.cuesa.urjc.es

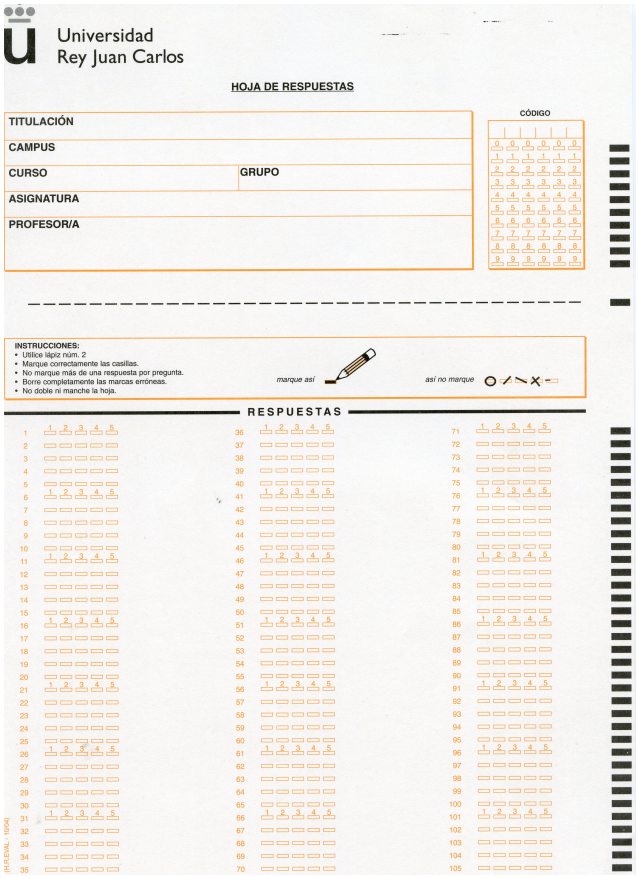


Figura 1: Página de edición de encuestas.

en torno a 15 horas para 80.000 formularios, que es aproximadamente lo que se recoge en las épocas de evaluación de profesores. Siguiendo este procedimiento, se consigue un porcentaje del 99 % de participación.

Para ambos modos de obtención de información, la Comisión Vicerrectora de Formación Académica se encarga de decidir qué preguntas se incluirán en las encuestas. En este aspecto se han producido cambios importantes, pasando de las 40 preguntas iniciales a únicamente 10 en el caso de una encuesta presencia, y 9 de hacerse online. Entre los motivos de la reducción, se encuentra la optimización de la información obtenida con las respuestas, quedándose al escoger las preguntas que contienen información más relevante. También se ha creado una encuesta adicional que contiene cuestiones relacionadas con las instalaciones de la Universidad, que se realiza una única vez anual para todas las asignaturas.

1.4. Estado del arte

Para concluir esta sección de apertura de la presente memoria, en este apartado se detallará un estado del arte. La intención es analizar algunas aplicaciones actualmente existentes en la red, que ofrezcan una funcionalidad similar a la que se pretende implementar. De esta manera, observando las ventajas e inconvenientes de cada una de ellas podremos mejorar la aplicación desarrollada para intentar mejorar el resultado final.

Concretamente, en este caso nos centraremos en dos aplicaciones diferentes. Una de ellas será Google Docs, parte de la *suite* ofimática en la red del gigante Google. La hemos elegido por su amplia trayectoria en el sector, que le han permitido ampliar enormemente sus funcionalidades. Entre ellas, destaca la posibilidad de seleccionar el tipo de respuesta para las preguntas, de un modo similar a como lo consigue este proyecto. Siguiendo con características que también se dan en nuestro sistema, el editor de encuestas se basa en tecnologías web dinámicas, como por ejemplo la variable cantidad de preguntas que se muestran por pantalla, añadiendo o eliminando elementos de la página web sin necesidad de recargar la vista completa. Además, permite mostrar los resultados en dos posibles formatos: de modo resumido, o como hoja de cálculo para trabajar cómodamente con los datos. También existen algunas carencias en comparación con este proyecto, como la ausencia de una base de datos con preguntas utilizadas anteriormente. En el apéndice A de la presente memoria, se detalla la funcionalidad que ofrece este servicio en cuanto a creación y explotación de encuestas.

Por otra parte, se ha analizado la aplicación dudle, también presente actualmente en la red. Esta aplicación, a pesar de ser más modesta, presenta varias diferencias de funcionamiento con respecto a la solución de Google, por lo que hemos considerado apropiado incluirla en este apartado. En primera instancia, se elimina la necesidad de estar registrado para poder crear una encuesta, lo cual es muy práctico para usos esporádicos. También cabe destacar un control de acceso mediante credenciales, diseñado para que los usuarios no tengan que estar registrados en la plataforma para restringir el acceso. Adicionalmente, podremos comprobar las carencias presentes que se podrán solucionar en la implementación del proyecto. Todos los detalles se pueden encontrar en el anexo B de esta memoria.

2. Objetivos

Tras haber tratado el trasfondo en el que se sustenta el presente proyecto fin de carrera, pasaremos a analizar el propósito perseguido: diseñar e implementar un servicio web de encuestas.

2.1. Descripción del problema

Tal y como se ha tratado en el apartado 1.3., actualmente en la Universidad las encuestas de satisfacción se llevan a cabo de forma presencial en las aulas de clase: los alumnos reciben una hoja de respuestas, y otra con las diversas preguntas que aplican en la asignatura concreta que se está valorando. Este procedimiento no es el más apropiado para este ámbito, en el que existe mucho potencial para optar a un porcentaje de participación muy elevado al tener muy definida a la población objetivo del estudio.

Limitar la participación a los alumnos presentes en un aula en un momento determinado es una imposición bastante importante para el alcance de los datos recogidos. Fuera del estudio quedarían alumnos que, a pesar de no seguir las clases de manera presencial, tienen derecho a manifestar su opinión acerca de los contenidos o el material suministrado, y cualquier otro caso de similares características. Además, es necesario interrumpir una clase para permitir a los alumnos rellenar los formularios, perdiendo horas lectivas; esto es necesario únicamente por el mismo hecho de realizarlas en persona.

Debemos reseñar también el problema que supone la escasa flexibilidad de las preguntas de la encuesta. Dado el método escogido para el procesamiento de los resultados, necesariamente el formato de la hoja de respuestas debe ser siempre el mismo, con cinco posibles opciones a elegir por cada pregunta. Esto supone una fuerte restricción a la hora de elaborar el cuestionario de una encuesta; si se pretende, por ejemplo, incluir preguntas de libre respuesta, será necesario analizar los datos recogidos de forma manual, algo poco práctico si el alumnado es numeroso. También se elimina la posibilidad de incluir material de apoyo en la encuesta, como multimedia, que podría resultar realmente interesante.

Con la finalidad de abarcar y resolver todos los problemas descritos en esta sección, se llevará a cabo el desarrollo de una aplicación web para editar y explotar encuestas de forma online. Proporcionaremos las herramientas necesarias para

crear encuestas, completamente nuevas o en base a otras previas. Dicha creación será todo lo versátil posible para incluir varios tipos de preguntas. Y también se gestionará la recogida de resultados una vez publiquemos nuestras encuestas, de modo que los usuarios puedan consultar los datos obtenidos para realizar evaluaciones. De esta manera, conseguiremos solventar todos los problemas descritos en este apartado:

- Por un lado, queda patente que, al tratarse de una aplicación en la red, ampliaremos el rango de acción de las encuestas más allá de los alumnos presentes en un aula;
- Al no realizarse de forma presencial, también se podrán mantener las horas lectivas y aprovecharlas para impartir más contenidos;
- Por último, pero no por ello menos importante, al trabajar con una aplicación web se ampliará la versatilidad de las encuestas, pudiendo personalizar el contenido con muchas más posibilidades.

2.2. Requisitos

El objetivo del software desarrollado para este proyecto fin de carrera será, además de satisfacer todas las necesidades descritas en el apartado anterior, respetar los siguientes requisitos funcionales:

1. Requisitos generales
   - 1.1. La aplicación debe ser capaz de gestionar por sí misma todas las fases en el desarrollo de una encuesta: desde su concepción, al introducir las preguntas y personalizar las posibles respuestas a elegir; hasta su puesta en producción, para que los alumnos puedan responder a las cuestiones; y la recogida de información para uso del profesor.
   - 1.2. Diseñar el soporte de datos necesario para el funcionamiento de la aplicación.
   - 1.3. Mantener, en la medida de lo posible, el anonimato de los encuestados.
2. Creación de encuestas
   - 2.1. Permitir la creación tanto de encuestas completamente nuevas, como a partir de otras anteriormente creadas.

2.2. Ofrecer las herramientas necesarias para editar completamente una encuesta.

2.3. Almacenar las encuestas en cualquier momento para poder abandonar y reanudar la edición sin problemas.

2.4. Revertir los cambios durante la edición si se desea cancelar el proceso.

2.5. Publicar las encuestas para ponerlas al alcance de los demás usuarios.

2.6. Una vez que una encuesta es publicada, evitar que pueda editarse. No puede suceder que dos usuarios diferentes contesten a dos versiones distintas de la misma encuesta; sí se permite crear una nueva encuesta partiendo de aquella publicada.

2.7. Garantizar que dos encuestas no puedan tener el mismo título.

3. Creación de preguntas

3.1. Disponer un editor para generar preguntas nuevas.

3.2. Facilitar la inclusión de preguntas almacenadas en una base de datos, de modo que puedan reutilizarse.

3.3. Gestionar la eliminación de preguntas de la encuesta que se está creando.

3.4. Permitir la ordenación de las preguntas de la encuesta.

3.5. Realizar el almacenamiento de las preguntas en base de datos.

3.6. Controlar que no puedan existir dos preguntas idénticas, ni en la encuesta que se está editando, ni en la base de datos. Por idénticas entenderemos que tengan el mismo enunciado.

4. Obtención de respuestas

4.1. Distinguir entre varios tipos de preguntas, en base a su modelo de respuesta. Se dispondrá de los siguientes:

4.1.1. Preguntas de respuesta única.

4.1.2. Preguntas de respuesta múltiple.

4.1.3. Preguntas de respuesta libre.

4.2. Mostrar adecuadamente los formularios de respuesta para poder rellenarlos.

4.3. Permitir el acceso a las encuestas publicadas a través del portal y mediante una URL.

4.4. Tramitar la recogida de las respuestas introducidas por los usuarios.

4.5. Presentar los resultados de una encuesta al usuario que la publicó.

5. Herramientas de desarrollo

   5.1. Realizar el desarrollo bajo el *framework* Ruby On Rails, que utiliza el lenguaje Ruby como base. Puede encontrarse una descripción más detallada en los apartados 3.2 y 3.3 del siguiente capítulo. Este requisito nos determinará la metodología a seguir para el desarrollo de la aplicación.

   5.2. Diseñar una interfaz con efectos y animaciones lejos del HTML plano. Para ello, se necesitará de forma esporádica el uso de Javascript y las librerías AJAX para aquellas implementaciones que el *framework* de desarrollo no pueda cubrir por sí solo.

   5.3. Trabajar con MySQL como gestor de bases de datos.

2.3. Metodología y plan de trabajo

Para la implementación del presente proyecto se ha trabajado en base a un modelo de desarrollo espiral basado en prototipos. Este modelo consiste en llevar a cabo una serie de tareas de forma prácticamente secuencial y cíclica un número determinado de veces. Con cada una de las iteraciones, se obtiene un prototipo funcional con una porción de las características del producto final. A fin de llevar un control periódico en el desarrollo, se ha mantenido un contacto vía correo electrónico semanal con el tutor, y reuniones presenciales para presentar prototipos completos.

Los ciclos que han determinado el desarrollo del proyecto han sido los siguientes:

- *Formación en los lenguajes de programación y fundamentos del paradigma.* Esta primera fase tiene por objetivo conocer las herramientas que se van a emplear: aprender la sintaxis y posibilidades de las aplicaciones escritas en Ruby, y comprender la metodología de desarrollo de aplicaciones mediante este entorno. Se realizan algunas prácticas y pruebas para afianzar conceptos.

- *Desarrollo de controladores y vistas.* En esta fase se dará forma realmente a las funcionalidades que podrán disfrutar los usuarios. Se engloba tanto la apariencia de la aplicación y sus diferentes elementos, como la lógica que trabaja por detrás, dada su estrecha relación en este tipo de soluciones software.

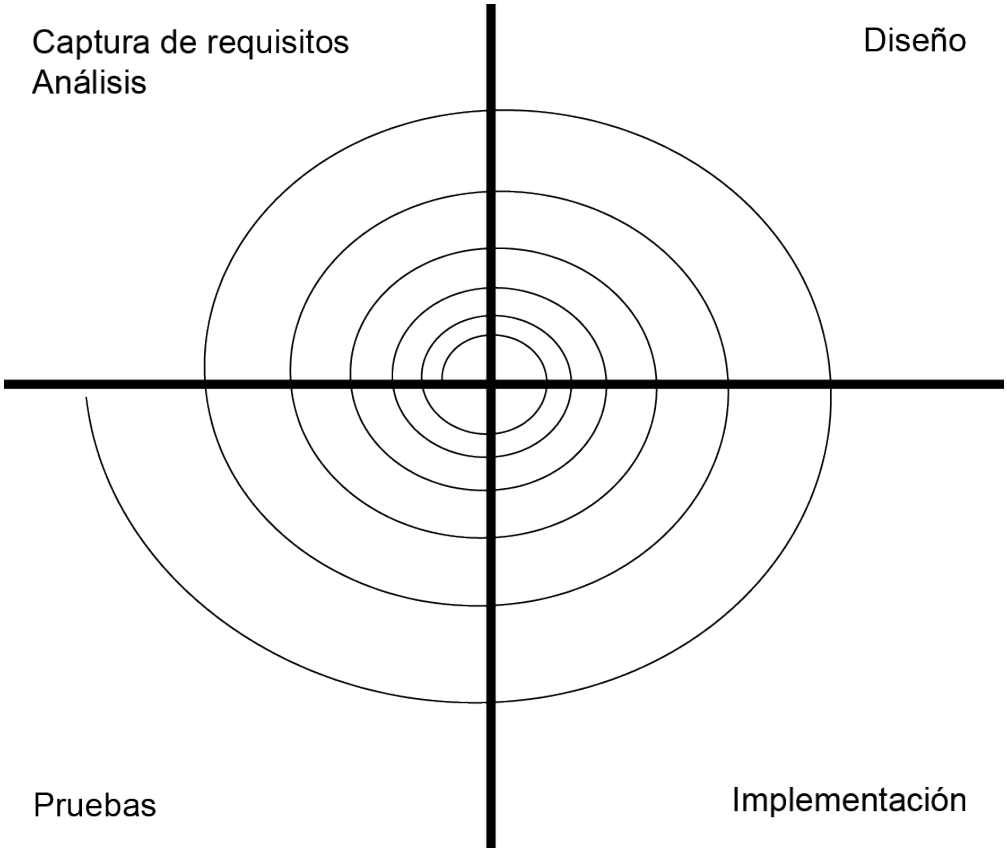


Figura 2: Modelo de desarrollo en espiral basado en prototipos.

- *Estructuración de la base de datos.* A la hora de aplicar nuevas capacidades al software, se demandarán nuevas relaciones y entidades en la estructura de datos de la aplicación que se tendrán que definir. Así mismo, será necesario adecuar los modelos para interpretar las bases de datos y hacerlos manejables por el resto de elementos del sistema.

Como resultado de utilizar esta metodología de desarrollo, obtendremos dos prototipos antes de la versión definitiva de la aplicación. El desarrollo del primer prototipo contemplará la siguiente funcionalidad:

- Control de acceso de los usuario mediante nombre de usuario y contraseña.
- Editor de encuestas, capaz tanto de crear encuestas completamente nuevas, como de editar las ya existentes, o partir de una encuesta previa.

- Añadir preguntas residentes en la base de datos, o crear elementos nuevos.
- Funcionamiento mediante arrastrar y soltar elementos en diferentes áreas de las vistas.
- Guardar encuestas en estados intermedios para poder editarlas.
- Cancelar la edición de una encuesta, deshaciendo todos los cambios.
- Publicar encuestas para su resolución por parte de otros usuarios.
- Presentar la encuesta para rellenarla.
- Mostrar los resultados recogidos al usuario que creó la encuesta.

Como puede observarse, esta primera versión del sistema ya contará con prácticamente todas las características requeridas para este proyecto. En cuanto al segundo prototipo, incluirá los distintos tipos de respuestas a las preguntas, además de todas las características del prototipo anterior. Por último, la versión final irá dirigida a algunos últimos ajustes, especialmente orientados a mejorar la interfaz.

En la sección 6 de conclusiones podremos ver en detalle el proceso de desarrollo de los prototipos aquí descritos.

3. Entorno y plataforma de desarrollo

En esta sección se darán a conocer las herramientas utilizadas para la implementación del presente proyecto: XCode, un IDE multilenguaje de Apple; el lenguaje de programación Ruby, interpretado y orientado a objetos; el framework Rails, diseñado para trabajar con aplicaciones web; y el gestor de bases de datos MySQL, para almacenar toda la información utilizada por el sistema.

3.1. IDE XCode

XCode es un entorno de desarrollo integrado (o IDE: Integrated Development Environment) desarrollado por Apple Inc., preparado para la programación de aplicaciones en múltiples lenguajes. Fundamentalmente, está diseñado para dar soporte a los desarrolladores de las plataformas propias de la compañía: el sistema operativo de ordenador Mac OS X, o el más reciente de dispositivos móviles iOS. En ambos casos, la plataforma tiene más recursos disponibles, como un editor de interfaces gráficas, o un acceso directo a los manuales de referencia, que evidentemente no existen para un desarrollo como el que ocupa esta memoria. En la figura 3 se muestra una captura de pantalla de la aplicación.

A pesar de este enfoque, XCode sigue ofreciendo muchas herramientas para los desarrollos en otros lenguajes. Por ejemplo, contiene un versátil gestor de proyectos que permite acceder fácilmente a los recursos del mismo. En el caso que nos ocupa, al comenzar un proyecto nuevo, Ruby genera un árbol de directorios sobre el que se asentarán todos los recursos de nuestra aplicación web, desde los controladores y los modelos de datos, hasta las múltiples páginas web que componen las vistas. XCode brinda la posibilidad de gestionar y acceder directamente a todos los ficheros desde la misma aplicación, facilitando la edición de múltiples archivos.

Por otra parte, también dispone de varias herramientas de implementación y mantenimiento de código, como coloreado de sintaxis para Ruby y html.erb (el formato de vistas que combina HTML y fragmentos de Ruby), aunque para conseguirlo sea necesario modificar un fichero de propiedades del IDE; o la encapsulación de bloques de código, permitiendo ocultar secciones para centrarnos en la que nos interesa.

Como se verá a continuación, Ruby es un lenguaje no compilado, lo que implica que los errores han de detectarse y depurarse en tiempo de ejecución de la aplica-

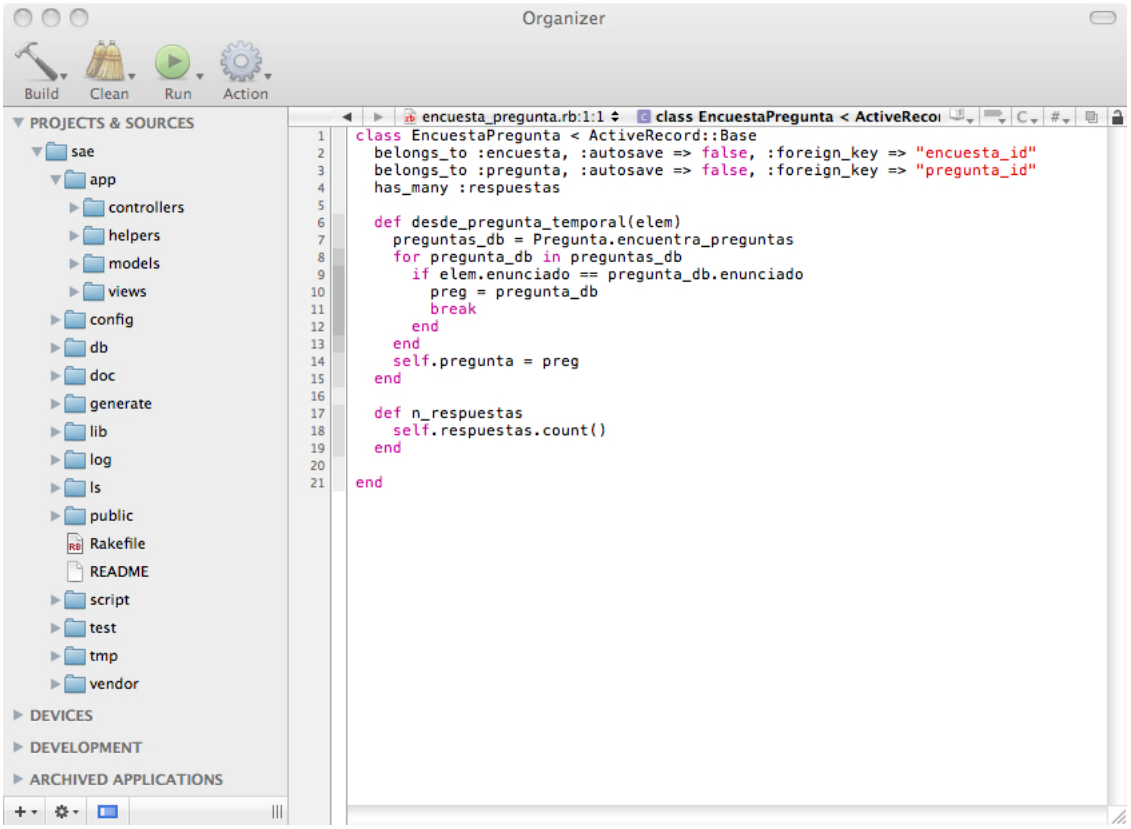


Figura 3: Organizador de proyectos de XCode

ción. Al tratarse de una aplicación web, para las ejecuciones es necesario simular el entorno de un servidor web, con base de datos incluida, y lanzar consultas a las páginas utilizando un navegador web. Por tanto, no existe necesidad de contar con un software más especializado en Ruby para realizar los desarrollos, pudiendo utilizar el que uno maneje con más soltura, o simplemente el que tenga disponible más fácilmente, como ha sido este caso al utilizar XCode 3.2.6.

3.2. Lenguaje Ruby

Ruby[2] es un lenguaje de programación interpretado multiplataforma de origen japonés, con licencia de software libre. Su creador, Yukihiro Matsumoto, recogió las partes que él consideró más interesantes de múltiples lenguajes (Perl, Small-

[2]http://www.ruby-lang.org/es/about/

talk, Eiffel, Ada y Lisp) con la intención de inventar un lenguaje sencillo que pudiese combinar programación funcional e imperativa. Desde su lanzamiento en 1995, ha ido adquiriendo popularidad dada su facilidad de uso y múltiples ámbitos potenciales de aplicación. En palabras del propio creador:

> "Ruby es simple en apariencia, pero complejo por dentro, como el cuerpo humano."

Uno de sus mayores atractivos es la profunda integración del paradigma de orientación a objetos que presenta [2]; todos los tipos de datos en Ruby se comportan como objetos, incluyendo tipos considerados primitivos como los números. De este modo, se pueden aplicar métodos y envío de mensajes prácticamente a cualquier variable que tengamos en nuestro código, exprimiendo al máximo esta filosofía de programación. Variables que no es necesario definir, ya que su ámbito de uso viene determinado por la sintaxis utilizada en su identificador [2]:

- *variable*: variables locales.
- *@variable*: variables de instancia.
- *$variable*: variables globales.

Otra de las grandes ventajas de este lenguaje es su flexibilidad, al permitir diversas formas de atacar un problema, y retocar sin complicaciones una implementación para optimizarla. Un ejemplo de esta flexibilidad es la definición de bloque [2]: la posibilidad de anexar cualquier conjunto de instrucciones a un método para que se ejecute de forma iterativa, como puede verse en el fragmento de muestra contenido en la figura 4.

El bloque se encuentra definido entre las cláusulas *do* y *end*. En este caso, se encarga de generar las preguntas en base de datos una vez que hemos decidido guardar cambios en una encuesta que estamos editando (*temp* es una variable que almacena las preguntas temporales que tiene la encuesta en un momento determinado). Definido dentro del método *each*, nos genera un bucle de tipo *for* que automáticamente aplica la secuencia de instrucciones a cada uno de los elementos de las iteraciones. Como se puede observar, la apariencia del código es muy limpia, de fácil comprensión lectora.

```ruby
def anhade_pregunta_encuesta_desde_temporal(temp)
  temp.preguntas.each do |elem|
    if elem.genera_pregunta_db
      pe = EncuestaPregunta.new()
      pe.desde_pregunta_temporal(elem)
      encuesta_preguntas << pe
    end
  end
end
```

Figura 4: Bloque en Ruby

3.3. Framework Rails

Rails [1] (también conocido como Ruby on Rails, dada su relación con el lenguaje) es un *framework* diseñado para permitir el desarrollo de aplicaciones web basadas en Ruby, aprovechando todas sus cualidades descritas en el apartado anterior. Se trata, por tanto, de una herramienta más sencilla de utilizar para este tipo de desarrollos que las convencionales (PHP, .NET). Para lograrlo, se sustenta en varios principios: por un lado, es una plataforma DRY (Don't Repeat Yourself), que consiste en tener muy bien definidas las zonas donde debe estar implementado un fragmento de código, de modo que no exista duplicidad. De esta manera, cualquier modificación que realicemos a nuestra aplicación afectará a una porción menor de código. Es, por tanto, un framework que respeta la modularidad del código de forma muy estricta.

El esfuerzo que se realiza con este framework para respetar DRY y la no repetición de código se basa en su filosofía de construcción de aplicaciones. Se utiliza siempre el paradigma MVC (Modelo, Vista, Controlador), que distingue entre tres tipos de elementos:

- *Modelo*: es la parte de la aplicación que se encarga de gestionar los recursos persistentes; en nuestro caso, se trata de la interfaz de la base de datos para el resto del sistema. Es la única parte de la aplicación que debería tener un acceso directo a estos recursos para garantizar la modularidad.
- *Vista*: es la interfaz que se presenta al usuario para interactuar con el contenido de la aplicación. También es responsable de mostrar los datos que se almacenan en el modelo, aunque no está permitido hacerlo de forma directa.

- *Controlador*: es la lógica propiamente dicha de la aplicación. Contiene el funcionamiento real que debe tener lugar cuando se realiza una acción en la vista, y obtiene los datos del modelo para que la vista pueda utilizarlos. Por tanto, el controlador es el nexo de unión entre vista y modelo.

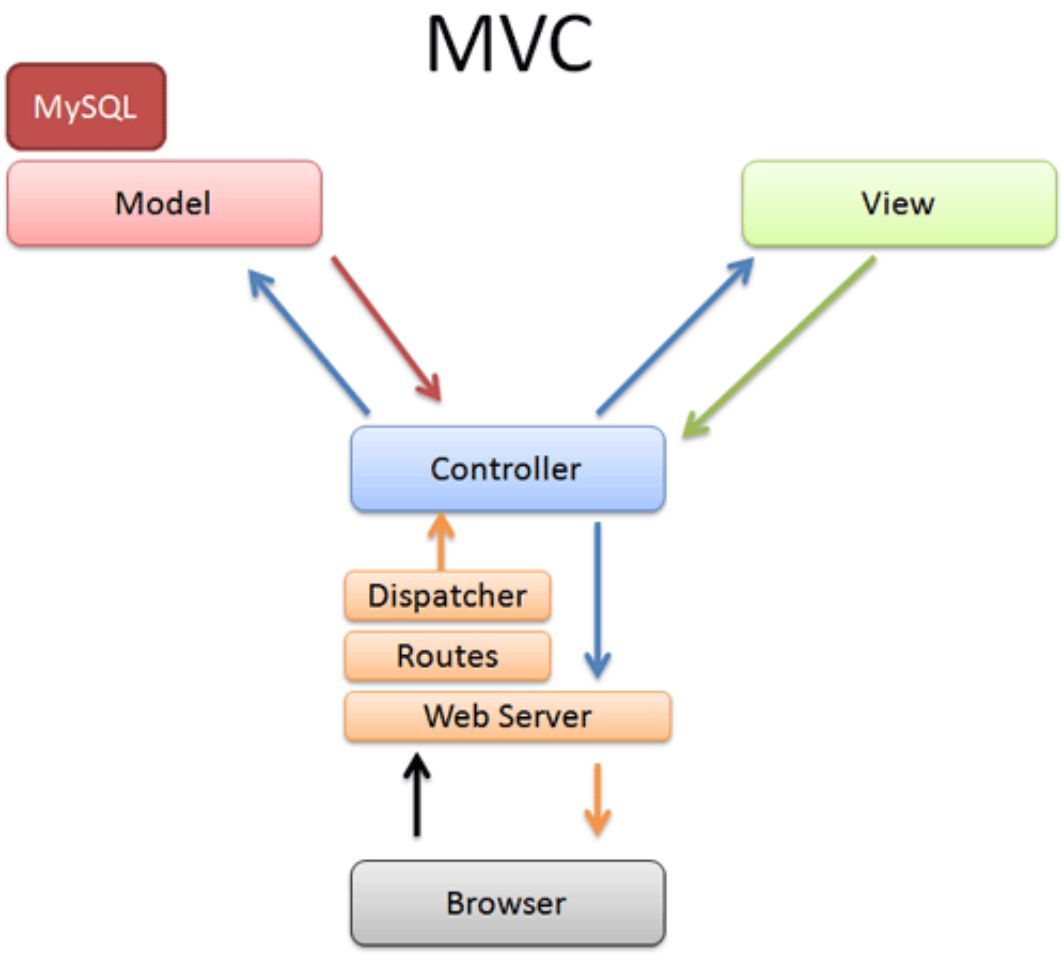


Figura 5: Esquema MVC en una aplicación hecha en Rails.

Como puede observarse en la figura 5, el usuario, a través de su navegador web, al interactuar con la vista que tiene presente, produce el envío de una petición al servidor web que tiene alojada dicha aplicación. Éste la hace llegar al controlador pertinente, que se encarga de gestionar el paso de información entre un modelo determinado y la vista a mostrar. Este esquema podría considerarse un fragmento

de una aplicación real, ya que este conjunto de módulos puede replicarse tantas veces como sea necesario para hacer más complejo el sistema.

Rails también permite el uso de tecnologías avanzadas para la creación de una aplicación web, con un alto grado de integración con el propio lenguaje. Una de las más utilizadas es Ajax; un conjunto de herramientas que permiten crear páginas web más dinámicas, con contenido variable de forma autónoma y asíncrona con el servidor, de forma que, para recargar una sección específica de una página determinada, no es necesario obtener una nueva; simplemente se altera la parte deseada. Esto, evidentemente, repercute en un menor consumo de recursos, y una presentación más interesante.

Finalmente, destacaremos la interfaz que ofrece Rails para la manipulación de bases de datos. De forma nativa, viene preparado para trabajar con SQLite, aunque es fácilmente configurable para utilizar MySQL, como en el presente proyecto. En cualquier caso, e independientemente de qué gestor de base de datos tengamos operativo, el framework ofrece el concepto de migración para trabajar de forma transparente, sin necesidad de utilizar consultas SQL. Una migración es una modificación sobre una base de datos, ya sea crear una entidad, incluir o retirar atributos, eliminar tablas, etc., implementadas como ficheros en Ruby. Al tener un registro de todas las migraciones que hemos llevado a cabo, se consigue un eficaz sistema de versiones; es extremadamente sencillo aplicar una mejora o efectuar una marcha atrás en una base de datos para modificar su esquema entidad-relacion, únicamente con un par de instrucciones.

3.4. MySQL

MySQL[3] es un sistema gestor de bases de datos de licencia libre GPL, con gran prestigio a nivel mundial, desarrollado por Oracle. Escrito en C y C++, cuenta con múltiples funcionalidades y características[4] que lo convierten en una interesante alternativa al gestor principal del mismo nombre que la empresa, Oracle.

Para este proyecto, se utiliza MySQL Community Server en su versión 5.1.51

[3]http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html
[4]http://dev.mysql.com/doc/refman/5.1/en/features.html

4. Descripción informática

Tras haber explicado los objetivos de este proyecto y las herramientas a utilizar, en este capítulo se expondrá la solución implementada en todas sus fases, desde el diseño global hasta las pruebas de funcionamiento.

4.1. Contenido de la sección

A lo largo de este capítulo abordaremos los siguientes temas:

- En la sección 4.2 se detalla el análisis funcional con diagramas de casos de uso para los dos roles de usuario existentes.
  - Rol de usuario registrado en la sección 4.2.1.
  - Rol de usuario no registrado en la sección 4.2.2.
- En la sección 4.3 se presenta la estructura MVC sobre la que se sustenta la aplicación (requisitos 1.1. y 5.1.).
- En la sección 4.4 estudiaremos el diseño de la base de datos relacional subyacente (requisitos 1.2. y 5.3.).
- En la sección 4.5 desglosaremos la implementación del editor de encuestas.
  - En la sección 4.5.1 se explica la gestión del origen de las encuestas (requisito 2.1.).
  - En la sección 4.5.2 estudiaremos la edición del título de la encuesta, implementado como una etiqueta de texto editable (requisitos 2.2. y 5.2.).
  - En la sección 4.5.3 veremos la implementación del editor de nuevas preguntas, como un formulario oculto desplegable (requisitos 4.1. y 5.2.).
  - En la sección 4.5.4 veremos la presentación de las preguntas pertenecientes a la encuesta y la funcionalidad relacionada: inserción de preguntas nuevas, inserción de preguntas almacenadas en base de datos y ordenación de las preguntas insertadas, todo ello mediante movimiento de elementos por la interfaz (requisitos 3.2., 3.4., 3.6. y 5.2.).

    - En la sección 4.5.5 se explicará cómo se realiza el borrado de preguntas de la encuesta, arrastrándolas a una zona determinada (requisitos 3.3. y 5.2.).
    - En la sección 4.5.6 desarrollaremos la presentación de las preguntas almacenadas en la base de datos, de modo que puedan ser utilizadas en la edición de una encuesta (requisito 3.2.)
    - En la sección 4.5.7 veremos el almacenamiento de la encuesta y sus preguntas en la base de datos (requisitos 2.3., 2.6., 2.7., 3.5. y 3.6.).
    - En la sección 4.5.8 detallaremos el cambio de estado de una encuesta a "publicado", para permitir a los usuarios responderla (requisitos 2.5., 2.6. y 4.3.).
    - En la sección 4.5.9 veremos cómo se va a gestionar la posible cancelación durante la edición de una encuesta (requisitos 2.4.).
- En la sección 4.6 abordaremos la presentación de la encuesta al usuario.
    - En la sección 4.6.1 veremos el diseño de la página donde se mostrará la encuesta (requisitos 1.3. 4.2. y 4.3.).
    - En la sección 4.6.2 se detalla la recogida de los datos introducidos por los usuarios como respuesta a una encuesta (requisitos 1.3. y 4.4.).
    - En la sección 4.6.3 se describe cómo se presentan de una manera adecuada los resultados obtenidos de la encuesta publicada (requisitos 1.3. y 4.5.).

4.2. Análisis funcional

Antes de comenzar con la explicación en profundidad de todos los elementos implementados para este proyecto, dedicaremos esta sección a detallar qué se espera funcionalmente de la aplicación. Para ello, estudiaremos los casos de uso disponibles para cada uno de los roles de usuario, y veremos una ejecución de ejemplo.

4.2.1. Rol de usuario registrado

En nuestra aplicación, los usuarios registrados serán aquellos que podrán acceder a toda la funcionalidad implementada. Por tanto, podrán crear encuestas,

editarlas, publicarlas, responder a encuestas de otros usuarios creadores, y comprobar los resultados que se han obtenido de sus encuestas. En la figura 6 podemos ver el diagrama UML representando todos los casos de uso existentes.

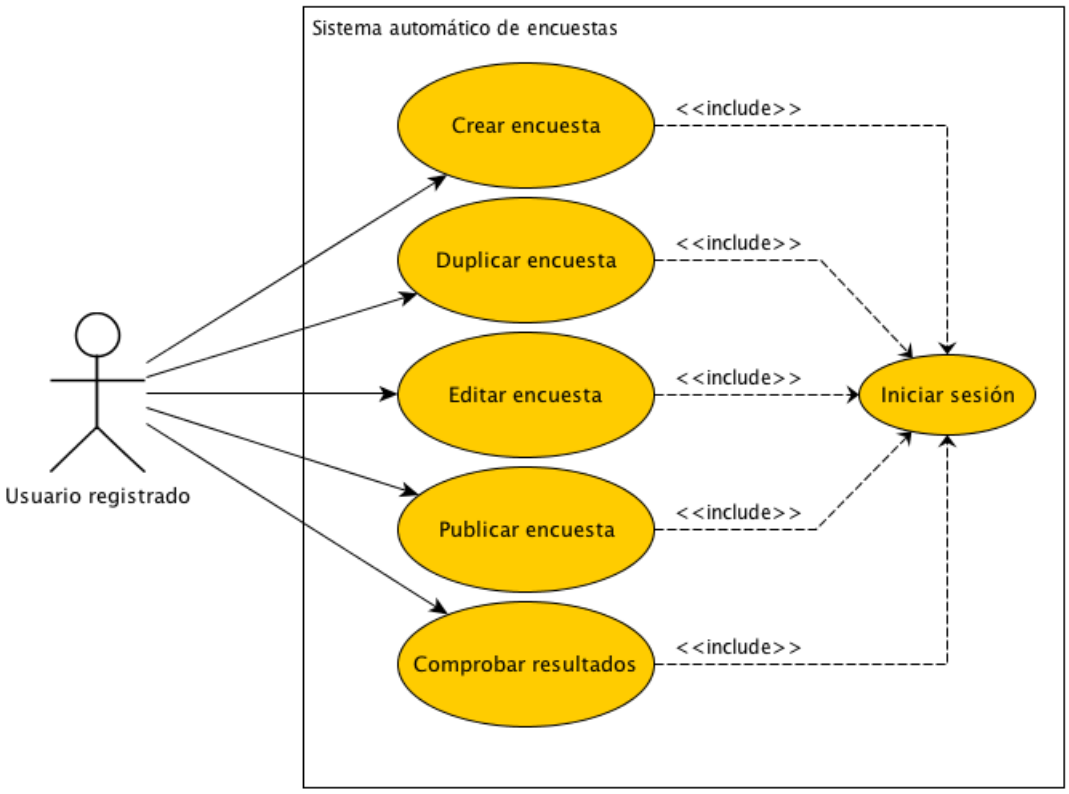


Figura 6: Diagrama de casos de uso para un usuario registrado.

Nuestro usuario anónimo inicia sesión en la aplicación con sus credenciales, utilizando el formulario de acceso de la página principal. En este caso, el usuario ya había entrado con anterioridad, creando y publicando una encuesta, como se puede ver en la figura 7. Esto le brinda la posibilidad de crear una encuesta utilizando aquella como plantilla; en este caso de ejemplo, en cambio, utilizara el botón "Crear", que producirá una encuesta temporal vacía.

En la figura 8, observamos que efectivamente la encuesta no contiene preguntas por el momento, aunque sí disponemos de varias en el área de la base de datos; esas son las preguntas que están en la otra encuesta y que se encuentran disponibles desde que se guardaron sus cambios. No encontrando la pregunta que tiene en mente para su nueva encuesta, activa el editor de preguntas nuevas, rellena los

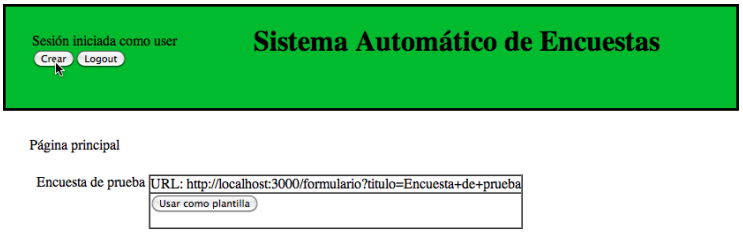


Figura 7: Vista principal tras iniciar sesión.

campos y la crea.

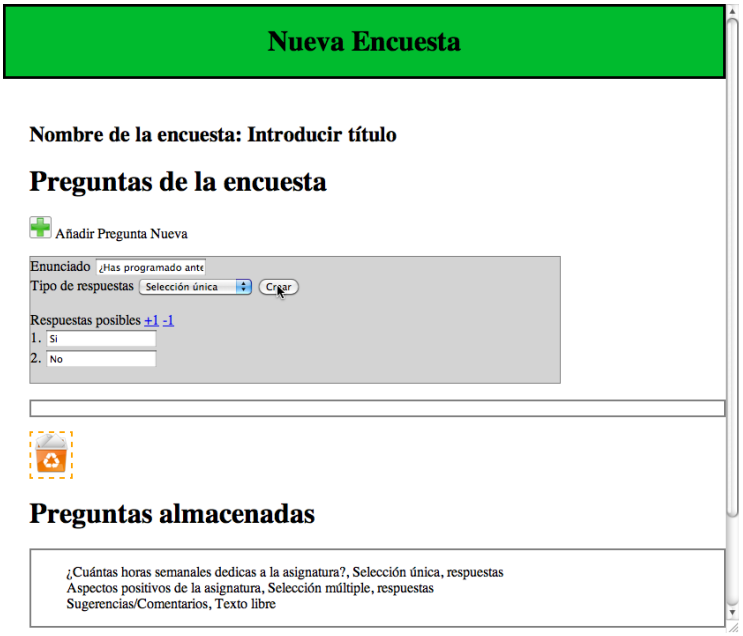


Figura 8: Editor de encuestas generando una pregunta.

Sin embargo, el usuario empieza a pensar que le resultaría más cómodo trabajar en base a la anterior encuesta, ya que realmente necesita utilizar todas las preguntas anteriores. Decide cancelar la edición y comenzar de nuevo, esta vez utilizando la encuesta anterior como plantilla. Por tanto, pulsa el botón “Cancelar” para eliminar todos los datos del modelo temporal (en este caso, la pregunta que ya tenía almacenada) y devolver el sistema a su estado original. En la figura 9 se muestra una ventana emergente, que aparece al cancelar la edición de la encuesta.

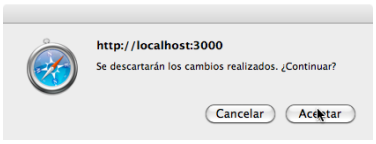


Figura 9: Cuadro de confirmación para eliminar la plantilla temporal.

Tras solicitar confirmación de la acción, la aplicación retorna al punto de partida. En la figura 10 podemos ver cómo se muestra información acerca de la actividad reciente, recuadrada en rojo; es una zona definida con el identificador de etiquta * *, que mostrará diversas notificaciones de las acciones del usuario. El usuario pulsa el botón para comenzar en base a la otra encuesta, de modo que se cargarán de forma automática todas las preguntas que tenía la original.

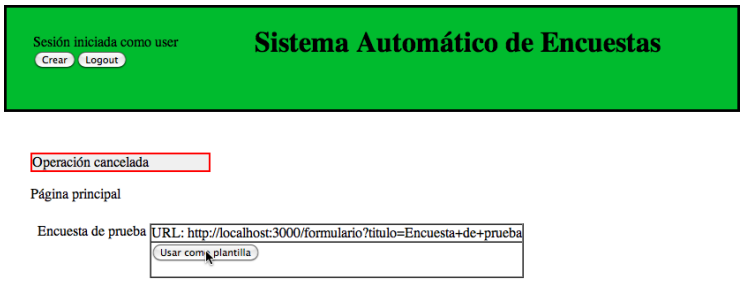


Figura 10: Vista principal tras cancelar la anterior encuesta.

En la pantalla de edición añade la pregunta que había creado en el anterior borrador y pulsa el botón "Publicar"para guardar la nueva encuesta y publicarla directamente para que los visitantes puedan rellenar con sus datos. Como podemos observar en la figura 11, se ha incluido una nueva entrada con la encuesta. Como el usuario no ha modificado el título de la misma, encontramos por defecto Çopia de encuesta de prueba", al tratarse de una versión basada en el otro formulario. El usuario podrá ahora utilizar la URL generada por el sistema para informar a sus encuestados de que ya se encuentra disponible la nueva encuesta. Dicha parte de esta demostración la veremos en el siguiente apartado.

Tal y como vemos en la figura 12, al volver a conectarse a la aplicación al cabo de un rato, el usuario puede ver que su nueva encuesta cuenta con algunos datos ofrecidos por algún visitante. Ahora podrá acceder a los resultados de su encuesta para obtener las conclusiones que necesita.

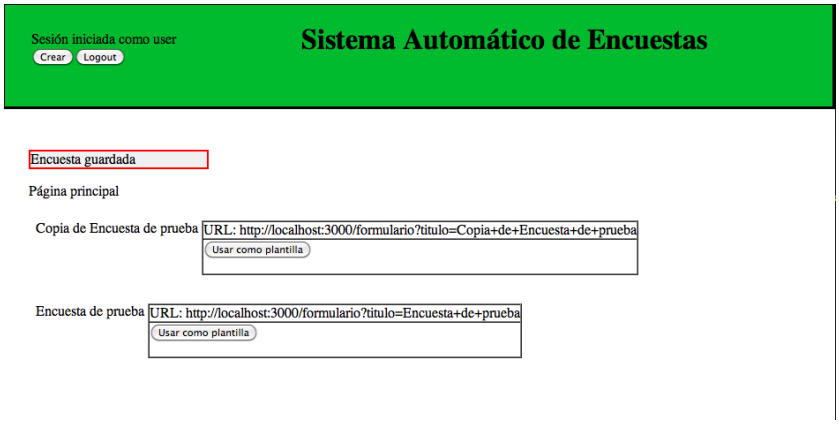


Figura 11: Vista principal tras guardar la nueva encuesta.

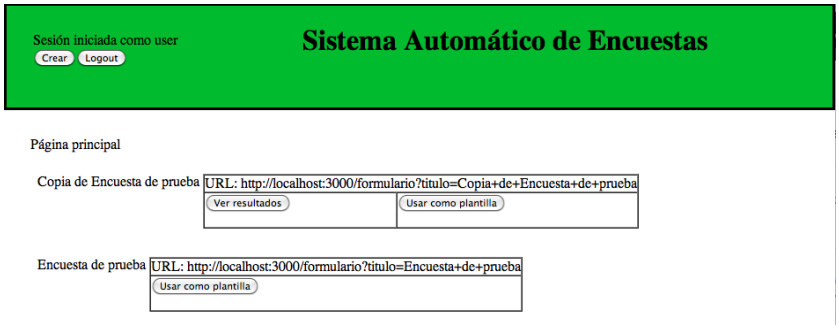


Figura 12: Vista principal con resultados en una de las encuestas.

4.2.2. Rol de usuario no registrado

A diferencia del usuario registrado, los que no cuentan con credenciales de acceso tendrán un perfil de uso restringido de la aplicación. Únicamente podrán acceder a la plataforma para rellenar las encuestas que los usuarios creadores hayan hecho públicas. En la figura 13 se muestra el diagrama UML correspondiente.

Un usuario no registrado, al acceder a la página web de la aplicación, se encuentra con una nueva encuesta disponible para rellenar. Al pulsar sobre el botón correspondiente, se cargan todas las preguntas que contiene el cuestionario, además de los distintos selectores de respuestas de acuerdo al tipo de pregunta del que se tratase. Recordemos que, en caso de acceder a través de la URL que nos hubiesen facilitado, el aspecto sería completamente idéntico.

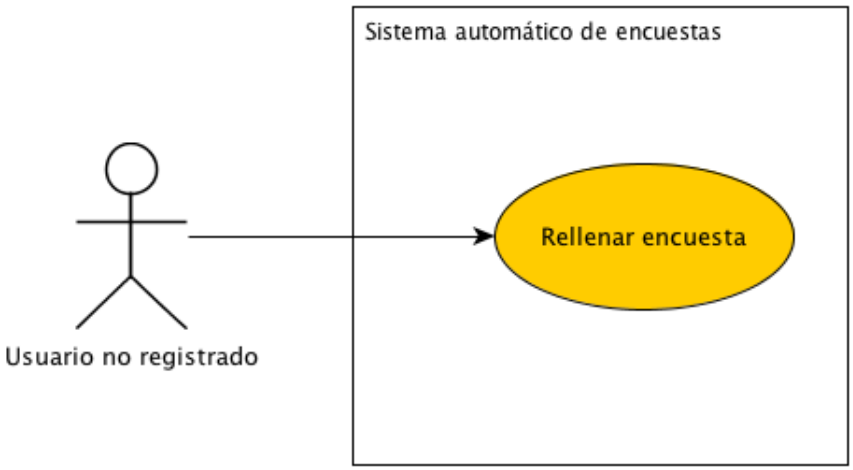


Figura 13: Estructura MVC de la aplicación.

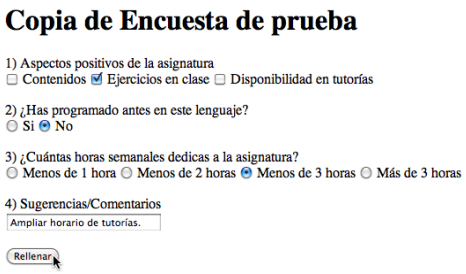


Figura 14: Formulario de una encuesta con las preguntas cumplimentadas.

Como puede verse en la figura 14 , el visitante completa los datos de la encuesta y pulsa sobre el botón “Rellenar” para guardar sus respuestas en el sistema para información del creador de la encuesta.

4.3. Diagrama MVC

Como se ha explicado en apartados anteriores, la piedra angular del desarrollo de aplicaciones en Rails es el uso del paradigma MVC. Trabajar respetando esta máxima nos brindará muchas ventajas; una de las más interesantes, es que el código

resultante será muy modular, lo cual nos será de gran ayuda a la hora de realizar prototipos e implementar nuevas funcionalidades.

En la figura 15 puede verse un esquema de cómo se ha estructurado el presente proyecto, siguiendo las pautas de MVC. En todo sistema implementado en Rails, existe un controlador raíz *Application*, que es el punto de partida de la aplicación: entre algunas de sus funciones, tiene que determinar qué controlador se encargará de la página inicial de la web y qué vista de las disponibles para ese controlador será la primera en mostrarse; en nuestro caso, tendremos un controlador *Main*. Es importante notar que se representan las vistas como algo abstracto, ligado a los controladores, por simplicidad; realmente, cada una de esas vistas estará compuesta por varios ficheros HTML, entre hojas de estilo, vistas parciales, etc.. que se interpretarán en el navegador para formar cada vista final.

Este controlador servirá como nexo hacia las diversas funciones de la aplicación, invocando a uno u otro bloque dependiendo de las acciones del usuario. Al ser la puerta de entrada, es necesario que controle el acceso de los usuarios; para ello, delega funciones en el controlador *Users*, que gestiona el acceso a la base de datos de usuarios. Básicamente, funcionará como una llamada a función de un lenguaje de programación: el usuario, en la vista *Main*, introduce sus datos de acceso, que sirven como parámetros de entrada para que el controlador de usuarios pueda verificar si los datos son correctos, devolviéndolo como resultado al controlador que lo ha invocado.

Llegados hasta aquí, es necesario realizar algunas aclaraciones sobre el esquema: todos los controladores llevan asociada una vista. Para los controladores de modelos, Rails crea por defecto unas vistas que permiten gestionar el contenido del modelo con una interfaz muy sencilla. Sin embargo, para este proyecto no se tienen en consideración; cuando se requiera que el usuario realice modificaciones de datos, se le proporcionará una vista creada a medida.

También con la idea de simplificar, en el diagrama se han agrupado todos los controladores que, conjuntamente, controlan toda la estructura de datos de las encuestas. Esto incluye varias clases diferentes: la encuesta como contenedor principal, las preguntas que existen en la base de datos, y las respuestas que se han facilitado a las preguntas de las encuestas. En la sección dedicada a ello, se explicará con mayor detenimiento la estructura elegida para la base de datos.

Por último, existen dos clases de modelos en la aplicación: aquellos que no se basan en almacenamiento persistente, y los que vuelcan la información en base

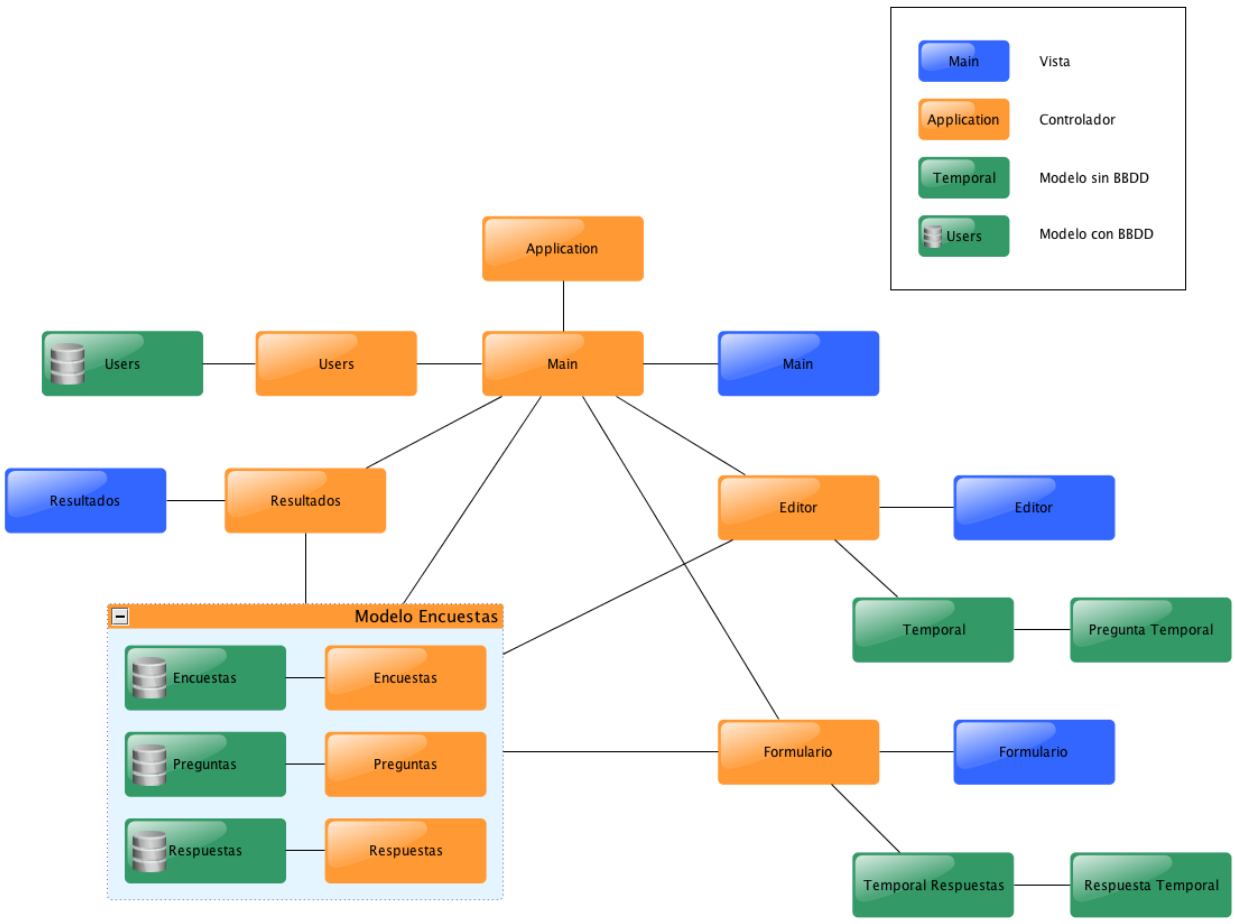


Figura 15: Estructura MVC de la aplicación.

de datos. Los primeros serán almacenes temporales de información, cuando la aplicación se encuentra en un estado en que el usuario está editando la información; por ejemplo, cuando se está creando una encuesta nueva, el usuario puede eliminar y agregar tantas preguntas como desee hasta obtener un conjunto que le satisfaga. En nuestro sistema, se almacenará la encuesta final una vez que el usuario pulse la opción correspondiente. Por supuesto, esta información será editable más adelante; se cargará de nuevo en un modelo temporal hasta que se guarden los cambios.

4.4. Diseño de BBDD

En este apartado trataremos el diseño de nuestra base de datos relacional para almacenar las encuestas y los resultados que vayan generando los usuarios. En la figura 16 podemos ver de forma esquemática los diferentes tipos de entidades que necesitamos almacenar para garantizar el correcto funcionamiento de la plataforma.

Como puede apreciarse, todas las entidades cuentan con un atributo en común *id*. Este atributo es generado por defecto al crear una nueva tabla en la base de datos en Rails, y hace las veces de clave primaria. Existe la posibilidad de modificar esta clave primaria por una de nuestra elección, pero para este caso no lo hemos visto necesario. Del mismo modo, Rails también genera dos atributos (*created_at* y *updated_at*) que indican la fecha de creación y última modificación de una instancia, respectivamente; ambos se han omitido por simplificar el esquema.

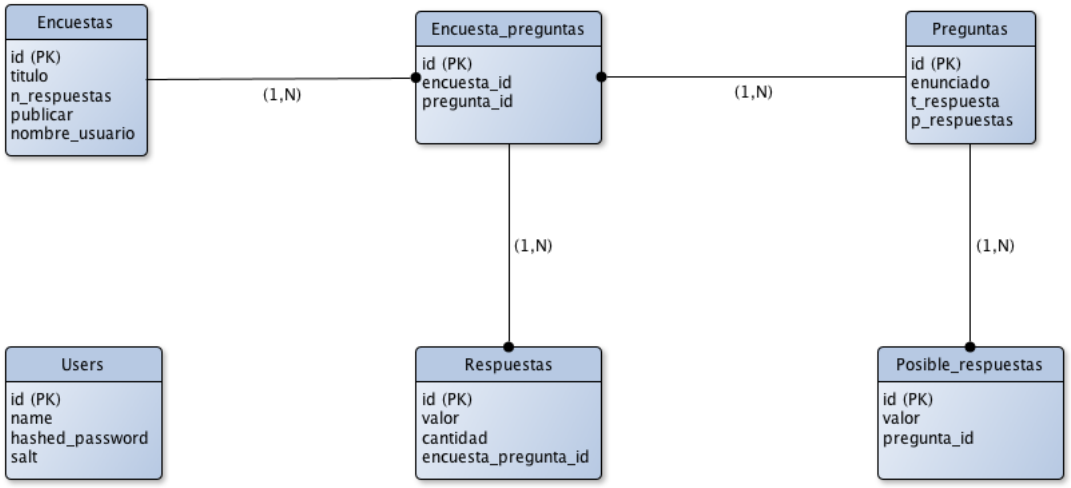


Figura 16: Diagrama entidad-relación de la base de datos.

En el anexo C, se detalla cada una de estas entidades y sus atributos.

4.5. Edición de encuestas

En este apartado estudiaremos en detalle una de las partes más importantes para el funcionamiento de este proyecto. Al ser todo el contenido generado por los usuarios, necesitamos proveer una herramienta que cumpla un doble objetivo: debe ser sencilla de utilizar, a ser posible empleando animaciones y efectos, ofreciendo una sensación dinámica (requisito 5.2.); y no por ello debemos renunciar a toda la versatilidad de una aplicación de estas características, con una personalización completa de las preguntas y sus respuestas, así como el orden de aparición de ambos elementos (requisito 1.1.). Para lograr este objetivo, debemos distinguir las tres partes principales a la hora de implementar nuestra solución siguiendo el concepto de MVC, como puede apreciarse en la figura 17:

- *Interfaz de usuario*: este papel lo interpretará la vista del controlador *Editor*, y debe cumplir los requisitos mencionados en el párrafo anterior. Como puede verse en el diagrama, contaremos con varios ficheros HTML formando la vista final. El archivo *index.html.erb* albergará a cada una de las partes mediante las directivas *render: partial* [3]. Esto de cara al usuario resulta transparente: el servidor enviará al cliente el código de una página completa en HTML. Para nosotros, supondrá poder dividir al máximo los diferentes contenidos de la vista, facilitando la tarea de modificar y depurar el código. En el código de ejemplo de la figura 18, se cargará el fichero *_preguntas_db.html.erb*, de modo que su código quedará dentro del div con id *preguntas_db"*.

  Además, puede observarse un archivo adicional en formato *js.rjs* formando parte de este conjunto; se trata de un fichero de javascript, dedicado a ejecutar unas animaciones concretas que no pueden llevarse a cabo únicamente con Ruby en base al *framework*. Se trata de un fichero muy sencillo, de apenas dos líneas de código, como se verá más adelante.

- *Controlador*: a pesar de ser únicamente un fichero, es de vital importancia para este módulo; concentrará todos los métodos que el usuario irá invocando a través de la vista, seleccionando los elementos que en ella aparecen e interactuando con ellos. Tal como se desgranó en el apartado dedicado al paradigma MVC, el controlador es el nexo de unión de todos los elementos de este sistema; es encargado, por tanto, de gestionar también los modelos de datos, tanto los temporales como los persistentes, de modo que todo funcione correctamente.

- *Modelos de datos*: como puede apreciarse en el esquema, el modelo de datos de este módulo de nuestra aplicación estará formado por dos grupos de

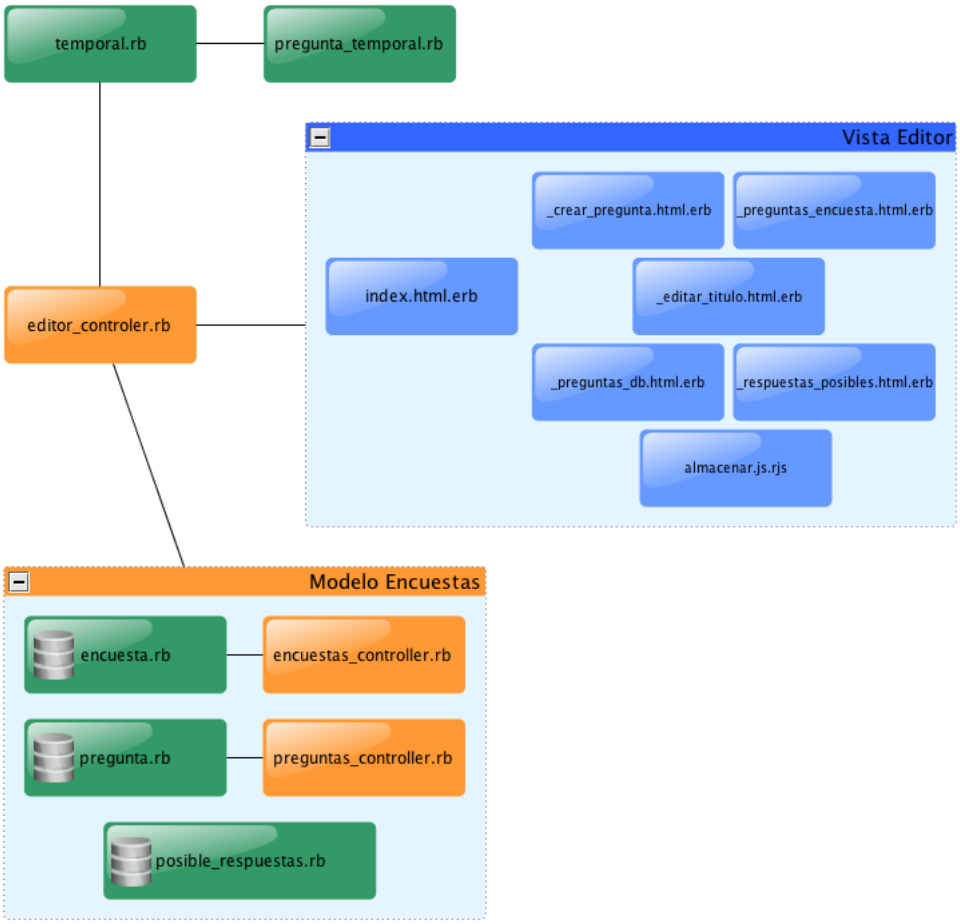


Figura 17: Diagrama MVC para el módulo de edición de encuestas.

archivos con distinta finalidad: en primer lugar, contamos con los ficheros pertenecientes al modelo temporal, encargados de gestionar la información de la encuesta que se está editando en tiempo real, en estado cambiante; tendremos una clase *temporal*, que simulará una encuesta que contiene en su interior *preguntas temporales*, cada una con sus posibles respuestas. Por otra parte, será necesario comunicar a la base de datos cual será la información definitiva que se quiere almacenar una vez el usuario cambie a una vista diferente. Para ello, estas clases cuentan con varios métodos destinados a volcar la información en la base de datos como nuevas instancias de las diferentes entidades.

Con la idea de organizar la explicación de las funcionalidades de esta parte

```
<h1>Preguntas almacenadas</h1>

<div id="preguntas_db">
    <%= render :partial=>"preguntas_db" %>
</div>
```

Figura 18: Detalle de la función *render* para cargar vistas parciales.

del sistema, nos centraremos en el aspecto de la vista, e iremos detallando los diferentes elementos que en ella aparecen y los segmentos de código que controlan su interacción con el usuario. Seguiremos la numeración indicada en la figura 19.

4.5.1. Nueva encuesta: selección del punto de partida

Una de las primeras partes que debemos valorar, a pesar de que pueda parecer algo trivial, es conocer qué necesita hacer el usuario exactamente cuando accede al editor de encuestas. Para ello, contemplamos tres escenarios posibles. Todos ellos comparten algunos aspectos comunes, especialmente la inicialización del modelo temporal, aunque variando el nombre por defecto de la encuesta. El comportamiento de la aplicación será idéntico para todos ellos, con ligeras excepciones a la hora de guardar el trabajo realizado.

En primer lugar, crear una nueva encuesta sería el caso básico: el usuario podrá pulsar el botón *crear* desde la página de inicio, cargando el editor de encuestas con ninguna pregunta en el modelo temporal. Podrá comenzar a insertar preguntas normalmente, como veremos más adelante. Por defecto, esta nueva encuesta no tendrá ningún nombre, por lo que se insertará *Introducir título* a modo informativo. Sería el caso que puede apreciarse en la figura 19. Huelga decir que el campo *preguntas almacenadas* podría contener múltiples preguntas que se hayan insertado al crear otras encuestas.

El usuario también podría necesitar editar una encuesta creada anteriormente para terminar de construirla a su gusto. Es imprescindible destacar que esta opción únicamente estará disponible para aquellas encuestas que aún no se han publicado y, por tanto, no puedan recibir respuestas. Una vez que se decide cambiar su estado

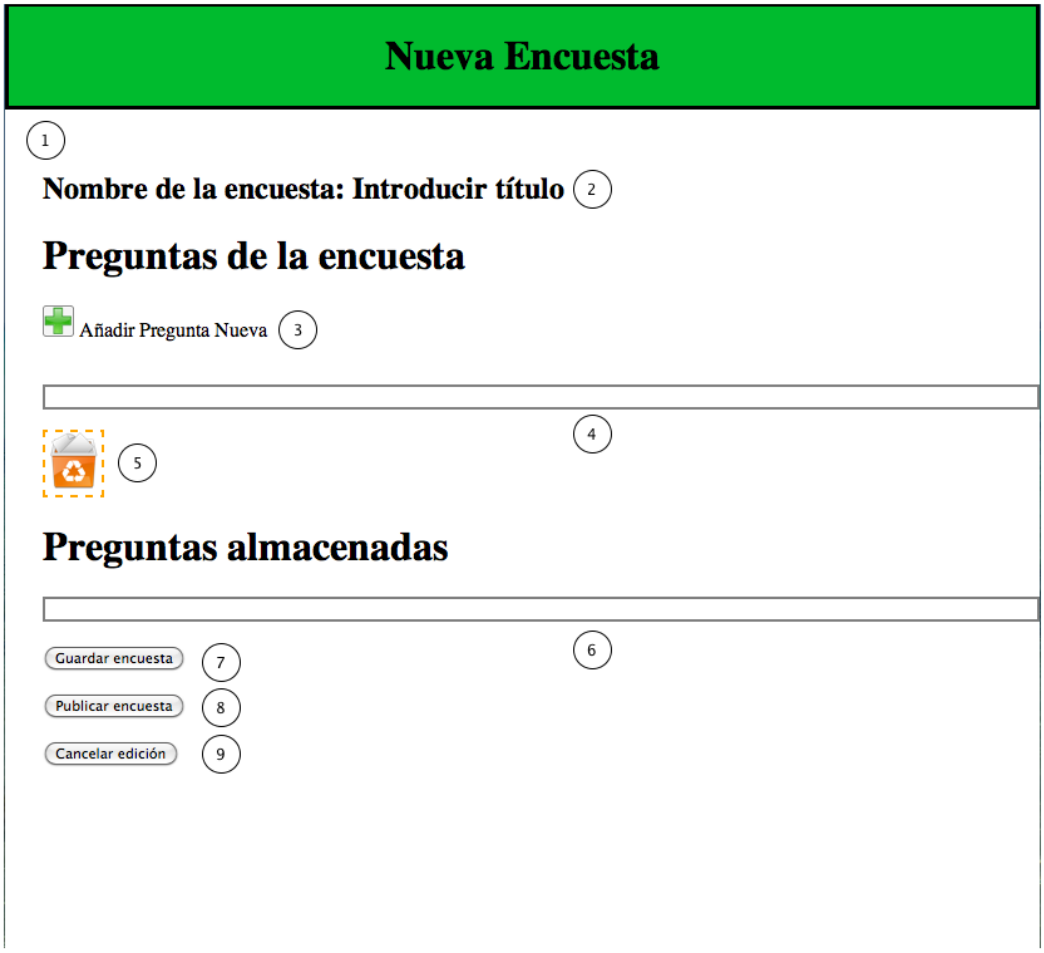


Figura 19: Página del editor de encuestas.

a publicada, no podrá volver a editarse. Esto se ha decidido así por mantener la coherencia en los datos recogidos, ya que un usuario no debería contestar una versión con más o menos preguntas que el resto.

Una vez hemos decidido pulsar el botón *editar* de la encuesta, se almacenará en el modelo temporal el valor *true* en su atributo *editando*, de modo que el resto del sistema sea consciente de nuestra acción. Desde ese momento podremos realizar todas las modificaciones que creamos necesarias, desde cambiar el título de la misma, hasta modificar el orden de las preguntas y su cantidad en el cuestionario. Podremos almacenar los datos en cualquier momento, o deshacer los cambios para dejarlo en su estado anterior; todos estos procesos los veremos en siguientes

secciones.

Por último, el usuario podrá crear una encuesta partiendo de una anterior a modo de plantilla. De este modo, se cargarán todas las preguntas que pudiera tener aquella en el modelo temporal, con lo que partiremos de esa base. Por defecto, el nombre de la encuesta será el mismo que el de la encuesta original, con la cadena de texto *Copia de* concatenada al comienzo. Esto se realiza para recordar que no estamos editando la encuesta, sino creando una nueva; por tanto, no se permitirá almacenarla en base de datos si su título coincide con cualquiera de las ya presentes en el sistema.

4.5.2. Editar el título

Para que el usuario pueda modificar el nombre de la encuesta, se ha decidido implementar una etiqueta de texto editable. El funcionamiento es muy sencillo: cuando no se produzca ninguna interacción, en pantalla se mostrará el título actual como un texto cualquiera; si el usuario ubica el puntero del ratón sobre este título, tendrá lugar un efecto de subrayado amarillo, indicando que puede seleccionarse el elemento. También aparecerá un texto indicativo. Puede verse una muestra de este efecto en la figura 20.

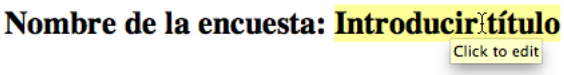


Figura 20: Detalle del cuadro de título, al ubicar el puntero encima.

Una vez el usuario ha seleccionado el campo, este se transformará en una caja de texto, donde se podrá introducir el nuevo título en caso de querer modificarse, tal y como puede verse en la figura 21. Se podrán aceptar o rechazar los cambios mediante los dos botones correspondientes.

Figura 21: Detalle del cuadro de título, al pulsarlo.

Todo este comportamiento se implementa de forma automática al utilizar el método *in_place_editor_field*, que sustituye a los clásicos *<textfield>* de HTML. Con indicarle qué atributo de qué objeto queremos modificar con este campo, el propio *framework* nos generará todo lo descrito en este apartado.

4.5.3. Crear y añadir una pregunta

Dentro del grupo de elementos relacionados con las preguntas de la encuesta (es decir, las preguntas que ya tenemos seleccionadas y, por tanto, almacenadas en el modelo temporal), el primero de ellos es el editor de preguntas. Este bloque del módulo permitirá crear preguntas nuevas, por si el usuario necesita alguna que no encuentre en la base de datos. Tal como puede verse en la figura 19, un botón indicará la existencia del editor. Al pulsarlo, se desplegará con efecto persiana un elemento nuevo como el de la figura 22.

Figura 22: Formulario desplegable para añadir nuevas preguntas.

Como puede observarse, dentro de este desplegable contamos con diversos campos y botones para realizar todas las tareas relacionadas con la creación de nuevas preguntas: en primer lugar, un campo de texto para editar el enunciado de nuestra pregunta; justo a continuación, un menú donde podremos determinar qué clases de respuesta queremos ofrecer a nuestro usuario, entre las ya conocidas *Selección múltiple*, *Selección única* y *Texto libre*, junto al botón *Crear* que añadirá la nueva pregunta a nuestra encuesta temporal; y, finalmente, podremos determinar los valores de las respuestas que queremos ofrecer como opciones al usuario. Contamos con dos botones indicados con *+1* y *-1*, que se utilizan para aumentar o reducir el número de alternativas que queremos mostrar, respectivamente. La aplicación está preparada para eliminar esta última sección si el usuario elige *Texto libre* como

tipo de respuestas posibles, ya que en ese caso el usuario podrá escribir en un cuadro de texto.

Una vez hayamos dado valores a todos los campos, una pulsación del botón *Crear* cerrará automáticamente el desplegable, y podremos ver nuestra nueva pregunta incluida dentro del bloque de las preguntas temporales. Esto también implica que los elementos generados no se podrán editar.

4.5.4. Listado de preguntas de la encuesta temporal

Este es uno de los elementos más importantes dentro de nuestro editor de preguntas. Puede verse qué aspecto presenta en la figura 23. En él se listarán todas las que tenemos en todo momento presentes en la encuesta; se trata, por tanto, del conjunto que se almacenaría en base de datos en caso de guardar los cambios. Cuando tengamos alguna pregunta en este área, se mostrará su enunciado, el tipo de respuestas que permite, así como el número de posibles respuestas que tiene almacenada.

Para añadir una pregunta a este área, existen dos posibilidades: por un lado, tenemos la descrita en el apartado anterior, creando una nueva pregunta; o bien podemos utilizar la función de soltar y arrastrar elementos. Esta última alternativa nos permite trasladar preguntas que tuviésemos en la base de datos (el área marcada como número 6 en la figura 19) hasta esta sección, simplemente arrastrando con el ratón por la pantalla la pregunta que nos interese.

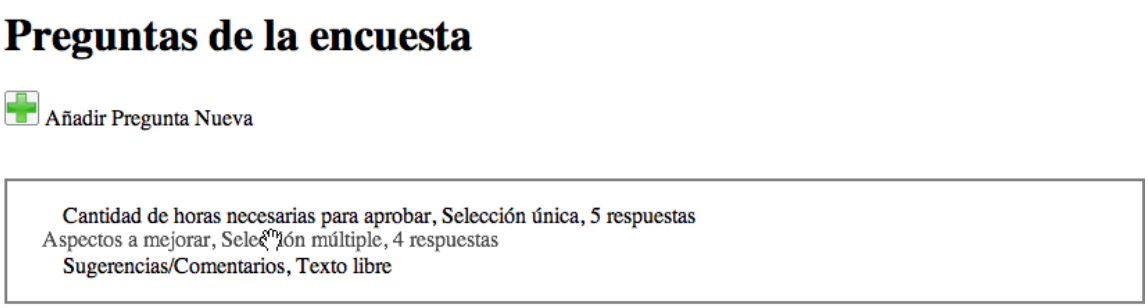


Figura 23: Detalle del área de preguntas de la encuesta, con un elemento siendo arrastrado.

Este comportamiento se logra utilizando los métodos *draggable_element* y su *drop_receiving_element* asociado: el primero indica qué elementos de la vista tie-

nen la propiedad de poder ser arrastrados por la pantalla. En este caso, se definen las preguntas de la zona temporal utilizando una variable *domid* en el código de la vista. El segundo elemento indica qué zonas de la vista permiten que se suelte en su interior un *draggable_ element*, desencadenando una acción en su controlador. Si las preguntas se sueltan en una zona que no tiene definido este método, la pregunta vuelve a la posición inicial donde se comenzó el desplazamiento.

Como se puede observar en los códigos adjuntos (figuras 24 y 25), ninguna de ambas funciones la encontraremos implementada dentro del fichero *_preguntas_ encuesta.html.erb*, ni siquiera para definir el área donde dejar los elementos seleccionados, como invita a pensar la lógica. Esto es debido al uso de vistas parciales, de modo que en el fichero *index.html.erb* se define un * * [5] dentro del cual se carga la vista parcial. Esta función señala dicho bloque como objetivo, y por ello no puede definirse en el archivo de vista parcial.

La potencia de este par de métodos radica en que una zona *drop_ receiving_ element* puede distinguir qué elemento es el que se está soltando, lo cual nos será de vital importancia para implementar el borrado de preguntas temporales.

```
<ul id="lista_preguntas_db">
  <% for @elem in @preguntas%>
    <% domid = "pregunta_#{@elem.id}" %>
    <li class="pregunta_db" id='<%= domid %>'>
      <%= @elem.enunciado%>,
      <%= @elem.t_respuesta %><% if @elem.t_respuesta != "Texto libre"%>,
        <%= @elem.p_respuestas %> respuestas
      <%end%>
    </li>
    <%= draggable_element(domid, :ghosting=>true, :revert=>true) %>
  <% end %>
</ul>
```

Figura 24: Código del fichero *_preguntas_ db.html.erb*, donde se aprecia el elemento *draggable_ element*.

```
<%= drop_receiving_element("preguntas_encuesta",
                           :accept => "pregunta_db",
                           :hoverclass => 'hover',
                           :with => "'pregunta=' + element.id.split('_').last()",
                           :url => {:action => :anhadir_pregunta})
    %>
```

Figura 25: Detalle de la función *drop_ receiving_ element*, en el fichero *index.html.erb*.

En esta versión de la aplicación, no se permite que existan dos preguntas cuyo enunciado sea idéntico; en caso de querer hacerlo, simplemente se ignorará la adición, quedando en la encuesta temporal una única pregunta con el enunciado concreto. Esto también incluye preguntas que cuentan con distintos tipos de respuestas y posibles respuestas; únicamente prevalecerá la primera de todas ellas que fue añadida.

Finalmente, esta zona también permite la reordenación de sus elementos, de modo que alteraremos el orden en el cual las preguntas se presentan en el formulario una vez esté en producción. De un modo similar al anterior, utilizaremos la función *sortable_ element*: únicamente indicando qué etiqueta contiene la lista de elementos y qué acción queremos realizar para la ordenación conseguiremos el efecto deseado. Es, por tanto, necesario definir las preguntas en este área como *<li>* dentro de un *<ul>* [5], que será el objetivo de este método.

4.5.5. Borrado de preguntas temporales

Basándonos en la filosofía de uso de la aplicación mediante *drag and drop*, esta zona de la vista está dedicada a permitir el borrado de preguntas de nuestra encuesta temporal. Utilizaremos los métodos descritos en el apartado anterior, de modo que podamos especificar que únicamente las preguntas que provengan de la zona de preguntas temporales puedan ser eliminadas; al intentar soltar una pregunta de la base de datos en esta zona de borrado, simplemente se comportará como si hubiésemos dejado la pregunta en un área no válida. Con la intención de dar mayor información al usuario sobre sus acciones, la zona cambiará de color una vez ubiquemos encima una pregunta que puede ser eliminada al soltar. Puede verse el comportamiento de este evento en la figura 26

Figura 26: Detalle del área de borrado con un elemento del tipo correcto encima.

4.5.6. Listado de preguntas de la base de datos

En este área, se mostrarán todas las preguntas que se hayan añadido a la base de datos, de modo que puedan reutilizarse en la encuesta sobre la que estamos trabajando. A efectos de la información almacenada, simplemente aparecerá una nueva fila en la tabla *Encuesta_preguntas*, indicando la nueva relación. Conviene recordar que, al reutilizar una pregunta, también obtendremos todas las respuestas posibles que tuviese asociadas. La figura 27 muestra la apariencia de esta parte de la aplicación.

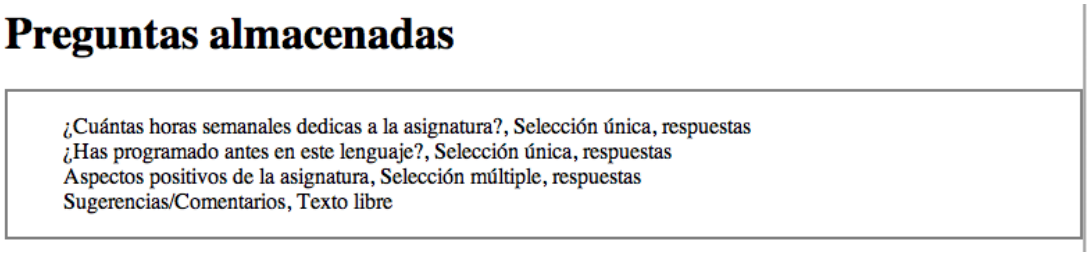


Figura 27: Detalle del área de preguntas almacenadas en base de datos, con algunos elementos.

Para que una pregunta aparezca en esta zona, al menos un usuario debió guardar cambios de una encuesta que tuviese alguna pregunta. A partir de ese momento, todas esas cuestiones podrán ser reutilizadas por cualquier usuario para sus propios cuestionarios. Por tanto, las preguntas de la zona temporal sólo se guardarán si el usuario así lo declara pulsando el botón correspondiente. Si se cancelase la edición, todas esas preguntas serían eliminadas irreversiblemente. No es necesario que la encuesta llegue a publicarse para almacenar las preguntas; únicamente es obligatorio guardar los cambios.

Como se ha comentado en los apartados anteriores, las preguntas que se encuentren en esta zona son potencialmente objeto de ser arrastradas para añadirlas a nuestra plantilla de encuesta temporal, pero no pueden eliminarse mediante la zona destinada a las preguntas temporales, al detectarse el origen desde el cual estamos desplazando un elemento.

4.5.7. Guardar la encuesta

Cuando el usuario decida mantener las preguntas que tenga en un momento determinado en la lista de su encuesta temporal, podrá utilizar el botón correspondiente. Al pulsarlo, se invocará a la función *guardar*, que simplemente redirigirá la llamada hacia *save_ encuesta*, ambos del controlador *Editor*; esta última función es quien ejecutará todas las acciones necesarias para llevar el contenido a la base de datos. El motivo por el que es necesario realizar dos llamadas es que el método *save_ encuesta* se utiliza también cuando se quiere publicar una encuesta, tal y como se explicará en su apartado.

Al ser el encargado de coordinar el volcado en memoria de toda la información del modelo temporal, también tiene algunas responsabilidades para garantizar que los datos son correctos. Así, al comienzo del método, se comprueba si el título de la encuesta no está repetido en la base de datos, excepto si estamos precisamente editando dicha encuesta. Para ello se comprueba el valor del atributo *editando*, que tomó valor al pulsar sobre el botón de editar la encuesta; si estamos tratando de guardar una encuesta con el nombre de otra sin estar editándola, el sistema cancelará nuestra acción, modificará el nombre de la encuesta añadiendo por defecto la cadena "Copia de", y mostrará un mensaje de error. La figura 28 muestra el código que implementa esta comprobación. Del mismo modo, al final del método se controla la excepción de encontrar la encuesta sin título, comprobando que el resultado de la operación de guardar la encuesta es correcto.

```
if ((not @encuesta == nil) and ((not @temporal.editando)) and (@temporal.id_encuesta != @encuesta))
  flash[:notice] = "Ya existe una encuesta con el mismo nombre"
  @temporal.actualizar_titulo("Copia de " + @temporal.titulo)
  redirect_to :action => :index
```

Figura 28: Fragmento del código de la función *save_ encuesta* del controlador *editor_ controller.rb*, donde se comprueba si existe duplicidad de la encuesta.

Básicamente, en nuestro sistema sólo tendremos dos tipos de encuestas que debemos almacenar: las encuestas que se están editando sobre una ya almacenada, y las encuestas que, o bien son nuevas, o bien toman como punto de partida una encuesta anterior (no se están editando, sino creando). De esta manera, la mayor parte del código a implementar será común a ambos escenarios. Aún así, existen algunas diferencias: la primera de ellas es la descrita en el párrafo anterior, con la comprobación del título.

La otra diferencia radica en el hecho de que, en una encuesta que estemos

editando, ya encontramos en la base de datos algunas preguntas relacionadas. A priori, no existe ninguna manera de conocer en qué estado quedará la encuesta tras la acción del usuario: podría suceder que modifique sólo algunas preguntas, que elimine todas las que tenía originalmente, o simplemente podría dejarla inalterada (aunque sería más correcto por su parte utilizar la función de cancelar la edición, es perfectamente válido guardar la encuesta sin realizar modificaciones). Nos encontramos, por tanto, una situación bastante compleja de resolver. Por, ello, se ha tomado la decisión de eliminar todas las entidades *encuesta_pregunta* que tuviese anteriormente la encuesta en caso de que estemos editándola. Esto se realiza empleando el método *delete*, que ejecuta una consulta a la base de datos para eliminar registros. Es decir, en caso de que estemos modificando una encuesta, procuraremos eliminar todas las referencias a preguntas que tuviera anteriormente almacenadas, de modo que podamos trabajar como si se tratase de una encuesta completamente nueva. Con esta práctica, conseguimos reducir notablemente el código de esta función, trabajando de una única forma para todas las encuestas.

Como ya hemos comentado, en este apartado del proyecto tendremos una sección de código que se encarga de coordinar la actuación de los demás elementos, para almacenar correctamente los datos. Así, el controlador *Editor* pasará el control a varios modelos de datos, invocando en primera instancia al modelo *Encuesta* a través de su método *anhade_pregunta_encuesta_desde_temporal*, cuya finalidad es crear las nuevas relaciones *Encuesta_pregunta* entre nuestra encuesta y sus preguntas. En la figura 29 puede verse su código.

```
def anhade_pregunta_encuesta_desde_temporal(temp)
  temp.preguntas.each do |elem|
    preg = elem.genera_pregunta_db
    pe = EncuestaPregunta.new()
    pe.pregunta = preg
    encuesta_preguntas << pe
  end
end
```

Figura 29: Detalle de la función *anhade_pregunta_encuesta_desde_temporal*, en el fichero *encuesta.rb*.

Esta función recibe como parámetro el modelo temporal a partir del cual estamos generando la nueva encuesta. Para cada una de las preguntas que tenga almacenadas, se genera su homónimo en base de datos mediante el método *genera_pregunta_db*, perteneciente a la clase *pregunta_temporal*; se está aplicando a cada elemento (*elem*) obtenido de la iteración en el atributo *preguntas*, que es un array de preguntas temporales. Dicho método internamente creará un objeto de

la entidad *Pregunta*, así como sus posibles respuestas asociadas en caso de existir (recordemos, para las preguntas que no son de tipo *Texto libre*).

```
def genera_pregunta_db
  existe = false
  preguntas = Pregunta.encuentra_preguntas
  for pregunta in preguntas
    if pregunta.enunciado == enunciado
      existe = true
      break
    end
  end
  if not existe
    pregunta = Pregunta.new()
    pregunta.enunciado = enunciado
    pregunta.t_respuesta = t_respuestas.to_s
    pregunta.save()
    if t_respuestas.to_s != "Texto libre"
      for elem in respuestas
        respuesta = PosibleRespuesta.new()
        respuesta.valor = elem
        respuesta.pregunta_id = pregunta.id
        respuesta.save()
      end
    end
  end
  pregunta
end
```

Figura 30: Detalle de la función *genera_pregunta_db*, en el fichero *pregunta_temporal.rb*.

Como puede apreciarse en la figura 30, el primer bloque se encarga de controlar que la pregunta que queremos añadir no existe previamente en la base de datos, de acuerdo a su enunciado, tal y como ya habíamos explicado anteriormente. En caso de no encontrar ningún elemento previo, procedemos a crear uno nuevo: invocamos a *Pregunta.new()*. Dicho método es la consulta SQL *CREATE* que, a través del framework Rails, se comporta como un constructor en una aplicación orientada a objetos; es una demostración de las facilidades que ofrece Rails, permitiendo programar en un único lenguaje. Procederemos del mismo modo para almacenar todas las posibles respuestas asociadas a la pregunta.

Una vez creada la nueva pregunta en caso necesario, se devuelve el control a la función inicial: crearemos una nueva relación encuesta-pregunta. La función anterior nos devolvió como valor de retorno el identificador de la pregunta recién creada o encontrada, y podremos utilizarlo para inicializar el valor correspondiente en la nueva relación.

Finalmente, tras almacenar la encuesta en la base de datos, el método inicial *guardar* se encarga de redirigir al usuario a la página inicial de la aplicación, donde el controlador *main* retoma la ejecución.

4.5.8. Publicar la encuesta

Finalmente, cuando el usuario considere que su encuesta se encuentra en un estado final, podrá ponerla a disposición de su grupo muestral para que empiece a generar resultados. A efectos prácticos, este cambio de estado simplemente corresponde con un atributo del modelo, cuyo valor tendrá que ser "verdadero" para que la aplicación pueda ponerla en producción.

Un detalle a tener en cuenta es que el usuario puede querer publicar su encuesta una vez ha concluido de crearla, o bien a posteriori, tras algunos retoques. Es decir, existe la posibilidad de publicar una encuesta tanto si está en la base de datos como si aún no se ha guardado, simplemente porque aún está en el modelo temporal. Para poder manejar estas situaciones, se realiza una sencilla comprobación: si no existe la encuesta, se llama al método *save_encuesta* descrito en apartados anteriores. Dicha llamada proporciona como valor de retorno el identificador de la encuesta recién creada; de este modo podremos modificar su atributo *publicada* para hacerla visible. Si, en cambio, la encuesta ya existía, ya la conoceremos, y por tanto podremos publicarla sin complicaciones.

La necesidad de emprender estas acciones de esta manera reside en las redirecciones: al concluir un método, la última línea puede ser un enlace a una URL, de modo que se pasa el control de la aplicación a otro módulo, pero no puede haber más de una línea de esta clase a la vez en la misma secuencia de ejecución. En nuestro caso, al tener el método *publicar* una ramificación (cuando necesita guardar la encuesta y cuando no hace falta) que únicamente puede desembocar en una redirección al final de la función, el método *save_encuesta* no puede implementar dicho enlace, ya que entraría en un claro conflicto. Era por tanto necesario utilizar el método *guardar* para realizar la redirección en caso de que se guardase una encuesta sin querer publicarla.

4.5.9. Cancelar la edición

En cualquier momento durante la edición de una encuesta, el usuario puede decidir que prefiere conservar el estado anterior. Dada la implementación de nuestra plataforma, esta acción es extremadamente sencilla: tras pedir confirmación, borraremos de la sesión el modelo temporal sobre el que estaba trabajando, y finalmente mostraremos un mensaje informativo indicando la cancelación.

4.6. Rellenado de encuestas y obtención de resultados

Hasta ahora, en esta sección de diseño de la aplicación nos hemos centrado en las partes del sistema que procesan la actividad de los usuarios registrados, todo lo relacionado con la creación y modificación de encuestas. Sin embargo, no debemos olvidar que el objetivo final perseguido por todos estos módulos es obtener información de una población, que tiene que resolver el cuestionario que les facilitemos. Para poder satisfacer esta necesidad, contamos con algunas partes específicas en nuestra aplicación, tal como puede verse en la figura 31.

Dividiremos la explicación entre los controladores *formulario_controller.rb*, encargado de mostrar la encuesta para poder resolverla, y *resultados_controller.rb*, donde se muestran los datos recogidos en una encuesta. En esta parte de la aplicación es cuando los usuarios no registrados pueden interactuar con el contenido. Con la idea de para distinguirlos fácilmente de los usuarios registrados en las explicaciones, a éstos los llamaremos usuarios registrados, o simplemente creadores, mientras que a aquellos nos referiremos como usuarios o encuestados.

4.6.1. Formulario de las encuestas

Recordando el apartado anterior, cuando un usuario registrado desea poner en producción su encuesta, debe modificar su estado a "publicado", de modo que los encuestados puedan responder a las preguntas. A partir de ese momento, cualquier usuario podrá acceder a las encuestas que hayan puesto a su disposición algún creador. Para acceder a una encuesta con el fin de rellenarla, existen dos alternativas: los encuestados pueden acceder a la aplicación y seleccionar la encuesta que desean responder; por otra parte, cuando se publica una encuesta, el sistema pone a disposición del usuario creador una dirección URL que podrá distribuir libremente, y

Figura 31: Diagrama MVC para los módulos de recogida y muestra de datos.

que dará acceso directo a la encuesta. Ambos modos de acceso son completamente idénticos, no existiendo ninguna diferencia funcional entre ambos.

Generalmente, los formularios web se basan en un único modelo; es decir, la vista nos presenta tantos campos como atributos tiene una entidad de su base de datos, y asigna los valores que hemos introducido en una nueva instancia. En nuestro caso, la cosa es más complicada: la información se encuentra contenida en múltiples objetos, desde las preguntas hasta sus posibles respuestas, y necesitamos almacenar las respuestas a dichas preguntas bajo el mismo formato, independientemente de dónde provenga. Tenemos preguntas de selección única, selección múltiple, y respuesta libre, donde cada una solicitará información de una forma diferente. Esto se conoce como un formulario múltiple [4].

En la figura 32 podemos ver el código íntegro de la vista donde se mostrará el formulario de la encuesta. Declaramos que en esta página ubicaremos un formulario

```
<h1><%= @encuesta.titulo %></h1>
<p>
<% i = 1 %>
 <% form_tag :action => :rellenar do %>
   <p>
    <% for respuesta in @temporal_respuestas.respuestas %>
      <p>
        <%= i %>)
        <% encuesta_pregunta = respuesta.pregunta %>
        <% pregunta = encuesta_pregunta.pregunta %>
        <% i = i + 1 %>
        <%= pregunta.enunciado %>
        <% fields_for 'respuestas[][]', respuesta do |f| %>
          <% if pregunta.t_respuesta.to_s == "Selección única" %>
            <% for resp in pregunta.posible_respuestas %>
              <%= f.radio_button :valor, resp.valor %> <%= resp.valor %>
              <%= f.hidden_field :pregunta, :value => encuesta_pregunta.id %>
            <% end %>
          <% elsif pregunta.t_respuesta.to_s == "Selección múltiple" %>
            <% for resp in pregunta.posible_respuestas %>
              <%= check_box_tag 'valores[]', "#{resp.valor}/#{encuesta_pregunta.id}" %> <%= resp.valor %>
            <% end %>
          <% elsif pregunta.t_respuesta.to_s == "Texto libre" %>
            <%= f.text_field :valor %>
            <%= f.hidden_field :pregunta, :value => encuesta_pregunta.id %>
          <% end %>
        <% end %>
      </p>
    <% end %>
   </p>
  <%= submit_tag 'Rellenar' %>
 <% end %>
</p>
```

Figura 32: Contenido del fichero *index.html.erb*, la vista de este módulo.

mediante la etiqueta *form_ tag*; todo el contenido relacionado con formularios que se introduzca dentro de esta cláusula (podemos ver un cierre *end* al final del código) se tratará al pulsar el botón indicado por *submit_ tag*, justo antes del mencionado cierre. De esta manera conseguimos que se envíen los datos al servidor mediante un mensaje POST de HTTP, cuyo contenido podremos definir nosotros libremente para poder tratarlo como mejor nos parezca.

En este punto es necesario hacer un pequeño inciso; la sexta línea de código muestra un bloque que itera los elementos *respuesta* que contiene un atributo del modelo temporal, que por lógica podemos deducir son elementos de la clase *respuesta_ temporal*. En el método del controlador que invoca a esta vista, se inicializa el atributo *respuestas* del modelo temporal, que es un array. Por cada pregunta que tiene la encuesta, se crea una nueva instancia de *respuesta_ temporal*. Cada una de ellas posee dos atributos, en el primero de los cuales introducimos la referencia al objeto *encuesta_ pregunta* de la pregunta que estamos tratando, ya que es donde se almacenará la referencia a la respuesta(ver sección 4.4: Diseño de BBDD). El segundo atributo representa el valor que se asignará a la nueva respuesta, que por el momento quedará como valor nulo. De este modo, estamos cumpliendo un do-

ble objetivo: iterar para mostrar por pantalla todas las preguntas de la encuesta, y obtener una referencia mediante la cual introducir algunos de los campos del formulario.

Mediante la etiqueta *fields_for* definimos el bloque de código dentro del cual se ubicarán las diversas entradas del formulario, que se procesarán conjuntamente cuando pulsemos el botón de enviarlo. El primer parámetro es esencial: define un array *respuestas* en base al objeto *respuesta* donde se almacenarán los datos que el encuestado introduzca en el formulario. Las respuestas para preguntas de tipo "Selección única" y "Texto libre" pueden almacenarse en esta variable; en ambos casos puede observarse que, para cada pregunta, se ubica el método de entrada (botón radial para selección única, iterando para abarcar todas las respuestas posibles, y campo de texto para texto libre), definiendo *:valor* como el atributo de la clase *respuesta* que estamos rellenando, además de un campo oculto para el atributo *:pregunta*, que almacenará la referencia al *encuesta_pregunta* que se está respondiendo.

En cambio, las preguntas de tipo "Selección múltiple" no pueden basarse en este método dada su propia naturaleza: el array *respuestas* anterior es capaz de almacenar una respuesta por cada entrada del formulario, simbolizada por la letra "f" al definir los campos de entrada de datos; en cambio, para las preguntas que estamos tratando es posible encontrar una respuesta con varios valores para la misma entrada del formulario. La manera encontrada para solucionar este problema consiste en utilizar un nuevo array *valores* alternativo; en él almacenaremos duplas {valor elegido, *encuesta_pregunta* asociada} para todas las casillas de verificación que se encuentren marcadas en el formulario. Es decir, en el mismo array tendremos almacenadas las múltiples respuestas de todas las preguntas que sean de este tipo.

4.6.2. Recolección de las respuestas

Una vez visto el código encargado de presentar las preguntas en pantalla, vamos a estudiar en detalle cómo funciona la recolección de las respuestas. Un detalle previo que es importante destacar es que ya en el apartado anterior se han dado algunas pinceladas dirigidas a la obtención de los datos, con ambos arrays *respuestas* y *valores* encargados de recoger la información de los distintos tipos de preguntas. Para este apartado, dividiremos la recolección de datos en dos fases fundamentales: en primer lugar, la agrupación de los datos propiamente dicha; y, por otra parte, la instanciación de los nuevos objetos a la base de datos. La figura

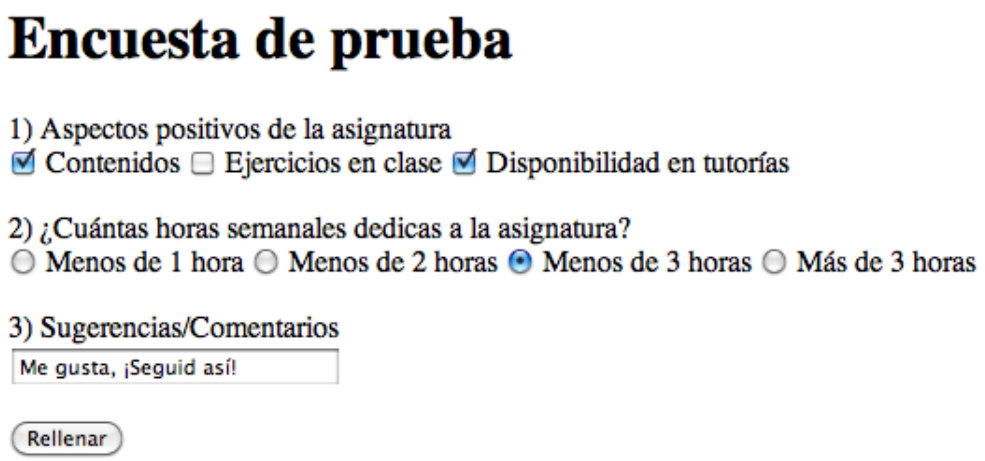


Figura 33: Vista del formulario con una encuesta cargada y preguntas introducidas.

33 muestra el aspecto que tendrá una encuesta en la vista del formulario, lista para recibir información.

```
Processing FormularioController#rellenar (for 127.0.0.1 at 2011-12-11 02:01:14) [POST]
  Parameters: {"commit"=>"Rellenar", "authenticity_token"=>"fqHVyuauQB5LkZP3tyvmmiyZ0rmpjxA
CSwqTKDdXOus=", "valores"=>["Contenidos/1", "Disponibilidad en tutorías/1"], "respuestas"=>
[{"#<RespuestaTemporal:0x103945ee0>"=>{"pregunta"=>"3", "valor"=>"Me gusta, ¡Seguid así!"},
 "#<RespuestaTemporal:0x103946138>"=>{"pregunta"=>"2", "valor"=>"Menos de 3 horas"}}]}
```

Figura 34: Traza de información ofrecida por el servidor.

Para comprender mejor en qué consiste esta fase de la aplicación, vamos a dedicar unas líneas al formato con el cual el navegador web del encuestado envía los datos introducidos en la encuesta hacia el servidor en su mensaje POST. Para ello, tomaremos como ejemplo qué sucedería si el usuario pulsase el botón *Rellenar* con las respuestas introducidas en la figura 33. Observemos detenidamente la figura 34; enmarcado en rojo, se encuentra el array *valores*, que como puede apreciarse únicamente contiene un par de elementos, separados por una coma, que no supondrán mayor complicación; el formato que muestran ambos elementos se explicará más adelante.

En cambio, el array *respuestas*, enmarcado en verde, presenta una estructura más compleja; por cada pregunta de tipo "Selección única" y "Texto libre", en-

contramos un elemento, que es otro vector en sí mismo. Realmente, estamos ante *hashes*, esto es, diccionarios con índices (a la izquierda del símbolo =>) y valores. Como podemos comprobar, se trata de varios diccionarios encapsulados uno dentro del anterior. En una aplicación convencional en Rails, podríamos recorrer únicamente el array completo, pudiendo crearse elementos nuevos de la base de datos empleando el constructor *new* utilizando como parámetro cada uno de los hashes internos (los que comienzan con la cadena *#RespuestaTemporal*).

No obstante, recordemos que en este proyecto utilizamos modelos temporales que no representan directamente una entidad de la base de datos, por lo que la conversión no es directa; el motivo por el que utilizamos este método era para evitar redundancia en la base de datos de respuestas idénticas, como veremos detalladamente más adelante. Por este motivo, necesitaremos recorrer también el array *respuestas* ignorando su estructura original interna, de manera manual.

Adquisición de datos

```
if resp != nil
for v in resp
  for w in v
  enc = 0
    for x in w
      for y in x
        for z in y
          if enc == 2
            @respuestas[i].actualizar_pregunta(z)
          elsif enc == 4
            @respuestas[i].actualizar_valor(z)
          end
          enc = enc + 1
        end
      end
    end
    i = i + 1
  end
end
end
```

Figura 35: Fragmento del código de la función *recoge_respuestas*, del fichero *temporal_respuestas.rb*.

Una vez comprendidas las motivaciones y necesidades para poder procesar las respuestas, vamos a adentrarnos en la solución implementada. En la figura 35 podemos ver la sección de código que se encarga de obtener la información encapsulada en el array *respuestas*. Al tener una estructura compleja, es necesario recorrer todos los elementos contenidos; de ahí la necesidad de contar con varios

bucles iterativos ramificados. La intención con cada uno de estos bloques es recorrer todos los elementos que componen el array, quedándonos únicamente con los que nos interesan.

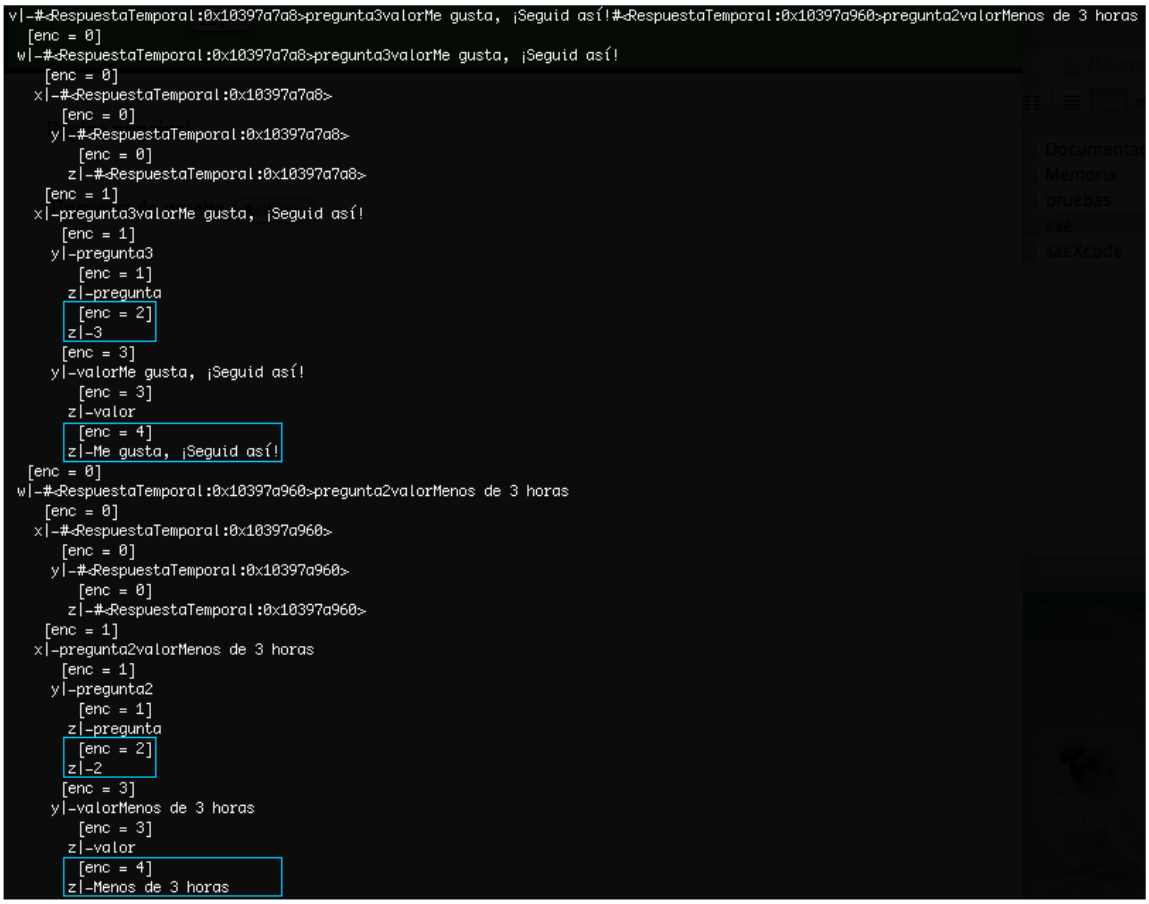


Figura 36: Traza del procesamiento en el servidor del código de la figura anterior.

En la figura 36 hemos introducido varias líneas de depuración, para poder observar cómo realiza el sistema el recorrido por los diversos elementos. Delante de cada línea se encuentra la letra que identifica en qué bucle se está en cada instante y el contenido del iterador; podemos ver cómo se van aislando los elementos hasta llegar a los más básicos. Además, contamos con un contador *enc*, que nos irá marcando en qué nivel de profundidad hemos encontrado el elemento actual. De esta manera, podremos filtrar los que a nosotros nos interesan, enmarcados en azul en la imagen: primero, el identificador de la *encuesta_pregunta* a la que pertenece la respuesta; a continuación, el valor que el visitante ha dado a dicha respuesta. De

este modo, iremos creando nuevas respuestas temporales y almacenaremos todos estos valores para posteriormente introducirlos en la base de datos.

Hasta aquí, al ejecutarse el código obtendremos tantas instancias de la clase *respuesta_temporal* como preguntas de tipo "Selección única"y "Texto libre" tuviésemos en nuestra encuesta. En la figura 37 podemos ver el fragmento de la función *recoge_respuestas* encargada de recolectar las respuestas dadas a preguntas de tipo "Selección múltiple". El cometido de esta porción del método consiste en añadir a nuestro modelo temporal tantas respuestas temporales como preguntas de respuesta múltiple haya rellenado el encuestado. Esto también incluye interpretar el formato de entrada que se da a los datos, que habíamos visto en la figura 24: por cada respuesta de este tipo, tendremos una cadena de caracteres con la estructura *valor/identificador de pregunta*. Es importante recordar que las múltiples respuestas de una pregunta se encontrarán de forma separada, formando duplas con distinto valor pero el mismo identificador, que tendremos que unir; realmente, se trata del mismo visitante que ha contestado varios valores simultáneamente a una misma pregunta, lo que supone una combinación de distinta validez a responder ambos valores por separado.

```
pregunta = 0
valor = ""
if val != nil
for elem in val
  par = elem.split('/')
  if pregunta == 0
    pregunta = par[1]
    valor = par[0]
  elsif pregunta == par[1]
    valor = valor + " y " + par[0]
  else
    @respuestas[i].actualizar_pregunta(pregunta)
    @respuestas[i].actualizar_valor(valor)
    i = i + 1
    pregunta = par[1]
    valor = par[0]
  end
end
@respuestas[i].actualizar_pregunta(pregunta)
@respuestas[i].actualizar_valor(valor)
end
```

Figura 37: Detalle del código de la función *recoge_respuestas*, del fichero *temporal_respuestas.rb*.

El motivo por el que se ha escogido el formato descrito en el párrafo anterior se puede comprender observando la 5ª línea del código extraído de la figura 37: la función *split* divide una cadena de caracteres en palabras, en base al símbolo que se introduce por parámetro. De esta manera, podemos recuperar con extrema

facilidad tanto el valor de la respuesta como su referencia a la pregunta eliminando la barra que divide ambos datos.

El comportamiento global para dicha función sería el siguiente:

- vamos recorriendo el array *valores* extrayendo los datos al procesar cada cadena de texto;
- si acabamos de empezar el proceso (la variable auxiliar *pregunta* valdrá cero), actualizamos las variables locales con los datos obtenidos;
- si el siguiente elemento también es una respuesta a la misma pregunta, significa que estamos ante una respuesta compuesta; por tanto, concatenamos los valores de ambas respuestas con la interjección "y". Esto lo hacemos para simbolizar esa múltiple respuesta.
- seguiremos concatenando valores hasta que el código de la pregunta sea distinto; almacenaremos los datos de las variables auxiliares como una nueva respuesta temporal, y continuaremos el proceso.

Almacenamiento de datos

Recapitulando todo lo comentado en este apartado, llegados a este punto tendremos en el modelo temporal tantas respuestas temporales como preguntas tuviese la encuesta. Todas las respuestas compartirán el mismo tipo de datos, y por tanto ocupan los mismos atributos; en el caso de las preguntas de tipo "Selección múltiple", damos formato al valor para simbolizar los múltiples valores si es necesario. Por tanto, tan solo nos queda volcar toda la información en el almacenamiento persistente de la base de datos.

Para facilitar el estudio de esta parte de la aplicación, nos fijaremos en la figura 38, donde puede verse todo el código involucrado en el traspaso de información desde el modelo *temporal_respuestas* hacia las distintas *respuestas*. Pero, antes de comentar su código, es importante recordar la idea perseguida; en nuestra base de datos buscamos reducir al máximo la redundancia de datos, eliminando las respuestas repetidas. Esto lo conseguiremos con el uso de contadores para representar repetición de valores.

```
for respuesta in @temporal_respuestas.respuestas
  respuestas_db = Respuesta.find(:all, :order => "encuesta_pregunta_id",
  :conditions => ["encuesta_pregunta_id = :u", {:u => respuesta.pregunta.id}])
  if respuestas_db.count() > 0
    i = 0
    existe = false
    while i < respuestas_db.count() do
      r = respuestas_db.at(i)
      if r.valor == respuesta.valor
        r.cantidad = r.cantidad + 1
        r.save
        existe = true
        break
      end
      i += 1
    end
    if not existe
      respuesta_n = Respuesta.new()
      respuesta_n.encuesta_pregunta_id = respuesta.pregunta.id
      respuesta_n.valor = respuesta.valor
      respuesta_n.cantidad = 1
      respuesta_n.save
    end
  else
    respuesta_n = Respuesta.new()
    respuesta_n.encuesta_pregunta_id = respuesta.pregunta.id
    respuesta_n.valor = respuesta.valor
    respuesta_n.cantidad = 1
    respuesta_n.save
  end
end
session[:temporal_respuestas] = nil
redirect_to "/"
```

Figura 38: Fragmento de código del método *rellenar*, del fichero *formulario_controller.rb*.

Como podemos ver, lo primero que se ejecuta dentro de la iteración sobre las *respuesta_temporal* es una búsqueda en la base de datos; concretamente, buscaremos todas las respuestas que ya tuviésemos almacenadas, cuyo identificador de la pregunta asociada coincida con la que tenga la respuesta temporal que estamos analizando. Como hemos comentado en otras secciones de esta memoria, las operaciones sobre bases de datos en Rails son de una sencillez extrema, quedando todo el código SQL recubierto por los métodos del *framework*.

A continuación, se comprueba si realmente hemos obtenido algún dato con la extracción anterior; en caso de no haber encontrado nada, nos indicaría que la respuesta que estamos tramitando es la primera de la encuesta; por tanto, almacenamos directamente su valor como una nueva entrada. Es en caso de que sí existan respuestas previas cuando tenemos que realizar las operaciones más relevantes: al existir la posibilidad de que la respuesta que pretendemos insertar ya haya sido

añadida con anterioridad, necesitamos recorrer todas las instancias. En caso de encontrar la duplicidad, simplemente aumentaremos el atributo contador *cantidad*; si la respuesta que estamos insertando no tiene un equivalente ya guardado, procedemos a crear una nueva. De nuevo, la potencia de Ruby on Rails permite simplificar una operación UPDATE a un par de instrucciones convencionales.

Con estos bloques de código habremos conseguido resolver uno de los mayores requisitos de nuestro sistema. Por tanto, tan solo queda eliminar nuestra entidad temporal y devolver el control al controlador *main*, mostrando la página inicial.

4.6.3. Mostrando los resultados obtenidos

Para concluir con las explicaciones pertenecientes a este módulo de nuestro proyecto, vamos a estudiar de qué modo puede un usuario registrado comprobar las respuestas que han ido dando los encuestados. El proceso será muy sencillo: contaremos con una única vista parcial *_cargar_resutados.rb*, que se procesará al invocar al método del mismo nombre en el controlador dedicado *Resultados*. Junto a cada uno de los valores que han ido proporcionando los visitantes, mostraremos el número de ocurrencias e esa respuesta específica. La figura 39 muestra el aspecto de esta vista.

Gracias a la estructura que hemos proporcionado a nuestro modelo, simplemente tendremos que buscar en la base de datos la encuesta que ha seleccionado el usuario, y llegar hasta las respuestas utilizando todas las claves ajenas que almacenan cada una de las entidades.

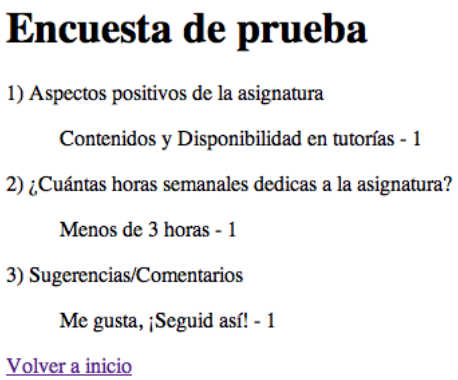


Figura 39: Vista *_cargar_resultados* con algunos datos cargados.

5. Pruebas

En los apartados anteriores se ha hecho un desglose detallado de todas las fases de desarrollo de este proyecto.

- Por un lado, se ha abordado el análisis del problema que se ha pretendido solventar (sección 2.1), presentando los objetivos y requisitos que era necesario satisfacer (sección 2.2), y finalmente delimitando los posibles casos de uso que se contemplan (sección 4.2);
- Por otra parte, hemos podido estudiar el diseño de la aplicación. Esto abarca desde la estructura global de la misma bajo el paradigma MVC (sección 4.3), hasta el diseño relacional de la base de datos para almacenar la información relevante del sistema (sección 4.4);
- Finalmente, se ha detallado la implementación llevada a cabo, haciendo énfasis en las partes más delicadas y funcionalmente relevantes para el correcto desempeño del proyecto (secciones 4.5 y 4.6).

Además, se ha podido comprobar el resultado obtenido en el apartado 4.2, con un ejemplo de uso para cada uno de los roles posibles. Dichos ejemplos se realizaron directamente sobre la aplicación final, haciendo las capturas de pantalla en los instantes que se consideraron claves del progreso de la aplicación, mostrando la actividad de los usuarios y cómo evoluciona el contenido.

Adicionalmente, en el anexo D se muestra la evolución de la base de datos, y el producto de las acciones ejecutadas en el caso de uso mencionado.

6. Conclusiones y trabajos futuros

Tras estudiar el diseño e implementación de este proyecto en la presente memoria, vamos a tratar las conclusiones finales que podemos extraer del resultado desarrollado. Por último, también repasaremos qué posibilidades de ampliación y mejora podría tener la aplicación.

6.1. Conclusiones

Tal y como hemos ido desgranando a lo largo de esta memoria, hemos desarrollado una aplicación web interactiva que facilita las labores de la universidad a la hora de obtener información de sus alumnos por medio de las encuestas. El sistema permite generar los cuestionarios personalizando las preguntas y el tipo de respuesta que se espera de ellas, de modo que se pueda obtener la información precisa que se requiera en cada momento. Todo ello sin olvidar la usabilidad, utilizando varias librerías y tecnologías que facilitan una experiencia de usuario más agradable.

En el desarrollo de este proyecto hemos cumplido con los objetivos planteados para este proyecto:

Ante todo, el cometido fundamental era proveer una herramienta web capaz de gestionar todo el proceso de creación y publicación de encuestas, desde su primer diseño hasta su puesta en producción. Para cumplirlo, contamos con varios módulos que se encargan de cada una de las áreas de desarrollo: contamos con un editor de encuestas completamente flexible, permitiendo la inserción de preguntas ya existentes o de otras completamente nuevas, sin olvidar la personalización de las respuestas. Este mismo editor también permite almacenar una encuesta en una versión previa a su estado final, de modo que el trabajo de creación pueda continuar en cualquier momento.

Por otra parte, contamos con un módulo dedicado a la generación del cuestionario que podrá rellenar cualquier encuestado, respetando el formato elegido por el creador de la misma. El acceso a las encuestas puede controlarse mediante una URL que nos facilita el propio sistema, de modo que los usuarios puedan llegar sin ningún problema al formulario en cuestión.

Todas estas especificaciones llevaban intrínsecamente algunos requisitos y res-

tricciones que ha sido necesario tener en cuenta:

- *Desarrollar la aplicación utilizando el framework Ruby on Rails.* Ha sido necesario utilizar el paradigma MVC para desarrollar toda la funcionalidad descrita en este proyecto. Como resultado, tenemos una jerarquía de directorios de diverso contenido, desde los ficheros *.rb* con código escrito íntegramente en lenguaje Ruby, pasando por las vistas en *.html.erb*, combinando HTML, Ruby y hojas de estilo, hasta finalmente algunos retales de Javascript en ficheros *js.rjs*.

- *Mantener el anonimato de los encuestados.* Hemos considerado la privacidad como uno de los requisitos clave de este sistema, dada la sensibilidad de la información recogida. Este objetivo se ha logrado eliminando la necesidad de registrarse como usuario en el sistema para poder rellenar una encuesta; de esta manera, la base de datos no registra ninguna información personal del visitante que envía los datos de un formulario. Únicamente se ha necesitado el uso de usuarios registrados para los creadores de encuestas, con la finalidad de poder identificarlos y presentarles sus cuestionarios en el estado en que se encontrasen, además de poder acceder a sus resultados.

- *Crear una aplicación autocontenida y completa.* Este requisito consiste en lo que hemos comentado hace unos instantes: procurar que la aplicación sea capaz de gestionar todo el proceso de las encuestas de forma autónoma, sin depender de otros desarrollos externos.

- *Mostrar una interfaz usable.* Finalmente, hemos tenido que llevar el desarrollo a una plataforma que pudiera proveer una experiencia web 2.0 al usuario, con elementos interactivos y efectos visuales. Prácticamente todo el esfuerzo ha quedado mitigado gracias al primero de los requisitos: el uso de Ruby on Rails facilita enormemente la integración de tecnologías como AJAX, permitiendo presentar una interfaz dinámica y acorde a nuestras necesidades.

Todo este trabajo ha desembocado en aproximadamente un año y medio de desarrollo, incluyendo el aprendizaje previo de las herramientas y lenguajes necesarios.

Como conocimientos adquiridos, cabe destacar sin duda alguna el desarrollo íntegro de una aplicación web, desde sus primeras fases de diseño hasta sus detalles de implementación; pasando por la resolución de problemas y depuración de

errores; hasta las fases de pruebas y posterior inserción de mejoras y nuevos módulos. Es también importante destacar el aprendizaje del framework Ruby on Rails, con una metodología de trabajo bastante diferenciada de otras más convencionales, como PHP; el uso de MySQL como gestor de base de datos, y la instalación y mantenimiento de un servidor; y algunas otras herramientas, como el editor de documentos LaTeX, utilizado para la presente memoria.

6.2. Trabajos futuros

Este proyecto fin de carrera nos brinda múltiples posibles vías de mejora y nuevas funcionalidades que podrían desarrollarse para mejorar el producto.

Como posible proyecto más inmediato podríamos destacar la mejora de la interfaz visual general; en el proyecto nos hemos centrado en la parte más técnica y funcional del sistema, relegando el aspecto a ofrecer una vista mínimamente usable para utilizar los procesos básicos. Algo interesante podría ser mejorar la presentación, de modo que pudiera resultar más intuitiva y práctica al usuario final. Esto podría incluir guías de uso interactivas, además de una presencia más intensiva de hojas de estilo.

Uno de los aspectos que quizá tengan un margen de mejora más amplio es la confidencialidad de las respuestas. La solución implementada, a pesar de ser eficaz, presenta algunos inconvenientes, como el no poder controlar qué usuarios ya han respondido un cuestionario. Sería necesario encriptar la información de los usuarios, que estarían registrados en el sistema, de modo que podríamos limitar su participación a una única vez por encuesta; otras posibles vías podrían incluir analizar la IP desde la que se accede a un formulario.

Finalmente, este sistema es sensible a desarrollar múltiples mejoras y ampliaciones de funcionalidad, desde la gestión de las encuestas en la base de datos, hasta incluir nuevos formatos de respuestas y preguntas. También podría ser muy interesante permitir la inclusión de material adicional, como por ejemplo multimedia, con la idea de explotar la naturaleza online de la aplicación y mejorar la interacción de los usuarios.

6.3. Esfuerzo dedicado

Para concluir la memoria del presente proyecto final de carrera, daremos unos datos sobre el rendimiento y dedicación empleados durante el desarrollo. Para ello, basaremos la explicación en una serie de gráficos y diagramas que facilitarán la comprensión. Hablaremos, además, de los prototipos desarrollados como fases previas a la versión final (tal y como se introdujo en la sección 2 de objetivos del proyecto).

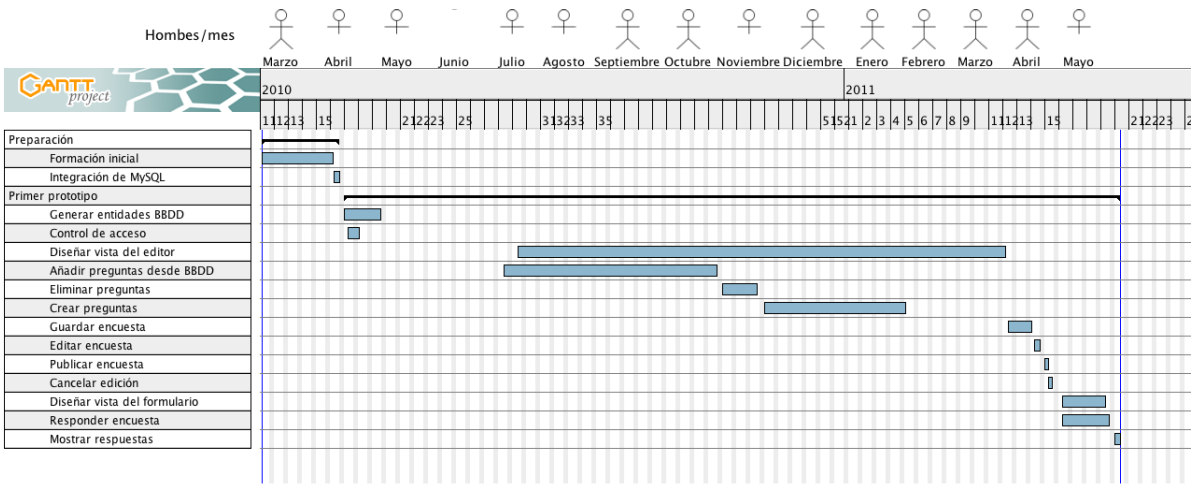


Figura 40: Diagrama de Gantt de tiempos de desarrollo para el primer prototipo.

En la figura 40 podemos ver el diagrama de Gantt que comprende el periodo de tiempo entre Marzo de 2010 y Mayo de 2011. En dicha franja temporal se ubicó el desarrollo completo del primer prototipo plenamente funcional del sistema. También se han añadido los hombres/mes relativos a cada periodo, con la intención de reflejar la dedicación personal de cada momento. Hay algunas observaciones que es necesario realizar para comprender toda la información contenida en el mismo:

- Este prototipo contempla la siguiente funcionalidad:
  - Control de acceso de los usuario mediante nombre de usuario y contraseña.
  - Editor de encuestas, capaz tanto de crear encuestas completamente nuevas, como de editar las ya existentes, o partir de una encuesta previa.

- Añadir preguntas residentes en la base de datos, o crear elementos nuevos.
- Funcionamiento mediante arrastrar y soltar elementos en diferentes áreas de las vistas.
- Guardar encuestas en estados intermedios para poder editarlas.
- Cancelar la edición de una encuesta, deshaciendo todos los cambios.
- Publicar encuestas para su resolución por parte de otros usuarios.
- Presentar la encuesta para rellenarla.
- Mostrar los resultados recogidos al usuario que creó la encuesta.

Se trata, por tanto, de una versión prácticamente completa. Este es uno de los motivos por los que se ha tardado tanto tiempo en completarla.

- En el período comprendido entre abril y julio, se ubicaron mis últimos exámenes y prácticas de la carrera, y en ellas volqué todo mi esfuerzo, a fin de completar todos los créditos pendientes de mi titulación.
- Puede observarse que las tareas *Diseñar vista del editor* y *Añadir preguntas desde BBDD* se alargaron varios meses en el tiempo. Se debió a un problema bastante complejo de resolver. Debe notarse, no obstante, que la primera tarea mencionada estaba estrechamente relacionada con las tres siguientes, y cualquier avance estaba subyugado a ellas. Es por ello que aparece como una tarea de muy larga duración.

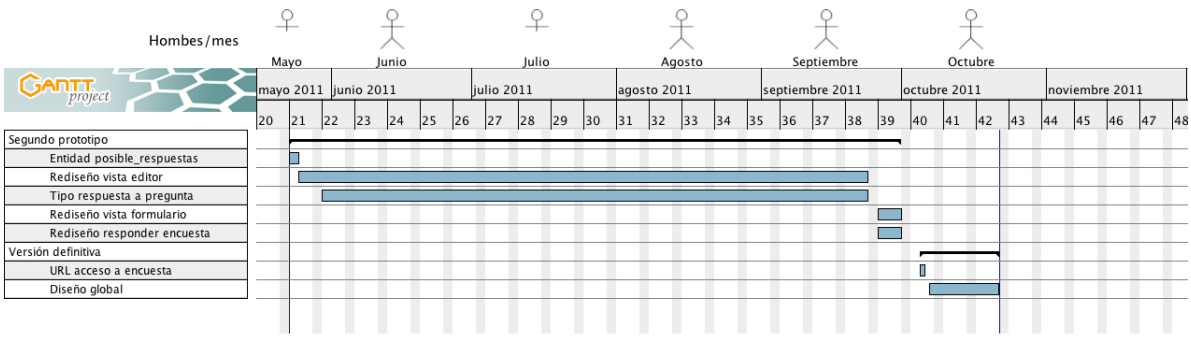


Figura 41: Diagrama de Gantt de tiempos de desarrollo para el segundo prototipo y versión final.

En la figura 41 podemos ver el diagrama de Gantt que comprende el periodo de tiempo entre Mayo de 2011 y Octubre de 2011, con los hombres/mes relativos

a cada mes. En esta última fase del proyecto se implementó una segunda versión intermedia, y el producto final. Al contar con un prototipo que prácticamente incluye toda la funcionalidad final, la cantidad de tareas necesarias es bastante inferior. Hagamos algunas observaciones:

- El segundo prototipo incluye, además de la ya implementada, la posibilidad de incluir preguntas con distintos tipos de respuestas: selección única, selección múltiple y respuesta libre.
- A pesar de ser una única tarea, tuvo una complejidad de implementación bastante importante, como se observa en el diagrama; de ahí la decisión de partir el desarrollo en un segundo prototipo para implementar esta funcionalidad.
- La versión final incluye, además de lo ya expuesto, algunos retoques. El más importante es el acceso a las encuestas mediante una URL. Además, se trabajó ligeramente en el diseño de las vistas, a fin de tener una presentación algo mejor.

Referencias

[1] S. Ruby, D. Thomas, and D. Heinemeier, *Agile Web Development with Rails*. The Pragmatic Programmers, 2009.

[2] D. Thomas and A. Hunt, *Programming Ruby: The Pragmatic Programmer's Guide*. `http://www.ruby-doc.org/docs/ProgrammingRuby/`.

[3] D. Heienmeier, "Ruby on rails guides." `http://guides.rubyonrails.org/index.html`.

[4] R. Bates, "Railscasts." `http://railscasts.com/`.

[5] "w3schools." `http://www.w3schools.com/`.

Anexos

A. Estado del arte: Google Docs

Dentro de la amplia oferta de aplicaciones web que ofrece la compañía californiana, *Google Docs*[5] es un completo editor de documentos en la nube, entre cuyas características se incluye la edición colaborativa de archivos. Y es precisamente aprovechando esta posibilidad donde toma sentido la creación de encuestas. Para ello, *Google Docs* ofrece la funcionalidad de generar formularios.

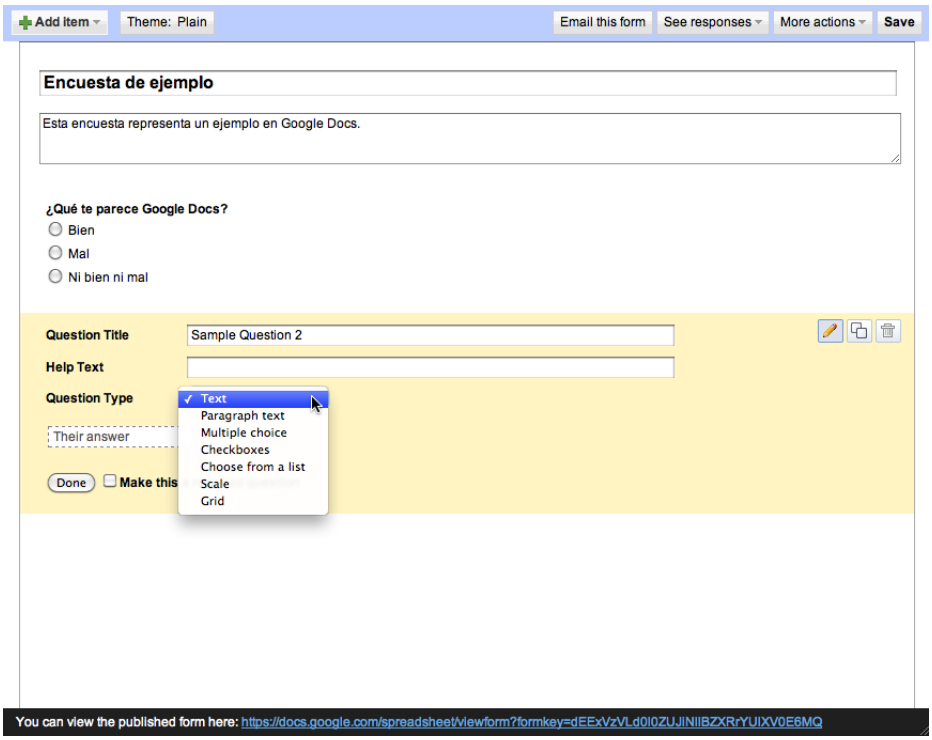


Figura 42: Página de edición de encuestas.

Seleccionando *form* como tipo de documento, se nos abre directamente una pantalla de edición de encuesta como la presente en la figura 42. Como podemos observar, en ella contamos con todos los elementos necesarios para la edición del

[5]https://docs.google.com/

formulario, como el campo para introducir el título y una breve descripción general del contenido. Para añadir preguntas, simplemente pulsamos sobre los campos correspondientes que se encuentran sin definir. De este modo, cada vez que insertemos una pregunta se nos ofrecerá otro conjunto nuevo de campos, de manera que podamos repetir la operación un número indefinido de veces. En cualquier caso, contamos con un botón en la esquina superior para poder añadir nuevas preguntas.

Figura 43: Apariencia de la encuesta para un usuario que recibe el enlace.

Cada una de las preguntas es completamente personalizable, contando con una amplia variedad de tipos de preguntas disponibles, tal y como se veía en la figura anterior. Dependiendo del tipo de respuestas esperadas, definiremos distintos valores posibles. Y, una vez tengamos creada una pregunta, tenemos la posibilidad de duplicarla pulsando un botón, editarla, o incluso reordenarlas simplemente arrastrándolas en el orden deseado. En este sentido, la aplicación se muestra muy versátil, con multitud de posibilidades que no se limitan únicamente a las preguntas: del mismo modo, se pueden editar otros detalles, como el mensaje que recibirán los participantes al rellenar el formulario, u ofrecer la posibilidad de insertar un *widget* en nuestra página web personal que contenga la encuesta.

Una vez tenemos seleccionadas las preguntas que queríamos, podremos invitar

a cualquier persona a rellenar la encuesta simplemente enviándole un correo electrónico. Este destinatario no tiene por qué estar registrado en Google para poder contestar a las preguntas, a diferencia del creador, que sí necesita una cuenta forzosamente. En la figura 43 podemos ver el aspecto que presentará el formulario para los receptores de la invitación. En dicha ventana podrán rellenar el cuestionario, enviando sus respuestas de vuelta al usuario creador.

Una vez que los usuarios han rellenado el formulario, podremos comprobar los resultados de dos formas diferentes:

- De forma resumida: en este modo, la información se muestra de modo compacto, para poder hacer una evaluación rápida de los datos obtenidos. Para ayudar en esta tarea, nos presentan diversos tipos de gráficos ilustrativos, incluyendo la tendencia de respuestas a lo largo del tiempo. En la figura 44 podemos ver algunos de los datos recogidos por nuestra encuesta de ejemplo.
- Como hoja de cálculo: si se pretende utilizar la información en algún tipo de procesado posterior, *Google Docs* almacena por defecto todos los resultados como una hoja de cálculo. En la figura 45 se muestran los datos recogidos en nuestro ejemplo como hoja de cálculo.

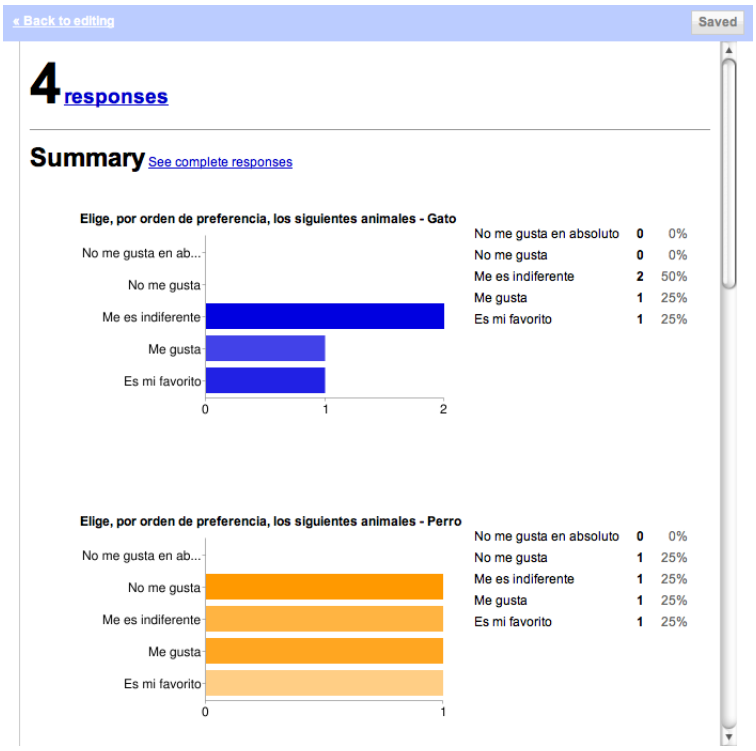


Figura 44: Presentación resumida de las respuestas a una encuesta.

Encuesta de ejemplo ☆

File Edit View Insert Format Data Tools Form (4) Help Last edit was made 3 hours ago by cperez.camarasa

$ % 123 10pt

fx Timestamp

	A	B	C	D	E	F	G	H	I
1	Timestamp	¿Qué te parece Google Docs?	Sample Question 2	¿Qué te parece Google Docs?	Elige, por orden de preferencia, los siguientes animales [Gato]	Elige, por orden de preferencia, los siguientes animales [Perro]	Elige, por orden de preferencia, los siguientes animales [Hamster]	Elige, por orden de preferencia, los siguientes animales [Caballo]	¿Cuál es tu deporte favorito?
2	1/23/2012 23:21:42			Bien	Me gusta	Me gusta	Me gusta	Me gusta	Baloncesto
3	1/23/2012 23:37:10			Ni bien ni mal	Me es indiferente	No me gusta	No me gusta en absoluto	Me gusta	Curling
4	1/24/2012 0:09:05			Ni bien ni mal	Es mi favorito	Es mi favorito	Me gusta	Me gusta	baloncesto
5	1/24/2012 18:30:56			Bien	Me es indiferente	Me es indiferente	Es mi favorito	No me gusta en absoluto	volleyball

Figura 45: Presentación en formato de tabla de cálculo de las respuestas a una encuesta.

B. Estado del arte: dudle

dudle[6] es una aplicación *online* de código abierto, que permite crear encuestas de forma muy sencilla e inmediata. Una de sus características más interesantes es la no necesidad de registrarse para poder publicar un sondeo. En la figura 46 se muestra la página inicial de la web. Además, se explican algunas de sus características, como seleccionar la plantilla de encuesta entre las dos disponibles (encuesta de fecha o genérica), personalizar la URL de acceso, y el limpiado automático de la información pasados 3 meses de inactividad. También disponen de una versión que cuida el anonimato de los participantes, manteniendo la misma funcionalidad (incluyendo las carencias que describiremos en adelante).

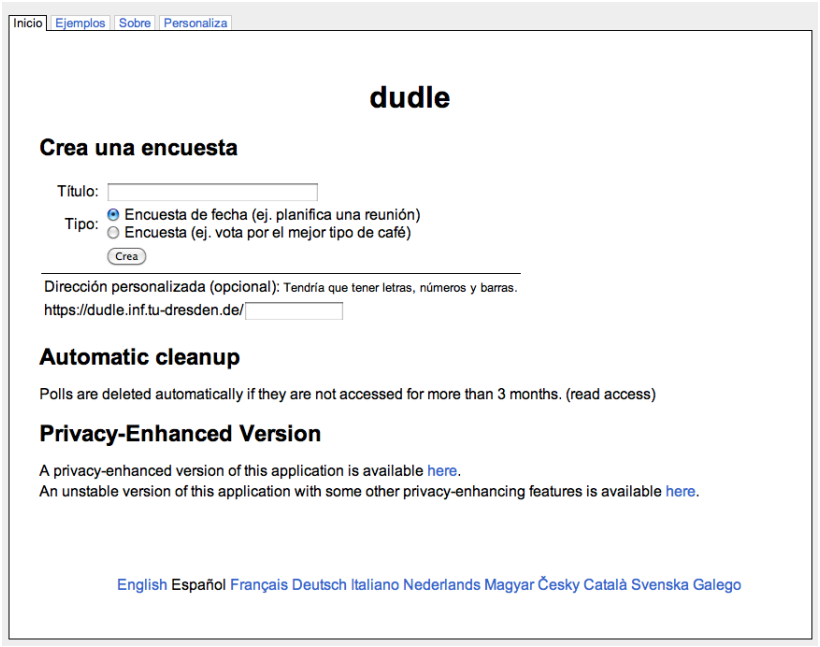


Figura 46: Página inicial de la aplicación online dudle.

Para este ejemplo, mostraremos cómo se puede crear una encuesta de tipo genérico. El motivo de esta elección es que, de las dos alternativas posibles, se trata de la más abierta. Se nos permite crear todas las opciones que queramos, ante las cuales los encuestados responden en base a unos valores predeterminados y no editables. La encuesta de fechas, por contra, únicamente permite definir fechas mediante un calendario. Puede observarse que, en cualquiera de las modalidades,

[6] https://dudle.inf.tu-dresden.de/

la flexibilidad no es una de las mejores propiedades de esta plataforma.

Una vez que hemos creado la encuesta, nos muestran la página para introducir las diferentes opciones sobre las cuales los usuarios podrán votar. Se puede añadir una descripción informativa para cada una de las opciones. También se permite editar todos estos datos a posteriori sin mayor complicación. En la figura 47 podemos ver nuestra encuesta de prueba con algunas opciones insertadas.

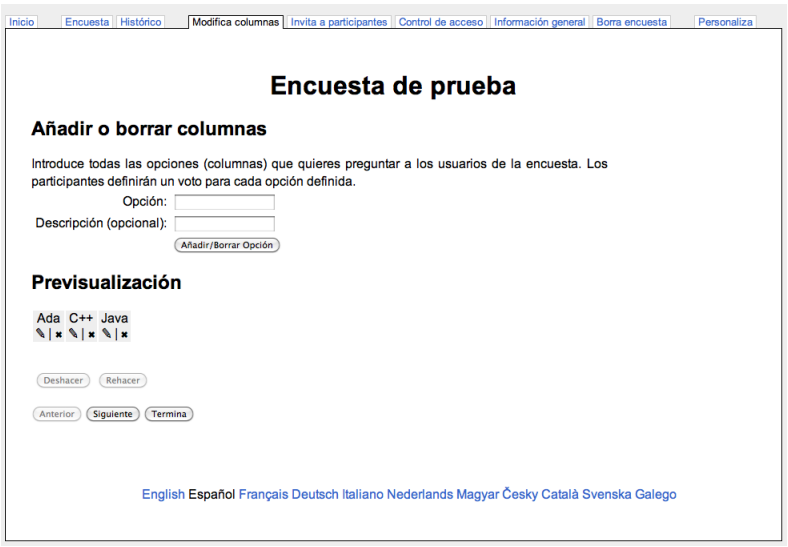


Figura 47: Página de inserción de opciones.

Tal y como se observa en la figura 48, al pulsar le botón *siguiente*, entramos en la página para añadir los usuarios que podrán rellenar la encuesta. Al no contar con control de acceso, la única manera de identificar a los usuarios será introduciendo nosotros mismos sus nombres. Luego, será cada uno de ellos el que tendrá que contestar en su nombre. Igualmente, se permite editar esta lista de nombres en todo momento.

En este punto, cualquier usuario podría suplantar la identidad de cualquier otro y falsificar las respuestas. Esto también incluye al propio administrador; figura que funcionalmente no existe, por lo que cualquier persona con acceso a la URL podría alterar la encuesta. Con la intención de subsanar en parte estos problemas, al avanzar al siguiente paso se nos permite la posibilidad de controlar el acceso de los usuarios. Para ello, en primer lugar creamos una contraseña de administrador, garantizando que nadie podrá alterar el contenido. A continuación, como ilustra la figura 49, definiremos una contraseña global de acceso para el resto de usua-

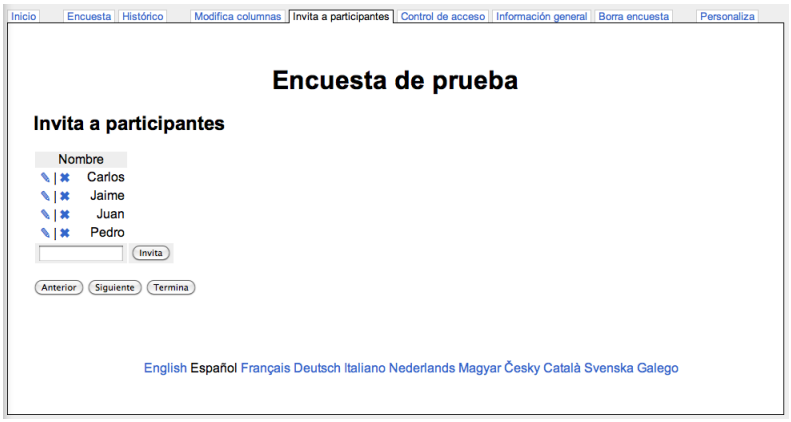


Figura 48: Página de inserción de participantes.

rios. Tras una confirmación, podemos terminar la edición de la encuesta, aunque podremos editarla en todo momento.

El acceso a nuestra encuesta de prueba es tan sencillo como a través de la dirección personalizada. Al introducir la URL, se nos pedirán las credenciales, ya sean de administrador o de usuario convencional. Una vez hemos accedido, introduciendo el nombre del usuario correspondiente del listado, o pulsando directamente sobre la imagen del lápiz de cualquiera de ellos, podremos seleccionar los valores deseados para cada una de las opciones posibles. En la figura 50 podemos ver nuestra encuesta con varias respuestas.

Sin embargo, recordemos que el acceso de los participantes de la encuesta se encuentra únicamente regulado por una contraseña global de acceso (además de una para administración), que impide que personas indeseadas el poder acceder. Es decir, todo usuario (incluido el administrador) tiene la potestad de cambiar los resultados de cualquiera de los participantes. Si no se confía plenamente en todas las personas a las que se facilitaron las credenciales, o si sospechamos que éstas pueden haber llegado a manos de gente no deseada, los resultados obtenidos son poco fiables. Además, el listado de personas no es en absoluto cerrado: cualquier usuario puede introducir nuevas respuestas para un nombre no definido a priori, o incluso puede eliminar a cualquiera de ellos. Para muestra, en la figura 50 se ha alterado el listado original de participantes desde un acceso sin contraseña.

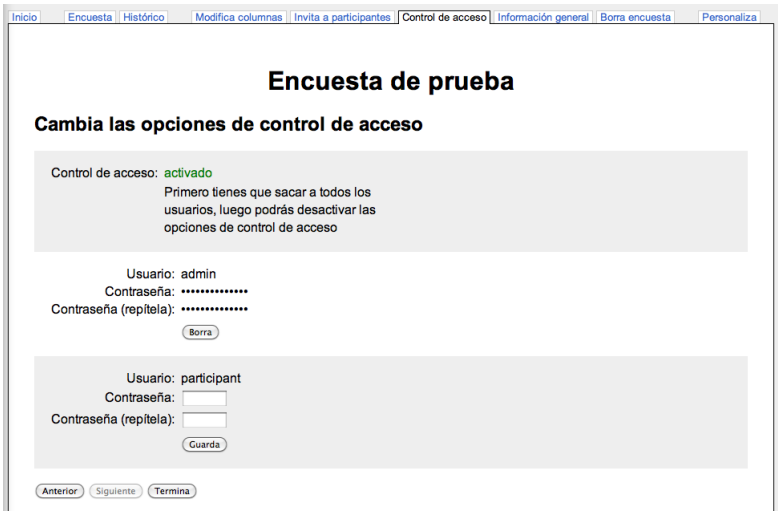


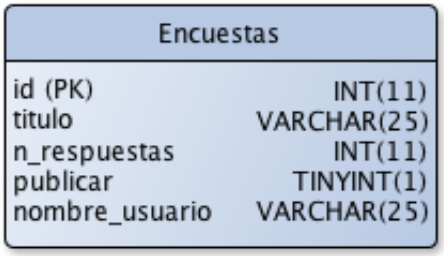
Figura 49: Vista de selección de credenciales.

Figura 50: Vista con el formulario modificado.

C. Entidades del modelo relacional

En este anexo se muestran todas estas entidades de nuestra base de datos en detalle. Dado que todas ellas cuentan con el atributo *id* en común, se obviará en todos los casos.

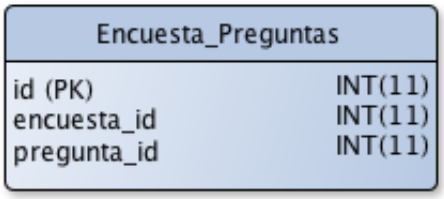
Entidad *Encuestas*



Esta entidad pretende representar la encuesta que crea un usuario registrado, entendiéndolo como contenedor al cual se incorporarán las preguntas que se deseen. Cuenta con los siguientes atributos:

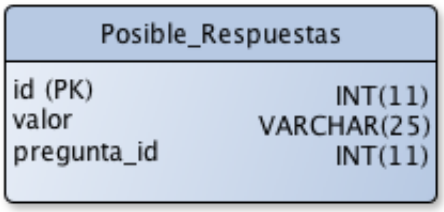
- *titulo*: título que el usuario ha dado a la encuesta. Este atributo no puede quedar en blanco, por lo que será necesario realizar un control en la aplicación para evitar que el usuario pueda almacenar una encuesta sin nombre.
- *n_respuestas*: número de respuestas que se han dado a las preguntas de la encuesta, una vez se ha publicado.
- *publicar*: este atributo indica si la encuesta está publicada o no para poder rellenarla.
- *nombre_usuario*: nombre del usuario que creó la encuesta. Se trata únicamente de una cadena de caracteres que contiene el nombre con que se dio de alta al usuario en el sistema; no se ha considerado necesario ubicar aquí una clave ajena a la entidad *users*, ya que no necesitaremos acceder a ninguno de los demás atributos de ésta.

Entidad *Encuesta_ preguntas*



En nuestro sistema, una encuesta puede contener varias preguntas y, a su vez, una pregunta puede estar contenida en varias encuestas de manera simultánea. Al existir una relación N:M entre las preguntas y las encuestas, se hace necesario utilizar una entidad auxiliar que establezca dicha conexión. Como puede verse en la figura, cuenta con dos atributos que servirán como clave ajena a cada una de las clases pertenecientes a la relación.

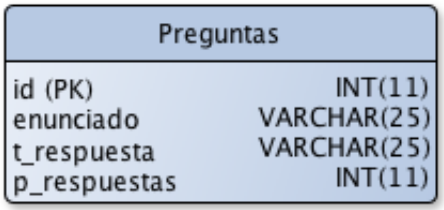
Entidad *Posible_ respuestas*



Una de las funcionalidades más interesantes y potentes de la aplicación será poder crear encuestas con preguntas completamente editables. Esta entidad representa el valor de cada una de las posibles respuestas que tendrá una pregunta, como se verá en el detalle de dicha clase. Por tanto, necesitaremos almacenar el valor concreto de una respuesta (en su atributo de idéntico nombre) y una referencia a la pregunta a la que pertenece. Esta entidad no representa el valor elegido por un

usuario al rellenar una encuesta, con ese fin se ha diseñado la clase *Respuestas*; sino las posibles opciones que tendrá disponibles.

Entidad *Preguntas*

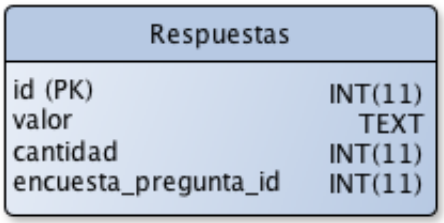


Junto con *Encuestas*, esta entidad es una de las más importantes de la aplicación. Representaremos en ella las preguntas como elementos que pueden incorporarse a una encuesta. Cuenta con los siguientes atributos:

- *enunciado*: enunciado de la pregunta. Este atributo no admite valores nulos, por lo que en la aplicación será necesario llevar un control.
- *t_respuesta*: en esta variable se almacenará el tipo de respuestas que admite la pregunta. En nuestra aplicación se podrán definir preguntas que contemplen tres tipos de respuestas: selección única, en la que el usuario únicamente podrá elegir una de las opciones; selección múltiple, donde podrá elegir varias o ninguna; y texto libre, en la que podrá introducir su propia respuesta libremente. Los valores de dichas posibles respuestas vienen representados por la entidad *Posible_respuestas*, descrita anteriormente. Nótese que, en el caso de las preguntas de tipo respuesta libre, no existirán posibles respuestas.
- *p_respuestas*: número de respuestas posibles que tiene la pregunta.

Entidad *Respuestas*

Una vez que un usuario registrado publica una encuesta, está disponible para poder completarla. Los valores enviados al servidor desde el formulario de la



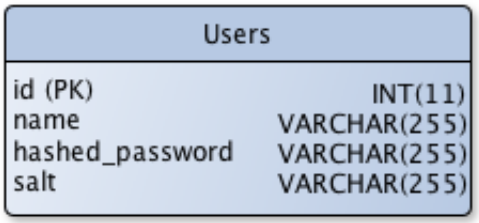
encuesta se almacenarán bajo esta entidad, que cuenta con los siguientes atributos:

- *valor*: valor elegido entre los posibles en respuesta a una pregunta, o el texto libre en su caso.
- *cantidad*: número de veces que se repite la misma respuesta en la base de datos.
- *encuesta_pregunta_id*: clave ajena hacia la entidad *encuesta_pregunta*.

Este diseño de la entidad tiene varias repercusiones positivas. Su principal cometido es reducir la redundancia de respuestas, un fenómeno que va a ser muy común en esta aplicación; por cada pregunta de cada encuesta, hay muchas probabilidades de que múltiples usuarios introduzcan la misma respuesta. Al no contener ningún identificador del usuario que introdujo dicha respuesta, todas las que se emitan serán completamente idénticas, salvo por su clave primaria *id*. Por ello, desde el controlador *Formulario* que recoge la información se llevará a cabo una búsqueda en la base de datos: si la respuesta no existe, se creará una nueva instancia; si ya existía una respuesta con idéntica información tanto en *valor* como en *encuesta_pregunta_id*, simplemente se aumentará *cantidad* para reflejarlo. La clave para el buen funcionamiento de este método es la referencia simultánea a las entidades *Encuesta* y *Pregunta*, lo que garantiza que la misma pregunta en otra encuesta tendrá almacenadas instancias de respuestas independientes.

También es interesante la transparencia que este método ofrece para nuevos tipos de respuestas que puedan añadirse a una encuesta; esta entidad no se ve afectada, almacenando de la misma forma tanto preguntas de selección única como de texto libre, o cualquier otra nueva clase que queramos introducir como nueva funcionalidad.

Entidad *Users*



La última entidad de nuestro sistema es la encargada de almacenar los datos de los usuarios que estén registrados. Como puede verse, no se pretende almacenar un perfil completo, sino simplemente los datos necesarios para el acceso y manipulación de encuestas: nombre de usuario, el código *salt* que se combinará con la contraseña introducida en el formulario de acceso, y posteriormente se comprobará si coincide con *hashed_password* al aplicarle la encriptación; en esta aplicación utilizaremos SHA-1 para generar las contraseñas. Finalmente, con la intención de reforzar aún más la contraseña almacenada, se concatenará la cadena de caracteres *"palascuanda"*, escogida de forma azarosa. En la figura 51 se muestra el método encargado de encriptar una contraseña de entrada aplicando un código salt.

```
def self.encrypted_password(password, salt)
  string_to_hash = password + "palascuanda" + salt
  Digest::SHA1.hexdigest(string_to_hash)
end
```

Figura 51: Método *encrypted_password*, encargado de generar contraseñas.

D. Evolución de la BBDD

Durante todo el proceso descrito en el apartado 2.3., han tenido lugar diversas modificaciones en la base de datos para mantener la coherencia de la información. En las siguientes figuras podemos comprobar el estado final de todas las tablas del sistema, tras todas las acciones descritas Los estados intermedios no sería más que las mismas tablas con algunas filas de información sin detallar. Así, en las dos figuras 52 y 53 encontramos el contenido de las tablas de encuestas y preguntas, respectivamente.

```
mysql> select * from encuestas;
+----+---------------------------+--------------+---------------------+---------------------+----------+----------------+
| id | titulo                    | n_respuestas | created_at          | updated_at          | publicar | nombre_usuario |
+----+---------------------------+--------------+---------------------+---------------------+----------+----------------+
|  1 | Encuesta de prueba        |         NULL | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |        1 | user           |
|  3 | Copia de Encuesta de prueba |         NULL | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |        1 | user           |
+----+---------------------------+--------------+---------------------+---------------------+----------+----------------+
2 rows in set (0.00 sec)
```

Figura 52: Estado final de la tabla *Encuestas* en la base de datos.

```
mysql> select * from preguntas;
+----+-------------------------------------------------+---------------------+---------------------+---------------+-------------+
| id | enunciado                                       | created_at          | updated_at          | t_respuesta   | p_respuestas |
+----+-------------------------------------------------+---------------------+---------------------+---------------+-------------+
|  1 | Aspectos positivos de la asignatura             | 2011-12-07 21:10:02 | 2011-12-07 21:10:02 | Selecci?n m?ltiple |         NULL |
|  2 | ?Cu?ntas horas semanales dedicas a la asignatura? | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 | Selecci?n ?nica    |         NULL |
|  3 | Sugerencias/Comentarios                         | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 | Texto libre        |         NULL |
|  5 | ?Has programado antes en este lenguaje?         | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 | Selecci?n ?nica    |         NULL |
+----+-------------------------------------------------+---------------------+---------------------+---------------+-------------+
4 rows in set (0.00 sec)
```

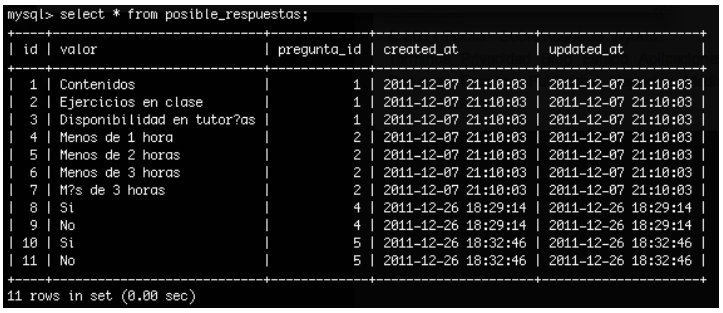
Figura 53: Estado final de la tabla *Preguntas* en la base de datos.

En la figura 54 podemos observar cómo nuestra aplicación relaciona cada pregunta con una o varias encuestas concretas, utilizando una relación de tipo *N:M*, tal como se explicó en la sección correspondiente. En este caso, tenemos varias preguntas que son compartidas por ambas encuestas.

```
mysql> select * from encuesta_preguntas;
+----+-------------+-------------+---------------------+---------------------+
| id | encuesta_id | pregunta_id | created_at          | updated_at          |
+----+-------------+-------------+---------------------+---------------------+
|  1 |           1 |           1 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  2 |           1 |           2 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  3 |           1 |           3 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  8 |           3 |           1 | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |
|  9 |           3 |           5 | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |
| 10 |           3 |           2 | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |
| 11 |           3 |           3 | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |
+----+-------------+-------------+---------------------+---------------------+
7 rows in set (0.00 sec)
```

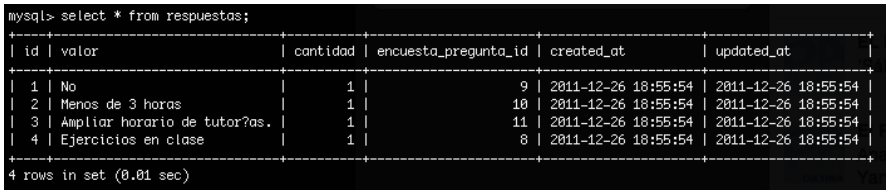
Figura 54: Estado final de la tabla *Encuesta_preguntas* en la base de datos.

En la figura 55, encontramos las posibles respuestas que se han asignado para cada una de las preguntas. A diferencia de lo que sucede con las respuestas (figura 56), en esta tabla no tenemos una única entrada para múltiples valores idénticos; esto se debe fundamentalmente a que, en el caso de las respuestas, la multiplicidad se da en una misma pregunta y su respuesta concreta, por lo que siempre estamos referenciando únicamente a un elemento; por ello, basta con un nuevo atributo en la entidad. En cambio, para las posibles respuestas, un mismo valor se puede repetir en varias preguntas diferentes; esta información podría reflejarse a través de una nueva relación *N:M* en nuestra base de datos, alternativa que hemos considerado sin beneficio frente a almacenar datos redundantes.



```
mysql> select * from posible_respuestas;
+----+-----------------------------+-------------+---------------------+---------------------+
| id | valor                       | pregunta_id | created_at          | updated_at          |
+----+-----------------------------+-------------+---------------------+---------------------+
|  1 | Contenidos                  |           1 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  2 | Ejercicios en clase         |           1 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  3 | Disponibilidad en tutor?as  |           1 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  4 | Menos de 1 hora             |           2 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  5 | Menos de 2 horas            |           2 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  6 | Menos de 3 horas            |           2 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  7 | M?s de 3 horas              |           2 | 2011-12-07 21:10:03 | 2011-12-07 21:10:03 |
|  8 | Si                          |           4 | 2011-12-26 18:29:14 | 2011-12-26 18:29:14 |
|  9 | No                          |           4 | 2011-12-26 18:29:14 | 2011-12-26 18:29:14 |
| 10 | Si                          |           5 | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |
| 11 | No                          |           5 | 2011-12-26 18:32:46 | 2011-12-26 18:32:46 |
+----+-----------------------------+-------------+---------------------+---------------------+
11 rows in set (0.00 sec)
```

Figura 55: Estado final de la tabla *Posible_respuestas* en la base de datos.



```
mysql> select * from respuestas;
+----+-----------------------------+----------+--------------------+---------------------+---------------------+
| id | valor                       | cantidad | encuesta_pregunta_id | created_at          | updated_at          |
+----+-----------------------------+----------+--------------------+---------------------+---------------------+
|  1 | No                          |        1 |                  9 | 2011-12-26 18:55:54 | 2011-12-26 18:55:54 |
|  2 | Menos de 3 horas            |        1 |                 10 | 2011-12-26 18:55:54 | 2011-12-26 18:55:54 |
|  3 | Ampliar horario de tutor?as. |        1 |                 11 | 2011-12-26 18:55:54 | 2011-12-26 18:55:54 |
|  4 | Ejercicios en clase         |        1 |                  8 | 2011-12-26 18:55:54 | 2011-12-26 18:55:54 |
+----+-----------------------------+----------+--------------------+---------------------+---------------------+
4 rows in set (0.01 sec)
```

Figura 56: Estado final de la tabla *Respuestas* en la base de datos.