UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Diseño e implementación de un sistema de evaluación de riesgo a través de dispositivos móviles

Luis Pérez Pérez
Tutor: Álvaro Ortigosa

Enero 2017

Diseño e implementación de un sistema de evaluación de riesgo a través de dispositivos móviles

AUTOR: Luis Pérez Pérez
TUTOR: Álvaro Ortigosa

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2017

Resumen (castellano)

El objetivo principal de este proyecto es cubrir una necesidad en la detección actual de malos tratos. Los métodos actuales suelen fallar en muchos casos relacionados con la violencia de género porque no existe denuncia o la pareja no tiene ningún tipo de antecedente que pueda prevenir a la policía.

Con esta aplicación intentaremos acercarnos a las usuarias y generar en ellas confianza, para ello utilizaremos una asistente virtual que se dirige al usuario en primera persona y una interfaz amigable, que facilite el uso de la aplicación. El "bot" ofrecerá un trato personalizado e irá guiando a través de la aplicación.

El contenido principal de la app son test creados por psicólogos especializados en el tema de la violencia de género, en ellos se trata de evaluar el nivel de riesgo de la persona.

Para ofrecer una mayor seguridad, intimidad y evitar que la pareja detecte la existencia de la aplicación, ésta ofrecerá métodos de ocultación que serán detallados más adelante.

Al final del test se muestran consejos personalizados dependiendo del resultado obtenido. Se está trabajando para en algunos casos extremos, ofrecer la opción de alertar o informar a la autoridad competente.

Centrándonos en los aspectos técnicos y en la estructura utilizada, el proyecto se basará en una aplicación móvil multiplataforma sincronizada con una aplicación web. Esta última permitirá trabajar con la información de entrada y salida de la aplicación móvil.

El entorno web está orientado a los psicólogos que vayan a trabajar con la aplicación, tendrá una interfaz sencilla y un uso muy intuitivo. Permitirá modificar los test cambiando o añadiendo preguntas, respuestas del bot, e incluso las valoraciones finales. También estarán disponibles filtros para consultar gráficas o descargarse hojas Excel con estadísticas de los últimos datos recogidos.

Ambas aplicaciones están pensadas para que se puedan ir incorporando nuevas herramientas de una forma sencilla. Por parte de la aplicación móvil, podrá ser reutilizada fácilmente como plantilla para generar nuevas apps que contengan otros tipos de test. Por otra parte, la aplicación web estará preparada para generar distintos tipos de test, además del que se utilizará en este proyecto, y sincronizarlos con distintas aplicaciones mediante una palabra clave.

En conclusión, en este documento se explicará cómo se desarrollará el proyecto. En la primera parte se hablará de las motivaciones que llevan a crear estas aplicaciones, después señalaremos los objetivos a cumplir e introduciremos el estudio realizado para la selección de herramientas que acompañarán al desarrollo. En la parte media del documento se profundizará en las herramientas utilizadas, explicando sus características y funcionamiento. Más adelante se expondrán el diseño completo de las dos aplicaciones y todo el desarrollo que se realizará para llevarlas a cabo. Para terminar explicaremos las pruebas que se llevaran a cabo, las conclusiones obtenidas de todo el proyecto y el trabajo futuro a realizar en la aplicación.

Palabras clave

Aplicación Móvil, Aplicación Web, Frameworks, Asistente virtual, evaluación de riesgos, ocultación, sincronización, multiplataforma.

Abstract (English)

The purpose of this project is to cover a necessity about women´s abuse detection. Nowadays, is very difficult that detections tools works efficiently when it does not exist any kind of complain or penal background over the partner.

Deploying this app we attempt to get closer to users using a friendly UI and personalized questions. A bot in a woman's appearance will be the guide of the application. This software uses tests created by psychologist, mostly specialized in gender violence. This tests tries to evaluate risks factors of the woman who uses it.

In order to preserve the users intimacy and prevent their partners to discover it, this app includes a safe way of hiding the information immediately from the user´s device. This methods will be described later in this document.

Every time a test is completed, and taking into account the answer provided, the user will be able to read some advices in screen. We are working for in a close future, in some case of extreme risk, show a button to alert right people.

About project structure, the objective of this work will be develop two applications, one mobile app multi-platform, synchronized with web application. The last one allows to control all input and output information of the first one.

The web application has been designed for psychologists. The main characteristics of this software are usability and simple management of all tests' components like questions, options, answers or the final resolution. Also, they will be able to use filters to get statistics and charts with data from all results received.

Both applications were designed to make easy to add improvements if needed. The mobile app can be used like template for make applications with other tests categories. By other way, the web app is ready to create and synchronize new kinds of tests and link them with another mobile app through a keyword.

In conclusion, this document contains all the information relating with the software implement in this project. The first part explains the motivations, goals and the technological research carried out in order to find the best develop tools. The middle part is focused on the tools selected, describing their main characteristics and how they works, also contains the design of the two applications. Finally, in the last part of the project, goes deeper in develop process, testing software, conclusion and future work that may be done in the app.

Keywords

Mobile app, web app, Framework, Bot, evaluate risks, multi-platform, safe way hiding, synchronization.

Agradecimientos

En especial este trabajo se lo quiero dedicar a mi familia, a mis incansables y sufridos padres, simplemente por todo. A mis hermanas, abuelos y primos, por apoyarme con todo su cariño y aconsejarme lo mejor que saben, muy especialmente a uno de ellos por el que me hubiera gustado acabar esto antes. A Paula, por cuidarme, por estar siempre a mi lado y hacerme sentir tan especial. Y por supuesto, a todos mis amigos.

Además, también quiero agradecer a la gente que ha hecho posible este proyecto, a Alejandro con su idea original “ChatBot” junto con sus compañeras que ayudaron a desarrollar los test, a Álvaro y a Carlota por sus consejos para el proyecto y a Seti, empresa donde conseguí los conocimientos que me han permitido llevar a cabo el desarrollo de las aplicaciones.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Actualmente existe una carencia en la mayoría de herramientas de detección y prevención de violencia de género, y es en aquellos casos difíciles de detectar donde más hincapié se quiere realizar actualmente para reducir al máximo el número de víctimas que sufren este tipo de violencia. Estos casos generalmente engloban a las mujeres que no son consciente de que sufren malos tratos, o sí lo son, pero no se atreven a denunciar y sus parejas no tienen ningún tipo de antecedentes de este tipo que facilitaría la detección de estos casos.

En algunos casos, es muy complicado que la víctima sea capaz de dejar a su pareja, o que tenga el valor para acudir a la policía si fuera necesario, o incluso cuando la víctima ni si quiera es consciente de los abusos que sufre. Con esta aplicación nos queremos acercar al máximo a este tipo de personas y ofrecerles la mayor ayuda posible para que puedan resolver su problema.

Por otro lado, existe demanda por parte de equipos de psicólogos para poder acercarse a cualquier tipo de individuo y ofrecerles cuestionarios sobre cualquier tema en el que les puedan ayudar, como por ejemplo en los ámbitos educativos, deportivos, de la salud, etc. Además del problema del acercamiento a todo tipo de público, también existen muchas complicaciones a la hora de recoger resultados para poder mejorar los cuestionarios y ofrecer mejores ayudas.

Dos vías principales nos llevan a querer desarrollar un proyecto tan ambicioso con tan pocos recursos:

- Sociedad. La facilidad que existe en la actualidad para acercarnos a todo tipo de público utilizando elementos ya existentes dentro de la población actual. Esto se debe al claro crecimiento del acceso a internet, el alto porcentaje de personas que dispone de dispositivos móviles inteligentes y el alejamiento del miedo a expresar lo que pensamos y de la estigmatización a aquel que busca recibir ayuda.

- Tecnología. El imparable crecimiento actual de la cultura de compartir todo tipo de recursos que facilitan el crecimiento del conocimiento del individuo y aumentan la capacidad de desarrollo de tecnologías a niveles impensables hace pocos años. Dentro de este ámbito denominado como “código libre” o “aplicaciones de código libre”, nos podemos encontrar con gran variedad de herramientas de programación, lenguajes, entornos de desarrollo, comunidades de programadores, tutoriales, librerías que proporciona acceso a funcionalidades de todo tipo en cualquier plataforma, y muchos más recursos libres a los que puede acceder cualquier persona.

Debido a la necesidad principal de acercarnos a la solución de problemas tan graves como la violencia de género o similares a cualquier otro tipo de violencia, adicción, trastorno, o simplemente problemas más cotidianos. La creación de este proyecto, proporcionará una herramienta de trabajo, para poder detectar y ofrecer algún tipo de ayuda.

1.2 Objetivos

La finalidad del proyecto es dejar dos aplicaciones completamente funcionales que estén centradas en el tema principal del TFG.

a) Una aplicación móvil, que ofrezca cuestionarios de evaluación de riesgos de violencia de género y además facilite consejos o métodos de ayuda a las usuarias.

b) Otra aplicación, ésta desarrollada para estar alojada en un servidor y que ofrezca herramientas para los expertos en la materia, que les permita la gestión de toda la información contenida en los test y consulta de los resultados obtenidos con las respuestas de los usuarios.

Un objetivo que se pretende alcanzar con el desarrollo inicial, es que se levante interés suficiente para poder seguir desarrollando la aplicación después de la finalización del proyecto. Esto implica que la herramienta tiene que estar preparada para crecer con facilidad y poder añadirle fácilmente nuevas funcionalidades.

Como objetivos secundarios, este proyecto pretende mostrar que con la tecnología actualmente disponible es relativamente sencillo desarrollar herramientas que ayuden resolver problemas que no deberían existir en la sociedad actual. Así, los dos objetivos secundarios los podemos enunciar de la siguiente forma:

a) Actualmente con muy poca inversión económica y con la colaboración entre diferentes profesionales se pueden desarrollar herramientas que faciliten la vida de las personas, llegando a ser posible introducir utilidades dentro de las aplicaciones que pudieran proporcionar cierta ayuda al usuario.

b) Dar a conocer las nuevas tecnologías que facilitan el desarrollo de aplicaciones en cualquier plataforma y cómo se puede establecer comunicación entre ellas de formas relativamente sencillas.

El plan de desarrollo de este TFG es desglosar los objetivos globales planteados de la siguiente forma:

1.- Establecer contacto con el equipo de psicólogos que poseen los test con los que trabajaremos a lo largo de la aplicación.

2.- Definir todas funcionalidades de las que queremos que disponga el proyecto.

3.- Sistemas que necesitamos desarrollar para cumplir con todas las funcionalidades exigidas.

4.- Número de usuarios a los que queremos llegar.

5.- Calcular el coste total económico del desarrollo de la aplicación ya que no queremos que la aplicación se quede solo en un proyecto. Aun teniendo en cuenta que toda la parte inicial será desarrollada a coste cero. Es muy importante ser consciente del tiempo que dispondremos, ya que el objetivo final implica la entrega

de una aplicación completamente operativa cuyas funcionalidades finales se describen más adelante.

6.- Decidir que lenguajes se adaptan mejor a nuestras necesidades.

7.- Preparar unos prototipos para mostrar a los interesados en la aplicación. Ellos son los que decidirán si se pasa al siguiente paso en el proyecto o se tienen que retocar algunos detalles antes de comenzar con el desarrollo.

8.- Desarrollar cada una de las aplicaciones, mostrando los avances a los psicólogos para que puedan ir dando el visto bueno y aportando nuevas ideas.

9.- Realizar las pruebas necesarias al software final.

10.- Desplegar las aplicaciones en cada plataforma. En caso de la aplicación web, la instanciaremos sobre un servidor Apache de la **Universidad Autónoma** al que se nos ha dado acceso para el proyecto. La distribución de la aplicación móvil, será gestionada por aquel al que se le entregue la aplicación.

Completando esta enumeración de objetivos, se intenta conseguir llegar al objetivo final principal.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1 - Introducción:**

  Motivaciones y enumeración de objetivos.

- **Capítulo 2 - Estado del Arte:**

  Estudio de herramientas y selección final con descripción detallada.

- **Capítulo 3 - Diseño:**

  Diseño de aplicaciones desarrolladas.

- **Capítulo 4 - Desarrollo:**

  Proceso de desarrollo con muestras de código y capturas de las apps.

- **Capítulo 5 - Conclusiones y trabajo futuro:**

  Conclusiones finales del proyecto y planificación de trabajo futuro.

2 Estado del arte

2.1 FRAMEWORKS

Desde un punto de vista objetivo, creo que son los recursos o las herramientas más potentes que ofrece actualmente la programación.

"Frameworks", una palabra muy común a día de hoy en el mundo de la informática, que algunas veces, se utiliza sin saber cuál es exactamente su significado.

Si estamos hablando de programación, a lo que nos estamos refiriendo exactamente es: *"estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto."* **[1]**

De una forma más coloquial se habla de un conjunto de módulos de funciones de bajo nivel, que no tienen que estar necesariamente en un único leguaje y que proveen de funciones de más alto nivel al programador, ahorrando mucho tiempo de programación y de diseño.

Una de las grandes virtudes de los frameworks y que hacen que sean una realidad más que palpables a día de hoy, son las comunidades de usuarios. Gracias a esto es muy sencillo resolver cualquier duda, conocer cuál es la mejor manera de escribir un algoritmo para exprimir al máximo la potencia del framework y hasta la opción de obtener librerías creadas por otros desarrolladores, generalmente muy estables ya que han sido testeadas por miles de usuarios.

2.1.1 Introducción

Existen miles de frameworks, tanto open sources como de pago, disponibles para prácticamente cualquier plataforma, web, móvil, pc. En esta subsección, enumeraremos algunos de los que fueron planteados para el desarrollo de la aplicación. Se han clasificado por plataforma.

2.1.1.1 Estudio herramienta para Smartphone

2.1.1.1.1 Unity

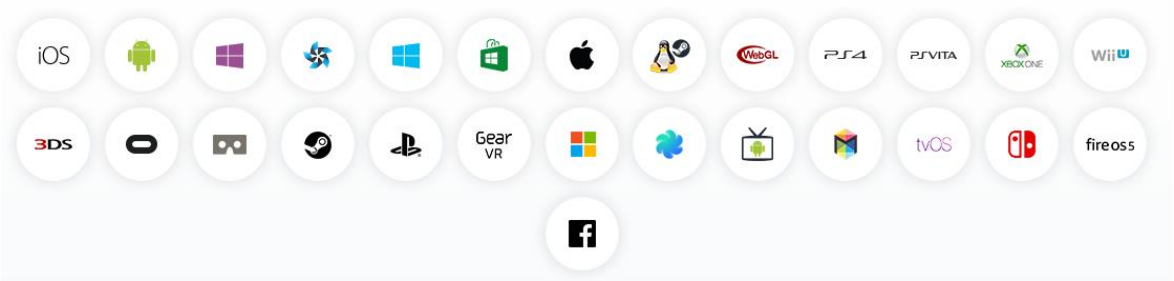

Figura 2-1: Plataformas soportadas por Unity. Fuente: https://unity3d.com/es/unity

Todas las plataformas que aparecen en la imagen son las plataformas soportadas por Unity. Aunque el reconocimiento de este framework viene dado principalmente gracias al desarrollo de videojuegos.

La capacidad de Unity para adaptarse tanto a IOS, Android o Windows Phone hace que sea una opción muy importante a plantearse a la hora de desarrollar una aplicación móvil, ya que la parte gráfica se compartirá con todas las plataformas y la mayor parte de los scripts. Unity utiliza C# para scripts, que combinando con la potencia del motor gráfico, permite desarrollar aplicaciones con gran calidad gráfica y generar animaciones de una manera mucho más sencilla que con cualquier otro framework.

También ofrece una licencia personal gratuita que permite el desarrollo comercial, siempre y cuando la capacidad de ingresos o los fondos recaudados no superen los 100.000$ por ejercicio fiscal. **[2]**

Por supuesto no son todo ventajas, la curva de aprendizaje para el manejo de la herramienta de diseño gráfico es bastante lenta. Además las aplicaciones generadas son bastante pesadas debido a la complejidad del motor gráfico, por lo tanto no es una opción muy aconsejable para desarrollos de aplicaciones sencillas.

2.1.1.1.2 Titanium (appacelerator titanium)

Figura 2-2: Datos Titanium tomados 10/01/2017 en. Fuente http://www.appcelerator.com/

Este framework apareció en 2008 pensado en las aplicaciones Cross-Plataform y desde su salida no dejo de crecer hasta el punto que en 2012 aplicaciones desarrolladas con Titanium se encontraban en el 10% de los smartphones de todo el mundo. **[3]**

Titanium utiliza JavaScript para el desarrollo de toda la aplicación. A la hora de generar la compilación se genera un contenedor web en su versión nativa, según la plataforma sobre la que se quiera ejecuta (IOS, Android, Windows Phone).

Actualmente se sigue utilizando para el desarrollo de aplicaciones en muchas empresas, pero el auge de nuevos frameworks gratuitos con mayor capacidad Cross-Plataform, ha hecho que se vaya debilitando lentamente.

Los principales inconvenientes de este framework, actualmente solo tiene una versión de prueba durante un mes en con perfil Indie, después para poder seguir desarrollando hay que abonar una cuota mensual. Adicionalmente, la comunidad abierta que posee es muy escasa, ya que casi todo el soporte necesario lo ofrece la propia empresa que lo distribuye.

2.1.1.1.3 Xamarin

Se trata de una plataforma que principalmente se define por su librería Mono, esta librería permite la traducción de código programado en C# a código Java nativo.

Inicialmente disponía de su propio entorno de desarrollo "Xamarin Studio". Pero desde que Microsoft adquirió la empresa, también se puede desarrollar desde Visual Studio Community.

También permite la programación directa en Objetive-C, Swift, o Java, por lo tanto no deja atrás a las personas que no tiene muchos conocimientos en C# o que simplemente quieran programar en código nativo y aprovecharse de las librerías que ofrece.

Una gran ventaja es que la licencia es gratuita para estudiantes o equipos pequeños de desarrollo. Sus principales puntos débiles son que es un framework muy reciente que todavía se está intentando consolidar y hacerse un hueco entre los grandes.

2.1.1.2 Estudio de herramientas de desarrollo Web

A continuación se presenta una comparativa de los softwares que vamos a proponer.

COMPARTIVA FRAMEWORKS WEB	SYMFONY 2	LARAVEL	FUELPHP
VERSIÓN PHP	5.5.9	5.5.9	5.3.3
DATOS SOPORTADOS TIPOS DE BASES	MySQL Postgress SQLite Qracle	MySQL Postgress SQLite SQL Server	MySQL SQLite Postgress
ALMACENAMIENTO EN LA NUBE	Amazon S3 (plug-in)	Amazon S3 Rackspace	Amazon S3 (plug-in)
DOCUMENTACIÓN EN EL SITIO OFICIAL	Tutoriales, referencia de la API	Tutoriales, referencia de la API, lecciones de vídeo	Sólo tutoriales
AUTOMATIZADO EXTENSIÓN DE LA INSTALACIÓN (A TRAVÉS COMPOSITOR)	Sí	Sí	No

Tabla 1: Especificaciones Frameworks. Fuente: https://kultprosvet.net/blog/10-best-php-frameworks-web-projects**.**

La siguiente imagen nos muestra una gráfica sobre los post realizados en StackOverflow con referencia a relacionados con Laravel y Symfony, no existían datos registrados sobre FuelPPHP.

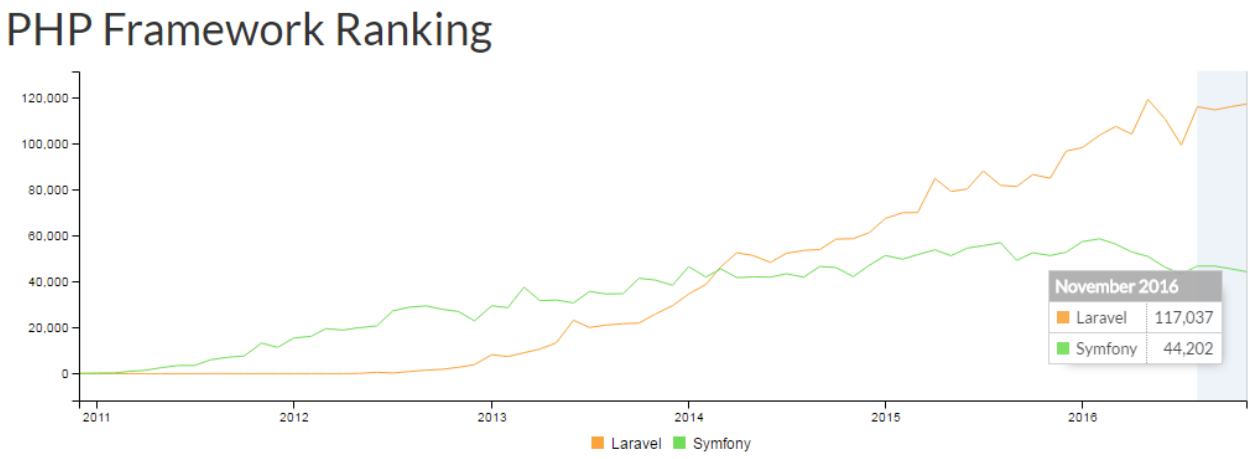

Figura 2-3: Evolución de posts publicados StackOverFlow sobre cada framework. Fuente: http://www.stackoverkill.com/ranking/php

2.1.1.2.1 Symfony2

Sin ninguna duda uno de los frameworks web open source más potentes del mercado actual, basado en PHP combinado con HTML y CSS, además permite la utilización de la herramienta Twig, un motor de plantillas para PHP.

Symfony es una herramienta muy compleja que requiere un alto grado de especialización para poder sacarle el máximo partido. Por otro lado, con unas nociones básicas ya es posible desarrollar aplicaciones web simples, siendo necesarios buenos conocimientos en PHP.

La potencia de esta herramienta permite crear una aplicación de inicio a fin y personalizada completamente. Por su complejidad y su buena optimización, está principalmente pensada para empresas que quiera realizar grandes desarrollos. Permite conectividad de miles de usuarios simultáneamente sin perder potencia (considerando que se despliega en un servidor acorde con las características requeridas por la aplicación desarrollada).

La herramienta utiliza la el gestor de paquetes Composer, que permite la instalación y actualización relativamente sencilla, tanto de Symfony como de sus módulos o librerías adicionales.

Utiliza Doctrine 2 como herramienta ORM, una librería escrita en PHP para el manejo de bases de datos, es considerada el ORM más utilizado actualmente. Sus principales características son la capacidad para hacer cualquier tipo de consultas a través de sus funciones, la robustez y la seguridad que ofrece en el manejo de datos.

Este framework fue descartado para este proyecto debido a la alta complejidad que presenta, ya que para el uso que se le va a dar la aplicación no es necesario un desarrollo tan potente.

2.1.1.2.2 Laravel

Otro framework de PHP, especialmente pensado para crear proyectos con una estructura claramente definida y fácilmente comprensible. Los principales lemas de los que presumen sus creadores para convencernos para utilizar su software son “Love beautiful code? We do too.” y “Did someone say rapid?”**[4]** haciendo clara referencia a la limpieza y estilo de su código y la rapidez de su software.

Cabe destacar que se trata de un software con las dependencias de Symfony, por lo tanto solo puede crecer de la mano de este último. A pesar de esto, es sin duda de los frameworks que más ha crecido en este último año, este crecimiento seguramente sea debido a la seguridad que ofrecen las dependencias de Symfony y a la mayor simpleza de su utilización. La curva de aprendizaje de esta herramienta es bastante rápida.

Posee una gran comunidad y mucha actividad en los foros como muestra la figura 2-3. Al tratarse de código abierto, una gran cantidad de librerías que comparten los desarrolladores independientes.

Como ORM utiliza Eloquent, un manejador personalizado, que como principal característica, debido a la modularidad de Laravel, es posible utilizar este módulo en cualquier proyecto, independientemente de que utilice el framework.

Destacar que este framework también utiliza Composer para poder mantener un control de versiones de los paquetes instalados.

2.1.1.2.3 FuelPHP

Seguramente el más pequeño, menos conocido y con menos potencia de los tres, pero no por esto hay que menospreciarlo. El framework sigue en crecimiento y se espera la llegada de la versión 2.0 para principio de 2018, la principal novedad será la compatibilidad con PHP7.

Se trata de un framework que puede utilizar prácticamente todas las librerías que utiliza Laravel y Symfony, con una instalación de estos módulos sencilla, a través de Composer o descargándolos de cualquier repositorio. FuelPHP consta de una comunidad bastante grande y activa, pero no comparable con las dos anteriores.

La principal característica de este framework es la simpleza, su flexibilidad, con una estructura del código muy bien definida muy similar a la de Laravel, la combinación de un modelo Vista-Controlador basado en vistas en Twig con controladores en PHP se suma a la potencia de JavaScript con JQuery. Por lo tanto, teniendo conocimientos básicos de programación web, la curva de aprendizaje experimentada con FuelPHP es muy rápida.

En cuanto a su ORM, FuelPHP se distingue por tener un manejador para la base de datos propio, más sencillo que el resto, muy simple de utilizar, menos potente que los otros, pero fiable y más que suficiente si se busca una aplicación de tamaño medio. Dónde más se pueden notar las carencias del módulo es al querer realizar grandes consultas anidadas. Este escollo lo salva con la librería DB, ésta ofrece un trato menos amigable con los datos extraídos de la base de datos, pero posibilita la ejecución de cualquier consulta directamente sobre la base de datos.

Las flaquezas de FuelPHP aparecen cuando se intentan llevar a cabo el manejo de gran cantidad de datos, ya que se queda atrás a la hora de compararla con Symfony, pero como ya hemos mencionado anteriormente, tiene el potencial para desarrollar la mayoría de aplicaciones de tamaño medio. También el tamaño de la comunidad, comparado con los otros dos mencionados, es una desventaja.

2.1.2 Motivos de selección de Framework

El desarrollo de la aplicación móvil se realizará con Xamarin trabajando en el entorno Visual Studio Community. El motivo es la alta reusabilidad de código gracias a la programación de lenguaje C# en las tres plataformas más importantes, además de la capacidad de poder utilizar las librerías .Net.

El apartado web se desarrollará en FuelPHP en el entorno PhpStorm. Los motivos que llevan a esta elección son su sencillez, que sus características técnicas se adaptan perfectamente a las necesidades del proyecto y por último, al realizarse el desarrollo por una sola persona, la facilidad para aprender el manejo del framework.

2.2 HERRAMIENTAS DE PROGRAMACION A UTILIZAR

2.2.1 Xamarin

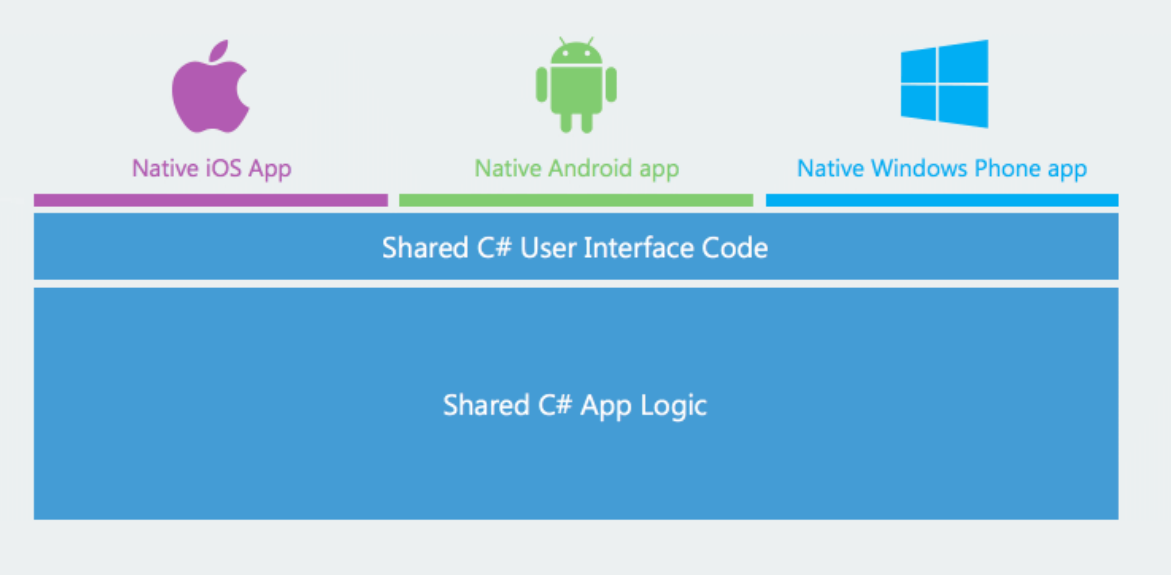

Figura 2-4: Estructura Xamarin. Fuente: https://www.xamarin.com/platform**.**

Antes de comenzar a utilizar Xamarin es necesario decidir qué tipo de proyecto vamos desarrollar. Existen tres opciones principales, aplicación exclusiva para Android, aplicación para IOS o aplicación Cross-Plataform. Con esta última la aplicación se podrá ejecutar en Android, IOS y Windows Phone.

2.2.1.1 Xamarin.Android

El núcleo de Xamarin.Android posee un motor que puentea C# con Java permitiendo a los desarrolladores tener acceso a la API de Java. Esta característica permite crear aplicaciones nativas programadas con C# con .Net.

Un pequeño ejemplo de la traducción que realiza el software se muestra en la siguiente tabla. En ella se puede ver un mapeo de cómo se conectan entre si las siguiente interfaces.

Collection mappings

Java Type	System Type	Helper Class
java.util.Set<E>	ICollection<T>	Android.Runtime.JavaSet<T>
java.util.List<E>	IList<T>	Android.Runtime.JavaList<T>
java.util.Map<K,V>	IDictionary<TKey,TValue>	Android.Runtime.JavaDictionary<K,V>
java.util.Collection<E>	ICollection<T>	Android.Runtime.JavaCollection<T>

Tabla 2: Mapeo de tipos Java-Xamarin. Fuente: https://developer.xamarin.com/guides/android/advanced_topics/api_design/.

La principal ventaja de programar en C# es que el código se puede reutilizar la parte lógica para el código de IOS.

2.2.1.2 Xamarin.IOS

Las librerías Xamarin.IOS y Xamarin.Mac pueden llamar a las librerías de Objetive-C a través de código escrito en C#. Esta conexión hace mucho más sencilla la programación para los productos de Apple, que normalmente solo son programables en Objetive-C o Swift.

2.2.1.3 Xamarin.Forms

Esta opción es una herramienta de diseño de interfaces (UI ToolKit) multiplataforma. Permite a los desarrolladores crear pantallas para los sistemas iOS, Android, Windows Phone, Windows Store y Universal Windows con un único diseñador.

El inconveniente que tiene esta herramienta es que es un tanto limitada, se queda muy lejos de la calidad y la versatilidad de las herramientas de diseño nativas.

2.2.1.4 Packages

Tanto Visual Studio como Xamarin Studio, incluyen NuGet, uno de los más populares gestores de paquetes para .Net.

Este gestor permite buscar y descargarse la librería que necesites, además de instalarla automáticamente, también permite seleccionar que versión del paquete deseas utilizar, una gran ventaja para poder desarrollar seguro de la compatibilidad.

2.2.2 FuelPHP

Comenzar un proyecto en FuelPHP es muy sencillo, simplemente descargar de la página oficial la última versión. El contenido descargado será directamente una plantilla donde desarrollar el proyecto.

2.2.2.1 Vistas

FuelPHP trae incluido de forma predefinida los drivers para utilizar cualquiera de las siguientes plantillas para html5:

- Mustache, Markdown, Smarty, Twig, Haml, Jade, Dwoo y Phptal.

Estos templates en los sistemas MCV son muy útiles al permitir manejar mejor las variables en las vistas. Además permiten el uso de extensiones que puedes personalizar programándolas directamente en PHP.

2.2.2.2 Controladores

Los controladores se programan en PHP utilizando la estructura y las librerías del framework. Se incluyen tres clases base para este tipo de archivo:

Controller-Template: Se utiliza para controladores de páginas simples.

Controller-Rest: Permite la creación de APIs con facilidad.

Controller-Hybrid: Combina las dos características anteriores en un solo controlador.

A continuación se expone el código de un fichero ejemplo que viene contenido en la propia plantillada de la versión 1.8. En él podemos observar la simple estructura utilizada para escribir un controlador. Primero se define el nombre de la clase, en este caso "Controller_Welcome", ésta contiene funciones que devuelven las vistas del controlador, estos métodos serán llamados a través de una dirección definida por el fichero routes.php.

Para obtener más información sobre el fichero routes.php, ver sección 3.3.1 punto 2. Para consultar el código de los controladores generados en el proyecto, ver sección 4.1.2.

```php
/**
 * Fuel is a fast, lightweight, community driven PHP5 framework.
 *
 * @package    Fuel
 * @version    1.8
 * @author     Fuel Development Team
 * @license    MIT License
 * @copyright  2010 - 2016 Fuel Development Team
 * @link       http://fuelphp.com
 */

/**
 * The Welcome Controller.
 *
 * A basic controller example.  Has examples of how to set the
 * response body and status.
 *
 * @package  app
 * @extends  Controller
 */
class Controller_Welcome extends Controller
{
    /**
     * The basic welcome message
     * @access  public
     * @return  Response
     */
    public function action_index()
    {
        return Response::forge(View::forge('welcome/index'));
    }

    /**
     * A typical "Hello, Bob!" type example.  This uses a Presenter to
     * show how to use them.
     *
     * @access  public
     * @return  Response
     */
    public function action_hello()
    {
        return Response::forge(Presenter::forge('welcome/hello'));
    }

    /**
     * The 404 action for the application.
     *
     * @access  public
     * @return  Response
     */
    public function action_404()
    {
        return Response::forge(Presenter::forge('welcome/404'), 404);
    }
}
```

Figura 2-5: ejemplo controlador. Fuente: FuelPHP V1.8 fuelphp-1.8\fuel\app\classes\controller\welcome.php)

2.2.2.3 Packages

Los paquetes sirven para mejorar la organización del código, además facilitan la reutilización y el poder compartirlos.

De forma predefinida se incluyen cinco paquetes, estos están preparados para realizar tareas con llamadas a sus clases y métodos, acelerando así la producción del proyecto.

A continuación explicamos brevemente cada uno de estos paquetes:

- Auth: Proporciona un sistema de logueo en la aplicación, dando control total sobre los usuarios de la aplicación. Incluye los modelos de la base de datos a utilizar con la clase.
  Documentación http://docs.fuelphp.com/packages/auth/intro.html

- Email: Permite el envío de correos electrónico. Con unos simples pasos podremos enviar cualquier tipo de email a cualquier dirección.
  Documentación http://fuelphp.com/docs/packages/email/intro.html

- Oil: Este paquete realiza la ejecución de tareas a través de una clase llamada Task. La cualidad especial que proporciona es el poder realizar tareas en el servidor sin necesidad de hacer peticiones web.
  Documentación http://fuelphp.com/docs/packages/oil/intro.html

- Orm: Paquete que facilita la gestión de la base de datos mediante el uso de modelos que se instanciarán como clases para poder manejar los registros como objetos PHP.
  Documentación http://fuelphp.com/docs/packages/orm/intro.html

- Parser: Este paquete es el encargado de traducir las plantillas utilizadas para las vistas. Al inicio de proyecto hay que completar el fichero de configuración definiendo que tipo de plantillas usaremos.
  Documentación http://fuelphp.com/docs/packages/parser /intro.html.

2.2.2.4 Bootstrap

FuelPHP incluye Bootstrap, un framework “front-end”, que nos permite el diseño de pantallas responsives. Además trae una hoja de estilos muy completa para la creación de las vistas de cualquier web.

2.3 BASES DE DATOS

2.3.1 Mysql

Para el almacenamiento de datos de nuestra aplicación web, se creará una base de datos relacional sobre MySQL en su versión 5.6.

Gracias a la base de datos podremos guardar todos los datos relacionados con los test y sus resultados obtenidos de cada uno de los dispositivos donde se utilice la aplicación móvil.

2.3.2 SQLite

Es un gestor de bases de datos relacionales de un tamaño relativamente pequeño ~275kb **[5]**. Es un sistema compatible con Xamarin que utilzaremos para gestionar la base de datos de la aplicación móvil.

A pesar de ser un gestor relativamente limitado, se utilizará para la aplicación móvil ya que cumple nuestros requisitos y queremos realizar una aplicación lo más ligera posible.

2.4 REPOSITORIOS

2.4.1 Definición y Repositorio de código utilizado

Un repositorio es un lugar pensado principalmente para almacenar y llevar un registro de las modificaciones de datos, en nuestro caso, el repositorio lo formarán los ficheros generados por el proyecto.

Generalmente se encuentran alojado en unidades de red o servidores, para permitir el sincronizar el trabajo individual de cada miembro de un equipo. Se utiliza para realizar control de versiones.

Uno de los software más conocidos actualmente es Git, un controlador de versiones diseñado inicialmente para Linux que termite trabajar en diferentes ramas, combinar información entre ellas y detallar en cada actualización de las ramas los cambios realizados.

Para este proyecto se utilizará Bitbucket, que permite la creación gratuita de repositorios privados y soporta Git.

2.4.2 Repositorio Aplicación Móvil

La aplicación móvil se sincronizará con un repositorio, ya que esto permitirá además de llevar un control de las mejoras que se van incorporando al código, generar, si fuera necesario, diferentes ramas con cada parte del código para las distintas plataformas.

Adicionalmente tener la aplicación sincronizada en red permite el acceso al código de la aplicación desde cualquier ubicación.

2.4.3 Repositorio Aplicación Web

Para la aplicación Web también se creará otro repositorio. En este caso la utilidad será mucho mayor, ya que además de las funcionalidades explicadas anteriormente, en un entorno web, permite mantener sincronizado el código desarrollado en un entorno local con la aplicación desplegada en un servidor, independientemente de las ubicaciones.

Además la utilización de ramas permite tener distintas versiones en el entorno de desarrollo y en el de producción.

3 Diseño

3.1 Aplicación Web

En esta sección se detallara con la mayor precisión y simplicidad posible las características del diseño de la aplicación.

3.1.1 Descripción de la aplicación

Como ya se ha explicado anteriormente. La función principal de esta aplicación será la gestión de los test y la recogida y procesamiento de los datos enviados por los usuarios de la aplicación móvil.

El entorno web tiene que tener un sistema de logueo que proporcione seguridad a la aplicación. Se piensa en un sistema inicial simple, con proyección a un sistema de roles que permita control total sobre cada usuario que entre en la aplicación.

Una vez dentro de la aplicación, se presentará una pantalla de bienvenida donde se podrá consultar gráficas con las estadísticas de la aplicación. En la parte izquierda un menú ocultarle que permita navegar por la aplicación.

La aplicación nos tiene que permitir realizar la gestión completa de los test, es necesario crear un sistema CRUD de los siguientes elementos:

- Maestras: Estas serán las clases maestras de la aplicación. Elementos necesarios para el funcionamiento de la aplicación. Son registros únicos que se puede utilizar para realizar acciones o para definir propiedades de otros elementos de la aplicación.

- Test: Los test se podrán crear, modificar, eliminar. Principalmente se identificarán por dos valores, el tipo y el nombre. Adicionalmente tendrán dos valores más, descripción y una celda que determina si es el test activo de los tipos. Para poder trabajar con los test es necesario crear las herramientas para modificar los elementos que dependen de él.

- Preguntas: Cada test estará compuesto por un número variable de preguntas, éstas contendrán las siguientes propiedades: texto asociado a la pregunta, un valor para ponderarla y N opciones disponibles de respuesta.

- Opciones: Cada opción estará relacionada únicamente con una sola pregunta. Para construirlas se guardará un texto descriptor de la opción y un número que especifique el valor.

- Respuestas: El bot estará preparado para mostrar distintas respuestas dependiendo de la opción seleccionada y del estado actual del test. Para lograrlo, cada opción contiene tres posibles respuestas clasificadas por categoría: baja, normal y alta. Dependiendo de las respuestas que se vayan dando el test se dará una respuesta u otra.

- Resultados: Como ya se ha contado anteriormente, al final del test se ofrecerán los resultados obtenidos y unos consejos de ayudas personalizado para cada caso. Para esto tendremos que generar un módulo donde se permita añadir un número indeterminado de posibles resultados finales. Para cada uno de ellos habrá que determinar el número superior e inferior del intervalo al cual pertenecerá y el texto para con el consejo para cada caso.

3.1.2 Utilidades implementadas

3.1.2.1 Módulo Auth

Se utilizará el modulo perteneciente a FuelPHP, para crear todo el sistema de sesión de usuarios de la aplicación.

Para ello generaremos las tablas en nuestra base de datos utilizando las definiciones del módulo "auth" y las conectaremos con el ORM a través de los modelos predefinidos.

Introduciremos dentro de la categoría "maestras" que se encuentra en el menú principal, una entrada llamada "usuarios". Entrando en ella nos encontraremos una vista con una tabla que contendrá los usuarios actuales. En esta vista se nos permitirá crear nuevos usuarios, editar o borrar los ya existentes. El usuario principal no se podrá eliminar.

Del usuario guardaremos:

- Nombre: Servirá para identificar al usuario.
- Apellidos: Se utilizará para mostrar junto con el nombre siempre que nos queramos referir al usuario actual de la herramienta.
- Email: En un futuro se podría utilizar para emails con estadísticas o similares.
- Grupo: Actualmente solo existe el de "Súper Admin", que da acceso total a la aplicación. En un futuro se podrían añadir nuevos para definir las restricciones y privilegios a los usuarios.
- Usuario: Servirá como usuario para entrar en la aplicación.
- Password: Clave de entrada en la aplicación. Se almacenará en la base de datos cifrada con clave asimétrica. Por lo tanto no se podrá obtener la contraseña en texto de ninguna forma.

3.1.2.2 Creador de Test

Se diseñará un controlador que permita generar un test completo. Este proceso implicará varias pantallas ya que habrá que generar varios registros distintos.

Test:

Para acceder a los test, habrá una entrada en el menú principal nombrada como "Test". Al entrar en la sección se nos mostrará un selector donde se indicará el tipo de elemento test que se quiere consultar/crear.

Una vez elegido, se pasará a una pantalla de edición dónde aparecerán todos los test pertenecientes a este tipo y los datos del seleccionado. El test mostrado aparecerá sombreado y clicando en cualquier otro se accederá a sus datos para poder editarlos.

En el cuadro de edición se mostrará el nombre, la descripción y habrá una celda para convertir este test en el activo dentro de su grupo, si el test ya lo fuera, habría un texto especificándolo.

En la parte inferior habrá una tabla con todas las preguntas relacionadas, cada una de ellas con su pareja de botones asociados para editar y eliminar. En la parte superior de la tabla habrá un botón para añadir nuevas preguntas.

Preguntas:

Si clicáramos sobre el botón de edición o sobre el de nueva pregunta, entraríamos en otra vista. Si fuera una nueva, nos encontraríamos la pantalla vacía con dos celdas para completar con el texto y el valor de la pregunta. Una vez guardado los primeros datos, aparecerá un nuevo contenedor en la parte inferior para añadir las opciones.

Como máximo se permitirán cuatro opciones por preguntas. Para añadirlas, se colocará un botón en la parte superior, que de forma dinámica hará aparecer un cuadro para introducir la nueva opción. Las opciones se salvarán en el mismo botón que las preguntas, éste se encontrará situado en la parte superior de la pantalla nombrado como "Guardar".

Opciones:

En el cuadro de edición de las opciones, aparecerá una entrada seleccionable con los textos disponibles y otra entrada para introducir un valor. Por otro lado, tendrá que haber un botón de acceso directo a la edición de maestras de los textos de opciones, para facilitar añadir nuevos textos en el selector. El valor servirá para pesar la respuesta, después se utilizará en el algoritmo que genera el resultado.

Después de estos valores nos encontraremos otra zona donde las respuestas del bot están asociadas a las opciones. Habrá tres cajas por cada opción para registrar los textos, en cada una de ellas vendrá indicada la categoría (baja, alta, media).

3.1.2.3 Maestras

El acceso a esta utilidad se hará seleccionando una subcategoría dentro “Maestras” en el menú principal.

Existirán cuatro tipos de valores maestros. Estos servirán para configurar la aplicación, el formato busca la unificar y evitar la duplicidad de la información que se almacenará en la base de datos, así posteriormente facilitar la clasificación.

Clases maestras que se implementarán:

Usuarios: Esta clase se utilizará para generar los usuarios de la plataforma. Explicada con más detalle en la sección 3.1.2.1.

Test: Pantalla donde se mostrará una tabla de los test implementados. En la tabla se mostrarán los principales valores y como en los demás casos, los botones correspondientes para realizar las acciones: crear, editar, borrar.

Tipos de test: Aquí podremos generar nuevos tipos de test, esto servirá para registrar nuevas aplicaciones móviles asociadas al servidor. Será un CRUD muy simple, ya que solo tendrá un valor, un nombre que describa los tipos de test que se quieren generar.

Texto para preguntas: Textos bases para completar las opciones. Estos textos se han pensado como una clase maestra ya que nos vamos a encontrar que casi todas las mismas preguntas tienen los mismos textos en las opciones: “Sí”, “No”, “A veces”… Por lo tanto de esta forma evitamos tener información repetida en la base datos.

3.1.2.4 Sincronización

La aplicación tendrá la capacidad de sincronizarse con todos los dispositivos en los que se descargue. Realizará dos tipos de sincronizaciones, de entrada y de salida. Ambas se harán a través de cadenas de texto en formato JSON.

El algoritmo de salida, montará toda la información del test activo sobre una cadena de texto, como índice de cada valor pondremos el nombre del registro con el que se identifica en la base de datos. Una vez creada la cadena se enviará como respuesta a una petición hecha por la aplicación móvil. El dispositivo será el que se encargue de decodificar la información y almacenarla.

El algoritmo de entrada, deserializará la cadena de texto y almacenará la información recibida. Pueden llegar dos tipos distintos, información sobre el usuario o los resultados del test. Este tendrá la capacidad de diferenciarla y guardarlas en los registros oportunos.

3.1.2.5 Graficas

De los datos que se vayan recibiendo se irán calculando estadísticas, para mostrarlos de una forma útil y visual utilizaremos gráficas. Para generar las gráficas nos apoyaremos en JavaScript y recurriremos a librerías para pintar y modificar valores.

Como funciones especiales, habrá filtros que permitan seleccionar la información que se quiere mostrar adicionalmente. También se colocará un botón que permita descargar estos gráficos en formato “CSV” o si así se requiere, una hoja Excel.

3.1.3 Base de datos

Como ya describimos anteriormente, el servidor tendrá una base de datos manejada por MySQL Server 5.6. Será una base de datos bastante más grande que la de la aplicación móvil y con todas las tablas relacionadas entre sí.

3.1.3.1 Modelo Entidad-Relación

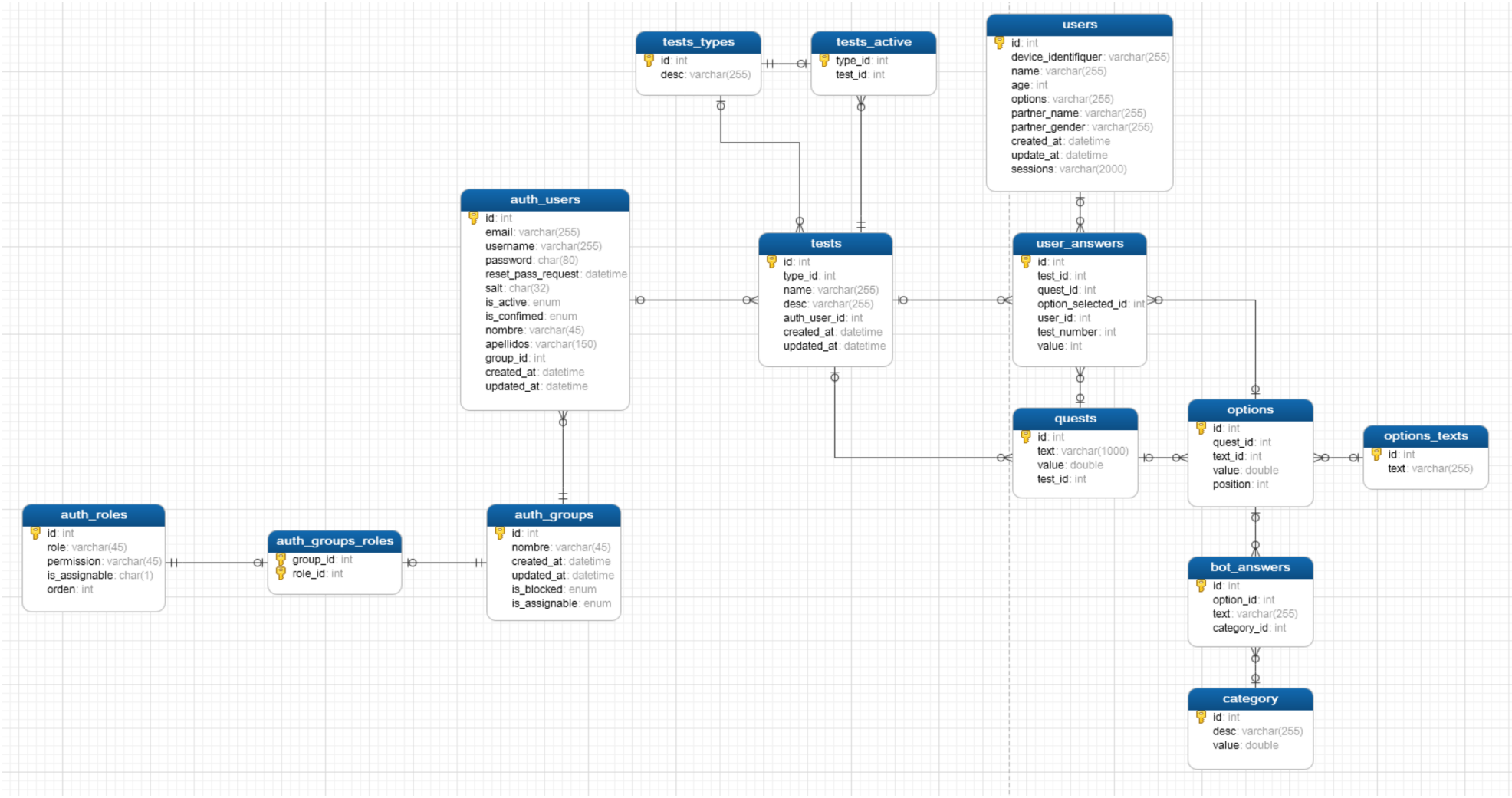

Figura 3-1: Modelo Entidad-Relación base de datos chatbot. Fuente: propia

3.1.3.2 Tablas

auth_groups: Contiene los grupos que existirán en la aplicación. Esta tabla no podrá ser modificada en la primera versión.

auth_groups_roles: En esta tabla se almacenarán la relación de roles con los grupos. Un grupo está relacionado con N roles. No podrá ser modificada en la primera versión.

auth_roles: Roles, determinan las acciones que se pueden llevar acabo por un grupo de usuarios. Solo existirá un registro y no será modificable en la primera versión, ya que todos los usuarios tienen el mismo rol.

auth_users: Usuarios de la aplicación. En esta tabla se almacenarán todos los datos para el acceso a la aplicación y el identificador se utilizará para registrar las acciones dentro de la aplicación.

bot_answers: Respuestas del bot. Cada registro está asociado N a 1 con las opciones.

Category: Determina la categoría a la que pertenece la pregunta. Se utiliza para ponderar las categorías. En un futuro podría servir para añadir nuevas clasificaciones.

options: Tabla donde se guardan las opciones asociados a cada respuesta.

options_texts: Textos para seleccionar en cada opción.

quests: Preguntas que se realizarán durante el test, estas estarán asociadas a un registro de la tabla, test.

tests: Contendrá los test existentes en la base de datos, se clasificarán por nombre y tipo.

tests_active: Determinará mediante una relación "*tests_types*"-"*tests*" cual es el test active para cada tipo.

tests_types: Determina a que grupo de test pertenece cada uno. En la primera versión se podrán crear todo los tipos deseados, pero solo se sincronizará el existente por defecto, ya que actualmente solo existe una aplicación de test.

user_answers: Aquí se almacenarán las respuestas individuales que ha dado cada usuario a cada pregunta.

users: Guarda los usuarios de la aplicación móvil que han llegado a sincronizarse

3.1.4 Despliegue sobre servidor

3.1.4.1 Fase Desarrollo

La fase de desarrollo se realizará sobre un entorno local ya que así se podrá ir realizando pruebas a mayor velocidad y sin tener que actualizar la versión de la aplicación alojada en un servidor en red.

El framework estará configurado en modo “develop” esto favorece las pruebas y reduce el número de recursos necesarios para mover la aplicación, ya que por ejemplo, no se esperan muchas conexiones simultaneas ya que solo habrá usuario utilizando la aplicación al mismo tiempo.

Aunque se utilice un entorno local, el código de sincronizará diariamente con el repositorio para poder mantener un buen control de las actualizaciones y tener actualizado el servidor en producción cuando esté desplegado.

3.1.4.2 Fase Producción

Para la fase de producción utilizaremos un servidor de la Universidad al que se nos ha dado acceso. El servidor funciona con Ubuntu y la aplicación funcionará sobre apache2.

La aplicación estará desplegada sobre un servidor web accesible desde internet. De esta forma podremos entrar desde cualquier ubicación y será posible la sincronización desde las aplicaciones móviles.

Por último, las actualizaciones se realizarán conectándonos en remoto por SSH con gestor FTP utilizando el repositorio GIT.

3.1.4.3 Fase Mantenimiento

Una vez terminada primera versión del software se desplegará una versión final en producción. A partir de entonces habrá que generar otra instancia en el servidor sobre la que realizaremos las pruebas de las nuevas actualizaciones antes de pasarlas a la instancia pública.

En el entorno de desarrollo se seguirá trabajando prácticamente de la misma forma, con la única diferencia es que las pruebas se realizará sobre la instancia secundaria de la aplicación en el servidor.

3.2 Aplicación Móvil

La aplicación móvil será la que llegue a los dispositivos de las usuarias, por lo tanto es en la que más hay que cuidar el diseño. Se necesitará que ofrezca **intimidad**, **cercanía** y **simpleza**.

3.2.1 Descripción de la aplicación

Al iniciar por primera vez la aplicación, nos encontraremos con una asistenta virtual que nos guiará desde el principio a fin. A través de la imagen de la chica intentaremos **acercarnos** lo máximo a posible a la usuaria, con un trato directo y amigable.

Las primeras preguntan sirven para identificar a la chica, nombre, edad, datos simples sobre su pareja y su estado personal actual. Estos datos más adelante se utilizarán para ofrecer un trato más personalizado y a nivel de servidor poder generar estadísticas.

El siguiente paso será pedirle una contraseña a la usuaria, que más adelante, utilizará para poder acceder a la aplicación. La contraseña será necesaria ya que una vez que se cierre la aplicación, cuando se vuelva a abrir, aparecerá otra aplicación de noticias ocultándola, para que en caso de que la pareja invada su **intimidad**, no pueda darse cuenta de que aplicación está utilizando realmente, aunque se recurriera al historial de aplicaciones, ésta no se podría identificar.

Una vez terminado el cuestionario inicial, se sugerirá si se desea continuar con la aplicación o si desea utilizarla en otro momento. Si se desea continuar o se retoma en más adelante, el siguiente paso será el test. El test podrá ir variando según cuando se ejecute, ya que la aplicación se sincroniza con el test que se encuentre activo en el servidor.

En el siguiente paso se presenta el test de una forma **sencilla**, con una serie de preguntas que contendrán un número de opciones de respuestas entre 2 y 4. Cada una de estas opciones está asociada a dos registros, un valor que se utilizará para calcular el resultado final de la prueba y una respuesta personalizada de la asistente virtual.

Al finalizar el test se calculará el resultado final y se dará unos consejos dependiendo del caso. En caso extremos estamos se plantea una herramienta que se pondría en contacto con la autoridad pertinente. En todos los casos al finalizar el test el resultado se sincronizará para poder realizar la estadística.

3.2.2 Utilidades implementadas

3.2.2.1 Cuestionario de registro inicial

Para el cuestionario inicial utilizaremos la misma pantalla que para el test principal. Será necesario recoger los siguientes datos de la usuaria:

- Nombre
- Edad
- Contraseña
- Nombre y género de la pareja
- Tres preguntas tipo adicionales (estas se han implementado para que puedan crecer de una forma sencilla)

También se presentará una forma rápida de abandonar la aplicación, que será tocando la imagen de la chica que nos guía en la aplicación.

En cuanto a la contraseña, después de registrar una, se mostrará de una forma sencilla como y donde habrá que introducirla posteriormente para entrar en la aplicación real.

Al acabar el cuestionario, se ofrecerá mediante dos botones la opción de continuar o abandonar la aplicación.

3.2.2.2 Cuestionario principal

Durante el test, el bot nos seguirá acompañando. Dicho bot, se situará en la parte superior de la pantalla, en la parte media las preguntas y las opciones y en la parte inferior el botón para continuar a la siguiente pregunta y un pequeño cuadro de texto oculto, que solo se mostrará cuando intenten pasar a la siguiente pantalla sin contestar.

La imagen del bot irá cambiando durante las preguntas para ofrecer un poco de dinamismo a la aplicación. Las opciones irán acompañadas de un "checkbox" para seleccionar la respuesta deseada. Los textos de alerta se mostrarán en rojo y en un tamaño inferior al del resto de los textos. Por último el botón para pasar a las siguientes preguntas se situará en la parte inferior izquierda de la pantalla.

Al finalizar los test, cambiará la estructura y en la parte media aparecerá un consejo acorde con el test y se activará la herramienta de alertas si fuera necesario.

3.2.2.3 Ocultación de la aplicación

Se diseñará una pantalla de noticias, que se mostrará siempre al iniciar la aplicación, si se ha completado el cuestionario inicial de registros.

Esta pantalla contendrá una serie de preguntas cargadas desde la base de datos, que ira variando aleatoriamente dependiendo del día de la semana y el mes, para que no se muestren las mismas si la pareja entrara distintos días.

Adicionalmente, se dotará a la herramienta con la capacidad para no ser identificable una vez minimizada, mostrando una pantalla en blanco si se ha salido de la aplicación desde cualquier pantalla que no sea la de noticias.

3.2.2.4 Sincronización de datos

Los datos se sincronizarán a través de internet, tanto por wifi como por conexión móvil. Al iniciar la aplicación aparecerá una pantalla de inicio con una animación, esta pantalla es conocida en Android como "splash". Durante los segundos que duré la pantalla el programa intentará sincronizarse con el servidor y actualizar la información de la base de datos. En caso de no tener conexión a internet, se utilizaría la información predefinida en la base de datos.

Durante la ejecución de la aplicación habrá dos momentos más donde se realizará la sincronización. Una después de completar el cuestionario inicial, en ella se enviará todos los datos correspondientes a la usuaria incluyendo el número de identificación del terminal, el cual se utilizará para identificarla a ella. La última sincronización se realizará al terminar los test e incluirá los resultados finales obtenidos en el test, junto con sus respuestas.

3.2.3 Base de datos

Esta base de datos se encuentra en format SQLite y almacena la información sincronizada por el servidor y la información generada por ella misma.

La aplicación viene con un test predefinido que se utilizará en el caso de que no haya conexión a internet.

3.2.3.1 Tablas

bot_answers: Almacena las respuestas del bot a cada opción seleccionada.
category: Se utiliza para seleccionar una respuesta del bot, dependiendo de cómo vaya transcurriendo el test.
options: Todas las opciones relacionadas con cada pregunta.
options_texts: Textos disponibles para cada una de las opciones.
quests: Preguntas que se realizarán durante el test.
results: Resultado que se mostrará dependiendo de la puntuación obtenida.
user: datos del usuario que se irán registrando a lo largo de la aplicación.

3.2.3.2 Modelo entidad-relación Base de datos móvil

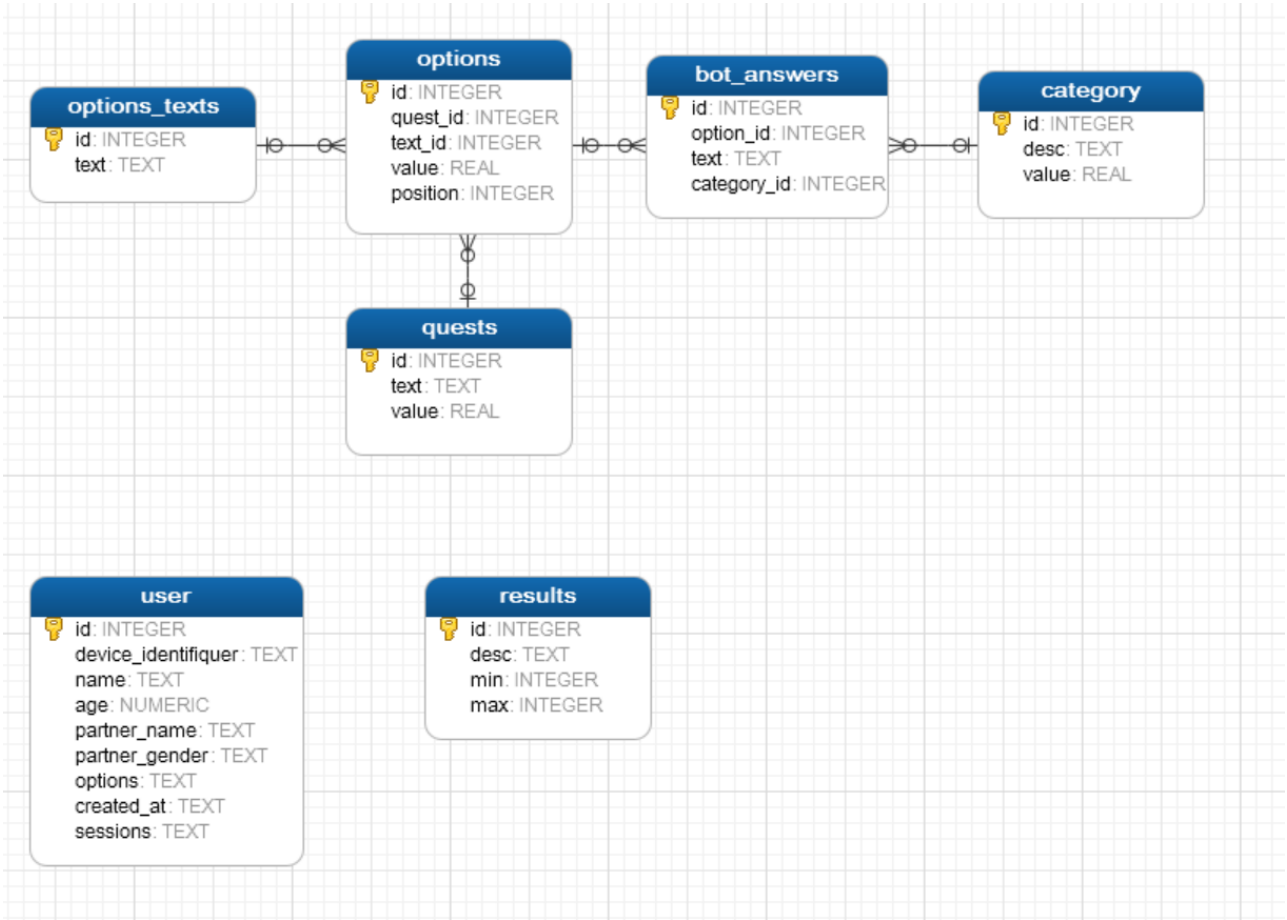

Figura 3-2: Modelo Entidad-Relación base de datos chatbot.db3. Fuente: propia

3.3 Sistema de comunicación aplicaciones

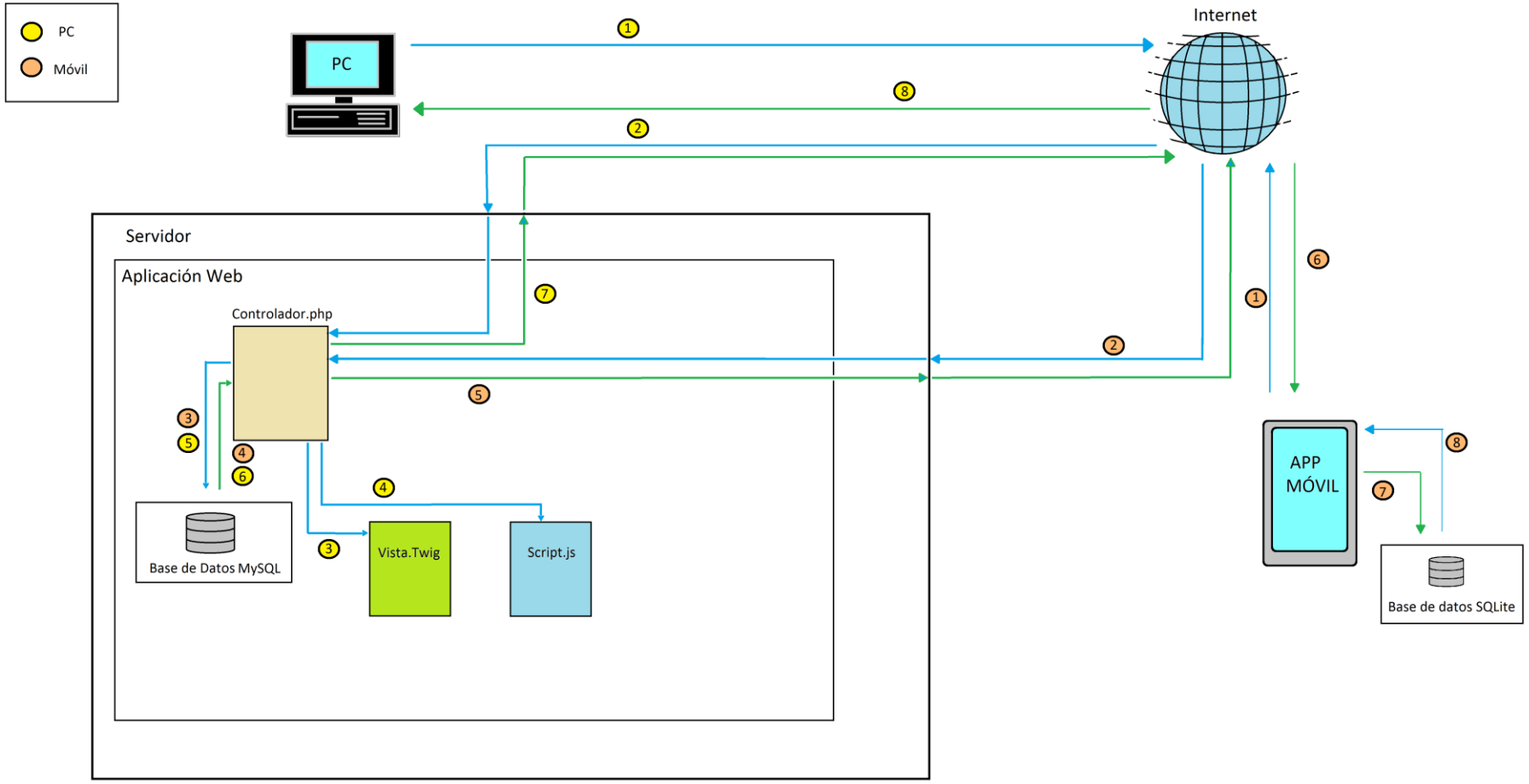

Figura 3-3: Diagrama de flujo del sistema de comunicaciones del servidor. Fuente: propia.

3.3.1 Aplicación Web

El gráfico describe la comunicación entre ordenador y servidor.

1- Petición GET/POST del PC al servidor.

2- La petición alcanza al servidor y se dirige al controlador para procesar la información. La aplicación utilizará el fichero routes.php para traducir su parte de URL y encontrar el método pedido.

3- Si la petición lo solicita, se procesará la vista indicada.

4- Si la vista necesita de funciones JavaScript también se incluiría el fichero asociado.

5- Se ejecutan consultas sobre bases de datos. De selección, para incluir información en la vista o para realizar cálculos. De inserción, para actualizar registros.

6- Resultado de la consulta.

7- Se genera el resultado de la petición y sale del servidor.

8- El resultado llega al PC origen.

3.3.2 Aplicación Móvil

El gráfico describe la comunicación entre dispositivos móviles y servidor.

1- Petición GET al servidor, para obtener la cadena JSON con el contenido de la base de datos. Post, cuando se envía la información del usuario o los resultados del test.

2- La petición llega al servidor y se procesa el tipo de solicitud.

3- Se actualiza o se selecciona información de la base de datos dependiendo de la petición.

4- Resultado de las consultas.

5- En el caso de la petición GET, se construirá la cadena JSON y se enviará. En otros casos, se enviará confirmación o error, dependiendo del resultado.

6- La respuesta llega a la aplicación móvil.

Los dos siguientes pasos serán los primeros si se trata de una petición POST

7- Se ejecuta selección o inserción sobre la base de datos móvil.

8- Se recogen y procesan resultados.

4 Desarrollo

4.1 Aplicación Web

4.1.1 Introducción

A continuación se describe el proceso de desarrollo que se ha llevado a cabo para cumplir los objetivos marcados.

Se describirán las clases principales, centrándonos esencialmente en los aspectos de programación, no explicaremos la instalación de paquetes externos, como si ha configurado los ficheros del framework o que parámetros se han establecido para el funcionamiento de PHP trabajaremos.

4.1.2 Controladores

Se han diseñado los siguientes controladores donde se encuentran todas las funcionalidades de nuestra aplicación. En la imagen se puede observar claramente definida la estructura.

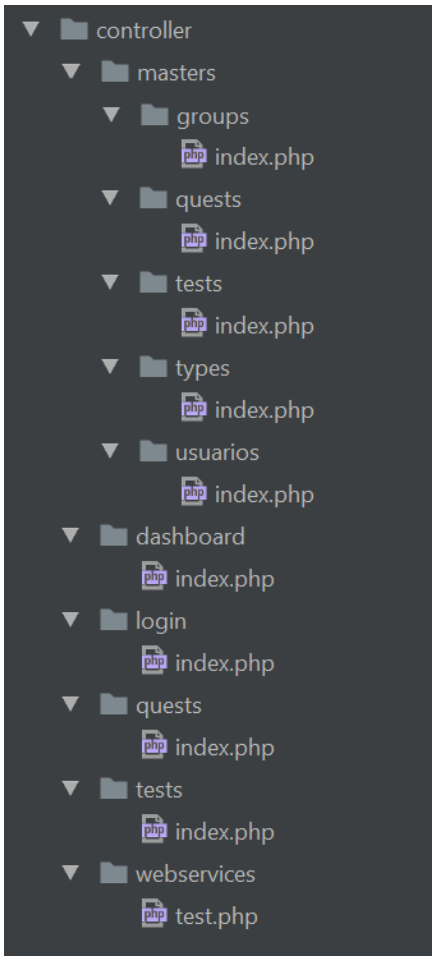

Figura 4-1: Estructura controladores PHP. Fuente: propia.

En este proyecto, cada controlador tiene una pantalla asociada, toda acción que se realice en una pantalla estará procesado por cada uno individualmente. Los controladores no se llaman entre ellos, éstos solo son llamados a través de peticiones web.

4.1.2.1 Login

Este controlador será al que se acceda cuando entremos en la aplicación web. La clase carga una vista donde se presentan un formulario para introducir usuario y contraseña.

El formulario es devuelto mediante una petición POST al controlador que certificará si las credenciales son correctas, en caso de serlo el usuario es redirigido a la pantalla de bienvenida.

Para implementar este controlador se ha utilizado la librería "Auth". Con ella se ha implementado un sistema para controlar el acceso a la aplicación. Si se autoriza el acceso desde la clase "login", se crea una variable de sesión que contiene los datos del usuario, que nos sirven para registrar sus acciones en la aplicación.

Con la variable de sesión también se controla el tiempo que lleva el usuario inactivo en la aplicación, se registra la hora en la que realizó la última acción, si la siguiente acción se realiza pasados veinte minutos, se elimina la sesión y se redirige al usuario a la pantalla de logueo.

4.1.2.2 Dashboard

Esta clase controla la pantalla de bienvenida, extrae los datos de los resultados de los test sincronizados por la aplicación móvil y los configura para introducir en una gráfica que se muestra en la vista. La gráfica se genera a través de la librería JavaScript "charts.js".

La utilización de esta herramienta es bastante simple, solo es necesario ubicar un contenedor para la gráfica en el fichero HTML (en nuestro caso Twig) y ejecutar la función JavaScript para que pinte la gráfica, a esta gráfica habrá que pasarle los datos que queremos mostrar. Para esto utilizaremos el controlador, que además de los datos predefinidos que muestra la gráfica, cuenta con un filtro que permite personalizarla.

Los datos alojados en la gráfica también son descargables a un fichero CSV. Para crear el fichero, simplemente usamos los mismos datos que utilizamos para dibujar la gráfica, pero lo volcamos sobre un fichero utilizando la función nativa de PHP "fputcsv", ésta nos permite introducir una línea sobre un fichero con cada llamada. La forma de uso es muy sencilla, solo es necesario pasarle como primer argumento el recurso que contenga el fichero que estamos generando y un array que contengan los datos de la fila, cada elemento del array se colocará en una celda. La función permite más parámetros de configuración, pero debido a que no han sido necesarios, no profundizaremos más en ello.

4.1.2.3 Masters

Esta Categoría contiene 5 clases, se agrupan todas juntas debido a que son distintos elementos, pero todos se rigen por el mismo concepto. Son elementos necesarios para funcionamiento de la aplicación, nos servirán para configurar el uso de la aplicación o como propiedades de otros elementos más complejos de la aplicación. Los cinco controladores están pensados de la misma forma, todo implican un modelo CRUD relacionado con su registro en la base de datos.

- Groups: Actualmente no se utiliza, se ha desarrollado para facilitar la inclusión de la herramienta de roles en un futuro.
- Quests: Permite manejo de los registros básicos de las preguntas (texto y valor).
- Tests: Desde aquí se crearán los test relacionados con cada tipo
- Types: Clase necesaria para generar distintos tipos de test.
- Usuarios: Gestiona los usuarios de la aplicación Web.

En la ***Figura 4-2***, en la siguiente página, se muestra este controlador y el desarrollo realizado.

Como se observa en dicha imagen, primero recurrimos a la función "Before", en FuelPHP esta función se ejecutará siempre que se acceda al controlador, sin importar el método llamado. En esta función se realiza la carga de los datos en común de todas las vistas controladas por este fichero. Del código que compone el método, puntualizar que el método "Casset" pertenece a un paquete perteneciente a FuelPHP, pero que no viene de forma predefinida, sirve para incluir un script en las vistas. Por último la función "parent::before()" llama a la clase de la que extiende el controlador, ésta realizará tareas de gestión de más bajo nivel.

El otro método que se muestra extendido en el código, es en el que se entrará cuando el controlador reciba una petición mediante GET a la dirección "index" de la clase. En el primer paso del código se carga la vista utilizando el método "View::forge", éste nos permite convertir en variable nuestra vista. Con la variable preparada, se comienza a cargar los datos, en la línea 48 se llama a la clase "Model_OptionsTexts", esta clase extiende de "Model", clase perteneciente al ORM, se profundizará en ella más adelante. Para terminar con el método se utiliza la función "Response::forge()", con ella construiremos la respuesta que se devolverá a la petición.

Los métodos que están contraídos completan el CRUD:

- Get_update($id,$id_opttion): devuelve la vista para actualizar o crear opciones para los textos.

- Post_update(): Método donde se recibe el formulario y se actualiza la base de datos.

- Get_detele($id): Elimina el registro indicado en el parámetro "$id".

- Load_form: método interno que utiliza la clase para rellenar el formulario de la vista de edición cuando se quiere editar un registro.

Los demás controladores pertenecientes a master se han construido reutilizando esta estructura.

```php
<?php

use Fuel\Core\Response;
use \Parser\View;

class Controller_Master_Optiontexts_index extends \Controller_App{

    //Variable donde se irán almacenando las migas de pan
    private $_bc = array();

    //En este array incluiremos los elementos que queremos que se activen del menú lateral
    //En este caso activaremos Maestras > Textos para las opciones.
    private $_option_menu=array('masters','texts');

    public function before(){

        //Se cargan funciones de javascript específicas de esta funcionalidad
        //Casset es una herramienta de FuelPHP que prepara el fichero JS para enviar con la respuesta.
        Casset::js('index.js');

        //Configuramos navegación de las migas de pan, se instanciará: descripción y routes.
        //La descripción de la página actual se configurará como la clase activa
        $this->_bc[] = array('ds'=>'Inicio','url'=>'/dashboard');
        $this->_bc[] = array('ds'=>'Maestras','url'=>'/dashboard');
        $this->_bc[] = array('ds'=>'Textos para las opciones','url'=>'/master/optiontexts/texts','class'=>'active');

        //Llamamos al método Before de la clase padre.
        parent::before();
    }

    public function get_index()
    {
        //Cargamos la vista que queremos mostrar.
        //La clase View del módulo nos permitirá cargar las vistas y trabajar con ellas antes de enviarla al navegador del usuario.
        $view = View::forge('master/optiontexts/index.twig');

        //Seteamos la variable que contendrá el título de la pantalla.
        //La clase View trata a la vista como un objeto, permitiendonos acceder de forma sencilla a las variables de la vista.
        $view->title = "Gestión de textos vinculados a las opciones";

        //Colocamos las migas de pan configuradas anteriormente en la vista que estamos preparando
        $view->bc = $this->_bc;

        //Ponemos a activas las entradas del menú correspondientes.
        $view->option_menu = $this->_option_menu;

        //Cargamos las opciones de texto almacenadas en la base de datos y las introducimos una variable de la vista.
        $view->options_texts = Model_OptionsTexts::query()->get();

        return Response::forge($view);
    }

    public function get_update($id,$id_quest=0){...}

    public function post_update(){...}

    public function get_delete($id){...}

    public function load_form($optionText, $tipo, $id_quest=0){...}


}

```

Figura 4-2: Fichero Controller_Master_OptionTexts_Index.php . Fuente: propia.

4.1.2.4 Tests

Este controlador, con la ayuda de **"Quest"**, está dedicado a la construcción de los test. La clase **"Test"** maneja las pantallas donde se muestran los test pertenecientes a una categoría, previamente seleccionada.

La primera y la segunda pantalla de la clase **"Test"** se obtiene a través de la función mostrada en la ***Figura 4-3***. En FuelPHP los métodos que empiezan con "get_" son accesible con una petición de tipo GET y reciben datos en los a través de los parámetros.

Como se observa en la función, si no se reciben parámetros, los valores por defecto de los argumentos serán cero, en ese caso el controlador solo extraerá los tipos de test de la base de datos y los dispondrá en el selector.

En caso de recibir parámetros, si el argumento "$type" es distinto de cero se procede a presentar la pantalla con todos los test de este tipo. La variable "$id" contiene el identificador del test solicitado, si es mayor que cero se busca en la base de datos y se presenta en la vista, sino, se selecciona el test activo dentro del tipo seleccionado. La vista devuelta se puede ver en la ***Figura 4-13*** y ***Figura 4-14***.

La pantalla de edición del test permite modificar los valores del test, gestionar las preguntas relacionadas con éste y seleccionar o crear test asociados del mismo tipo. Si se modifican los valores del test, es necesario guardar pulsando un botón que envía por el método POST el formulario al controlador, la función actualiza los nuevos valores y devuelve la vista. Si se selecciona otro test, se vuelve a acceder al método "get_index" con el id seleccionado, y se retorna la misma pantalla con los datos solicitados. Si se quiere gestionar las preguntas se accede al controlador **"Quest"**.

```php
public function get_index($id=0,$type=0)
{
    //Cargamos la vista
    $view = View::forge('tests/index.twig');

    //Este paramatro indica el id del test que estamos intentando cargar, si la función recibe un id distinto de 0
    //Procedemos a la carga de la vista con el id seleccionado. Sino, cargamos en la vista la selección de tipo.
    if($id!=0)
    {
        //añadimos a las migas del controlador un nivel adicional
        $this->_bc[] = array('ds'=>'Tests','url'=>'/tests');

        //Cargamos el test activo
        $active = Model_TestsActive::query()->where('type_id',$type)->get_one();

        if($id!=0)
        {
            //Cargamos los datos del test seleccionado
            $test   = Model_Tests::query()->where('id',$id)->get_one();
        }
        else
        {
            //Si hay algún test activo lo seleccionamos
            $test   = isset($active->tests)?$active->tests: null;
        }

        //Se extraen todos los test para mostrarlos en una lista
        $tests = Model_Tests::query()->where('type_id',$type)->get();

        //Guardamos en una variable todas las preguntas del test en una variable que posteriormente generará un tabla
        $quests = $test->quests;

        //Titulo de la vista
        $view->title = "Gestión de Tests";


        //Cargamos todas las variables en la vista
        $view->test           = $test ;
        $view->tests          = $tests ;
        $view->quests         = $quests ;
        $view->test_active_id = $active->test_id;
        $view->typeSelected   = $type ;

    }
    else
    {
        //Sino se carga ningún test simplemente mostraremos el seleccionador de tipos
        //Titulo de la vista
        $view->title = "Selección de tipo";

    }


    //Rellenamos el selector de la vista con todos los tipos
    $view->tests_types = Model_testsTypes::query()->get();

    //Paso de las opciones de menu que deben aparecer seleccionadas
    $view->option_menu = $this->_option_menu;

    //Paso de las migas de pan a la vista
    $view->bc = $this->_bc;

    return Response::forge($view);

}
```

Figura 4-3: Función get_index del fichero Controller_Test_Index.php. Fuente: propia

4.1.2.5 Quests

La clase **"Quest"** se encarga de la pantalla de creación y edición de las preguntas. Si se accede a ella para crear una nueva, se presenta una pantalla con un formulario vacío, si se direcciona para la edición, aparece el formulario completado con los datos de la pregunta seguida del número de opciones asociadas. Esta vista permite añadir opciones, editarlas o eliminarlas. Para ver la vista generada ver ***Figura 4-15***.

La vista posee un botón guardar, al ser pulsado se envía a través de POST un formulario que contendrá toda la información de la vista: Preguntas con texto y valor, opciones con descripción del texto y el valor de la opción y las respuestas a las opciones. Debido a que en una sola petición se almacenan muchos datos, se ha

desarrollado un algoritmo que intentando evitar al máximo la redundancia y la duplicidad del código.

Para ello dentro de la función "post_index" se ha desarrollado un bucle que recorre todos los elementos del formulario recibido, en el bucle se distribuye cada iteración mediante un "Switch" ejecutando únicamente las instrucciones necesarias para cada una. Se ha intentado crear un código, lo más genérico posible, para que sea reutilizable para cada propiedad de los objetos que se desea almacenar. Se muestra el bucle en ***Figura 4***, ***Figura 5*** y ***Figura 6*** se ha separado en dos imágenes para que facilitar la lectura del código.
Como principal virtud del algoritmo, al añadir nuevas propiedades a una tabla de la base de datos, solo es necesario incluir una nueva entrada en el formulario de la vista, siguiendo el formato indicado en los comentarios de código.

Ejemplo, si se añade la nueva celda "position" a la tabla de las preguntas, que indicaría cuál sería su posición dentro del test. A la hora de implementar el cambio solo sería necesario colocar una nueva entrada en el formulario nombrada como "quest_position". El controlador, sin tener que modificar el código, detectaría automáticamente la nueva propiedad y la almacenaría en la base de datos.

```php
//Acedemos al valor de quest_id e intentamos extraer la pregunta de la base de datos
$id = Input::post('quest_id');
$quest = Model_Quests::query()->related('options')->related('options.answers')->where('id',$id)->get_one();
//Validación para nuevas opciones, cada posición pertenece a una de las posibles nuevas opciones,
// si la posición tiene valor 1, la opción será almacenada
$new_options_validator = array(0,0,0,0);
$option_aux = null;
//Recorremos cada elemento de la entrada POST con la configuración $key => $value
foreach (Input::post() as $key => $value)
{
    //Los nombre de cada entrada del formulario tienen un formato especial dependiendo de que elemento vayamos
    // a almacenar

    //Con este método generamos un array con los 3 primeros elementos separados por "_"
    $key_array = explode("_",$key,3);

    switch ($key_array[0])
    {
        //QUEST. En el caso de los valores para las preguntas el formato será el siguiente:
        // "quest_nombreDeLaPropiedad".
        //Ejemplos: quest_id, quest_test_id, quest_text, quest_value
        case "quest":
        {
            //Si se trata de una pregunta nueva, generamos una variable para rellenarla.
            if(!$quest)
            {
                $quest = Model_Quests::forge();
            }
            //El valor de "id" no nos interesa, ya que lo obtenemos al inicio de la función,
            // si es cualquier otro valor, intentamos almacenarlo.
            if($key_array[1] !="id")
            {
                //si existe un tercer elemento quiere decir que el nombre en la base de datos esta formado
                // con "_", por lo tanto procedemos a la concatenación, para poder almacenarlo posteriormente.
                if(isset($key_array[2]))
                {
                    $key_array[1].="_".$key_array[2];
                }
                //Se guarda en la propiedad del objeto el valor recibido
                $quest->$key_array[1] = $value;
            }
            break;
        }
```

Figura 4-4: Inicio de bucle, caso "quest", función "post_index", fichero Controller_Quest_Index.php. Fuente: propia

```php
//OPTION. En este caso se complica un poco, existiendo dos formatos:
// si se trata de una edición "option_identificadorDeLaOpcion_nombreDeLaPropiedad",
// si es nueva "option_0_posiciónEntreLasOpciones_nombreDeLaPropiedad"
//Ejemplos: option_{{ option.id }}_value, option_0_{{ i }}_value.
case "option":
{
    //Si aun no existe Id para la pregunta, en la vistas no se muestran las opciones,
    // por lo tanto, no hará falta realizar ninguna acción.
    if($id=="")
    {
        break;
    }
    //En esta condición, identificamos si se trata de una nueva opción.
    if(intval($key_array[1])==0)
    {
        //Como en la anterior operación Explode, solo se cogen los 3 primeros elementos,
        // ahora se cogeran los restantes.
        //$key_array[2] = {{ i }}_value.
        $option_auxArray = explode("_",$key_array[2],2);
        //Una vez ejecutado explode $option_auxArray[1]= value;
        // nombre de la propiedad que queremos almacenar.
        $key_aux= $option_auxArray[1];
        //text_id es la propiedad fundamental de opciones ya que es la que texto se va a mostrar,
        // Si este valor es >0
        //procederemos a intentar guardar la opción
        if($key_aux == "text_id")
        {
            if($value!=0)
            {
                //$option_auxArray[n]-1 => option[0](1..4) $new_options_validator[0..3]
                //En esta opción se genera $option_aux,
                // Para reducir la complejidad de determinar en que pasos se debería
                // crear o destruir esta variable, se ha recurrido a $new_options_validator que indica
                // al algoritmo si en los siguientes pasos será necesario guardar los valores que lleguen.
                $new_options_validator[$option_auxArray[0]-1]=1;
                $option_aux = Model_Options::forge();
                $option_aux->$key_aux = $value;
            }
        }
        else
        {
            //$option_auxArray[n]-1 => option[0](1..4) $new_options_validator[0..3]
            //Si en valor de $new_options_validator es igual a 1 se guarda el elemento.
            if($new_options_validator[$option_auxArray[0]-1]==1)
            {
                $option_aux->$key_aux = $value;
                $option_aux->quest_id = $id;
                $option_aux->save();
            }
            else
            {
                $option_aux=null;
            }
        }
    }
    //Para los elementos que no son nuevos, simplemente recorremos la relación ya existente
    foreach ($quest->options as $option)
    {
        //Si el id de la opción es el mismo que el del ítem de esta iteración,
        // actualizamos el valor en la base de datos.
        if($option->id == intval($key_array[1]))
        {
            $option->$key_array[2] = $value;
        }
    }
    break;
}
```

Figura 4-5: Parte media de bucle, caso “option”, función “post_index”, fichero Controller_Quest_Index.php. Fuente: propia

```
            //Answer.
            //Si pertenece a una opción que ya existe:
            // answer_{{ answer_aux_id }}_text_category_{{ category.id }}_option_{{ option.id }}
            //Si pertenece a una nueva opción: answer_0_text_category_{{ category.id }}_option__0_{{ i }}.
            // Con la i podremos relacionarla con la opción.
            case "answer":
            {
                //Si aún no existe Id para la pregunta, en la vistas no se muestran las opciones,
                // por lo tanto, tampoco se muestran respuestas.
                if($id=="")
                {
                    break;
                }
                //Se comprueba si se trata de un nuevo elemento, si es así se crea y se guardan sus propiedades
                //Sino se busca y se modifica el elemento
                if(intval($key_array[1])== 0 )
                {
                    if($value!="")
                    {
                        $botAnswers = Model_BotAnswers::forge();
                        if($value==0)
                        {
                            if($option_aux==null)
                            {
                                break;
                            }
                        }
                        $botAnswersArray = explode("_",$key_array[2]);
                        $botAnswers->category_id = $botAnswersArray[2];
                        $botAnswers->option_id   = ($option_aux==null) ? $botAnswersArray[4]: $option_aux->id;
                        $botAnswers->text        = $value;
                        $botAnswers->save();

                    }
                }
                else
                {
                    $break_flag = false;
                    foreach ($quest->options as $option)
                    {
                        foreach ($option->answers as $answer)
                        {
                            if($answer->id == intval($key_array[1]))
                            {
                                $key_aux = explode("_",$key_array[2])[0];
                                $answer->$key_aux = $value;

                                $break_flag=true;
                                break;
                            }
                        }
                        if ($break_flag)
                        {
                            break;
                        }
                    }
                }
                break;
            }
        }
    }
    //Una vez acabado el bucle se guardan todos los cambios que se hayan producido en la pregunta
    $quest->save();
    //Por ultimo se redirecciona a la edición de la pregunta con los campos actualizados.
    return Response::redirect("quest/update/$quest->id");

}
```

Figura 4-6: Fin bucle y función, caso “answer”, función “post_index”, fichero Controller_Quest_Index.php. Fuente: propia

4.1.2.6 WebServices

Este controlador se encarga de la gestión de la sincronización con la aplicación. Se trata de una clase libre de las restricciones del módulo "auth", por lo tanto es posible acceder a ella sin haber creado antes la variable de sesión, es decir, sin loguerarse. Como los procesos que realiza son para gestionar la comunicación de la aplicación, en este caso no se devuelve ninguna vista, sino una cadena de texto en formato JSON o una simple respuesta de confirmación.

Se han definido dos tipos de sincronizaciones, uno por función.

a) Envió de información a plataforma móvil.
b) Recepción de información de la plataforma móvil.

Para el tipo **A**, se ha definido la función "get_index" en el controlador "Controller_Webservices_index", la función se puede observar en la ***Figura 4-7***.

En el método, primero se generan las variables de texto que se usarán para formar la cadena JSON.

El siguiente paso decodifica las variables creadas, convirtiéndolas en objetos PHP y se extrae de la base de datos el test actualmente activo.

Después introducimos los valores del test en las variables creadas utilizando una función propia "setQueryInArrayJson", a este método le pasaremos los registros y la cadena donde queremos introducirlos y nos devolverá el objeto completo, se puede ver información más detallada sobre esta función en la sección 4.1.5.

Por último, se prepara la respuesta, se introduce la cadena de texto codificada en el cuerpo de la respuesta y se retorna.

El tipo **B**, se muestra en la ***Figura 4-8***, para este tipo de comunicación se ha creado la función "post_index" en el mismo controlador que en el caso A. Esta función recibe, tanto la información del usuario, como el resultado de los test.

Se ha programado de una forma genérica para simplificar el código, la función almacena cualquier información perteneciente al usuario que se le envíe, siempre que la cadena JSON enviada este bien formada.

En la función primero se decodifica la información recibida para poder trabajar con ella, después se comprueba si existe el usuario que envía la información, sino existe se crea uno nuevo.

Recorremos el JSON, almacenando cada uno de los valores contenidos en la cadena. Si no se produce ningún error se envía una respuesta con valor "OK".

```php
public function get_index()
{
    $test_json = "
    {
        \"test_info\" : {},
        \"category\": [],
        \"options_texts\":[],
        \"result\":[],
        \"quests\":[]
    }";
    $result_json = "
    {
        'text' :'#',
        'value':'#'
    }";
    $category_json = "
    {
        \"id\"  :\"#\",
        \"desc\":\"#\",
        \"value\":\"#\"
    }";
    $option_texts_json = "
    {
        \"id\"  :\"#\",
        \"text\":\"#\"
    }";
    $query_json = "
        {
            \"text\"   :\"#\",
            \"value\"  :\"#\",
            \"options\":
            [
                {
                    \"text_id\"   :\"#\",
                    \"value\"     :\"#\",
                    \"position\"  :\"#\",
                    \"answers\":
                    [
                        {
                            \"category_id\":\"#\",
                            \"text\"      :\"#\"
                        }
                    ]
                }
            ]
        }
    ";
    $test_info_json = "
    {
        'test_id':'#',
        'updated':'#'
    }";

    $category_json      = json_decode($category_json);
    $option_text_json   = json_decode($option_texts_json);
    $query_json         = json_decode($query_json);
    $test_json          = json_decode($test_json);
    $test_info_json     = json_decode($test_info_json);

    $test_active = Model_TestsActive::query()->related('test')->related('test.quests')
                            ->related('test.quests.options')->related('test.quests.options.answers')->get_one();

    $test_info_json['test_id'] = $test_active->test->id;
    $test_info_json['updated'] = $test_active->test->updated_at;

    //GET ALL VALUES OF QUESTS FROM ACTIVE TEST
    $test_json->quests        = $this->setQueryInArrayJson($test_active->test->quests        , array($query_json));
    $test_json->category      = $this->setQueryInArrayJson(Model_Category::query()->get()    , array($category_json));
    $test_json->options_texts = $this->setQueryInArrayJson(Model_OptionsTexts::query()->get(), array($option_text_json));
    $test_json->result        = $this->setQueryInArrayJson($test_active->test->result        , array($result_json));
    $test_json->test_info     = $test_info_json;

    $response = new Response();
    $response->set_header('Content-Type', 'application/json; charset=utf-8');
    $response->body(json_encode($test_json));

    return $response;
}
```

Figura 4-7: Función de sincronización de la base de datos con la aplicación, pertenece al fichero Controller_Webservices_Test.php. Fuente: propia.

Formato del JSON que devuelve la función “get_index":

```
{
        "test_info": {
                "test_id": "#",
                "updated": "#"
        },
        "category": [{
                        "id": "#",
                        "desc": "#",
                        "value": "#"
                }
        ],
        "options_texts": [{
                        "id": "#",
                        "text": "#"
                }
        ],
        "result": {
                "text": "#",
                "value": "#"
        },
        "quests": [{
                        "text": "#",
                        "value": "#",
                        "options":
                        [{
                                        "text_id": "#",
                                        "value": "#",
                                        "position": "#",
                                        "answers":
                                        [{
                                                        "category_id": "#",
                                                        "text": "#"
                                                }
                                        ]
                                }
                        ]
                }
        ]
    }
```

```php
public function post_index()
{
    $user_array_json = json_decode(Input::post('content'));

    $user = Model_Users::query()->where('device_identifiquer', $user_array_json->device_identifiquer)->get_one();

    if($user)
    {
        unset($user_array_json->device_identifiquer);
    }
    else
    {
        $user = Model_Users::forge();
    }

    foreach ($user_array_json as $key => $value)
    {

        if($value!="#$key#")
        {
            $user->$key = $value;
        }
    }

    $user->save();

    return $this->response("OK");
}
```

Figura 4-8: Función sincronización información sobre ususario aplicación móvil, pertenece al fichero Controller_Webservices_Test.php. Fuente: propia

Formato JSON que recibe “post_index”:


```
{
        "device_identifiquer": "FUH02168XXXXXXXX",
        "name": "#name#",
        "age": "#age#",
        "options": "#options#",
        "partner_name": "#partner_name#",
        "partner_gender": "#partner_gender#",
        "created_at": "2016-12-23 21:41:23",
        "sessions": "2016-12-23 21:41:23,2016-12-24 17:53:17"
}
```

4.1.3 Vistas

Para el desarrollo de las vistas nos hemos apoyado en la Inspinia, una plantilla desarrollada con Bootstrap, HTML5 y CSS3. Además implementa gran cantidad de componentes con plugins jQuery integrados. La utilización de plantillas también se ha extendido mucho los últimos años, su bajo coste y las numerosas herramientas que ofrece, facilita y abarata el desarrollo de cualquier aplicación web.

En esta sección nos centraremos más en mostrar el resultado que el código contenido en las vistas, dando ejemplos de las vistas de los controladores de los que se ha hablado anteriormente en el documento.

4.1.3.1 Login

Esta vista se muestra al entrar en la aplicación, es una página inicial simple que podría contener alguna funcionalidad más, pero no se consideró oportuno debido a que en un principio la relación entre interesados y programador es directa. Si se tratara de un software a terceros, tendría que añadir como mínimo el sistema de recuperación de claves.

El desarrollo de esta ventana es sencillo, se ha colocado un contenedor de imágenes junto a un formulario que envía la información de vuelta al controlador y autoriza la sesión. La única dificultad que entraña este módulo es el manejo de la plantilla y sus propiedades, ya que requiere un pequeño estudio previo de la documentación y conocimientos HTML y CSS.

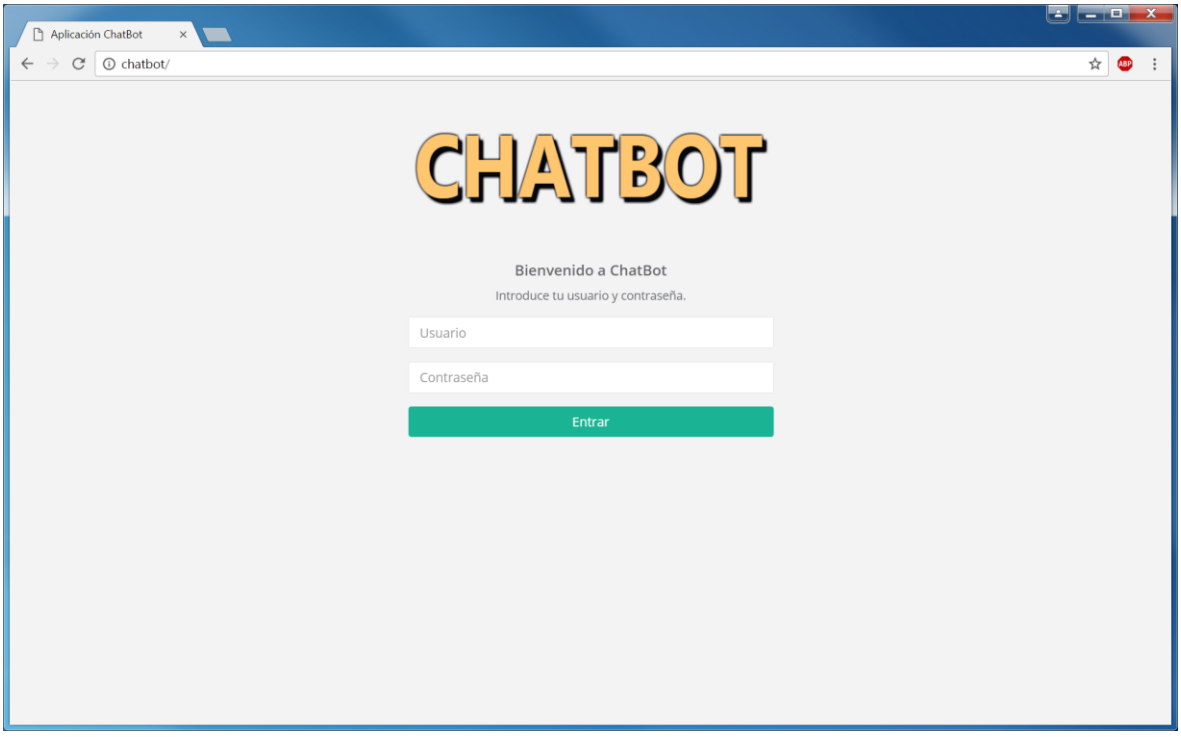

Figura 4-9: pantalla Login ChatBot. Fuente propia.

4.1.3.2 Dashboard

En esta pantalla es donde se muestra la gráfica con sus filtros, la gráfica se genera sobre un contenedor de la vista a través de un módulo JS asociado.

En la vista que se muestra aún no se había colocado ni la gráfica ni los filtros, pero ya da una idea de lo que se busca con esta pantalla.

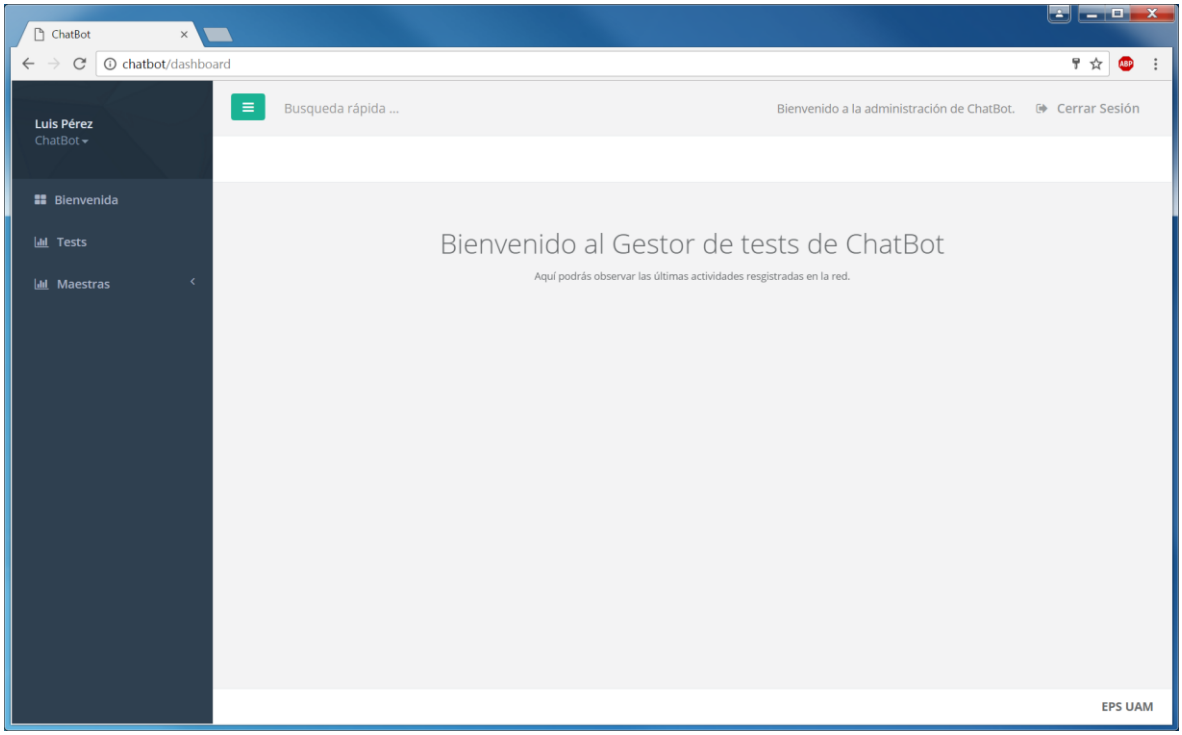

Figura 4-10: pantalla Dashboard ChatBot. Fuente propia.

4.1.3.3 Masters

De Master se muestras las vistas más sencillas, las de opciones de texto, ya que este es el modulo explicado en los controladores. La primera vista ***Figura 4-11***, contiene una tabla donde podremos gestionar los registros existentes y un botón donde añadir nuevos registros.

La tabla está implementada con "datatable" de Bootstrap que incluye funciones jQuery que permiten ordenar la tabla simplemente pulsando sobre los textos. "Datatable" también ofrece múltiples herramientas, relativamente sencillas de añadir, como exportación de la tabla a CSV, casilla de búsqueda en la tabla, paginado, etc. Los estilos de la tabla están configurados a través de la plantilla.

Por otro lado, los botones de añadir y editar, llevan a la misma vista (***Figura 4-12)***, que en un caso estará vacía y en otro completa por los datos pertenecientes al registro que se quiere editar. En el caso de la imagen se muestra una edición.

Esta segunda pantalla solo contiene un formulario con una celda. Para crear un nuevo registro o modificar el valor, se introduce un valor en la casilla y se pulsa “Guardar”, este redireccionará a la pantalla anterior y donde en la tabla se podrá localizar el nuevo elemento creado o modificado.

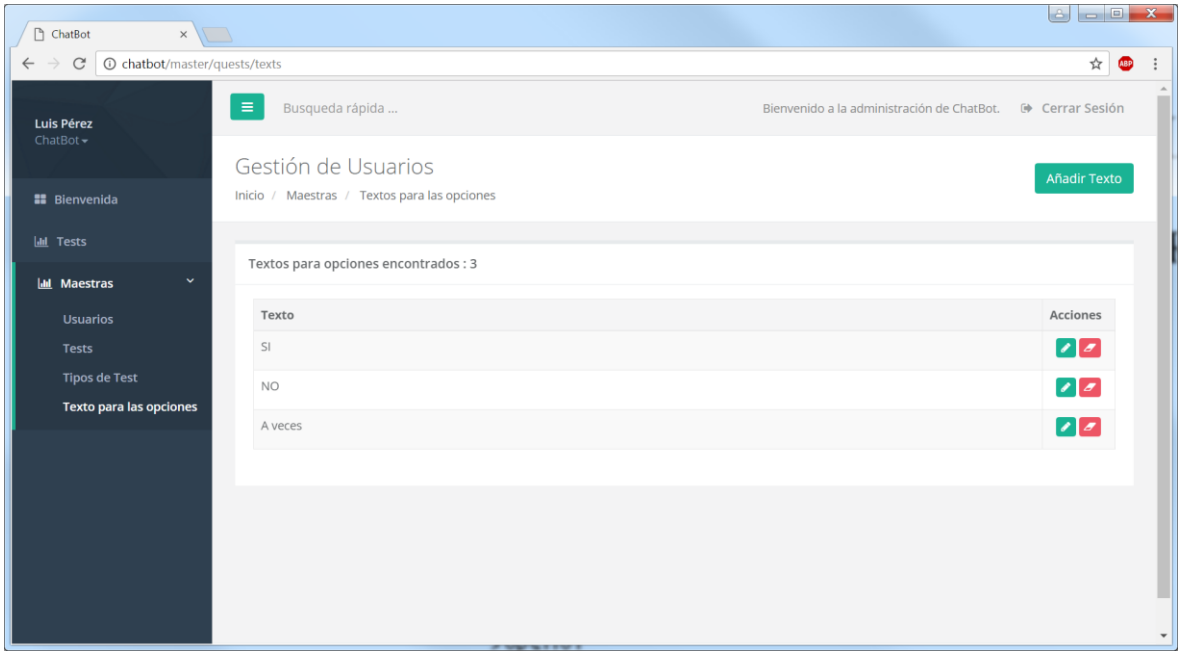

Figura 4-11: Pantalla listado Maestra Texto para opciones. Fuente: propia.

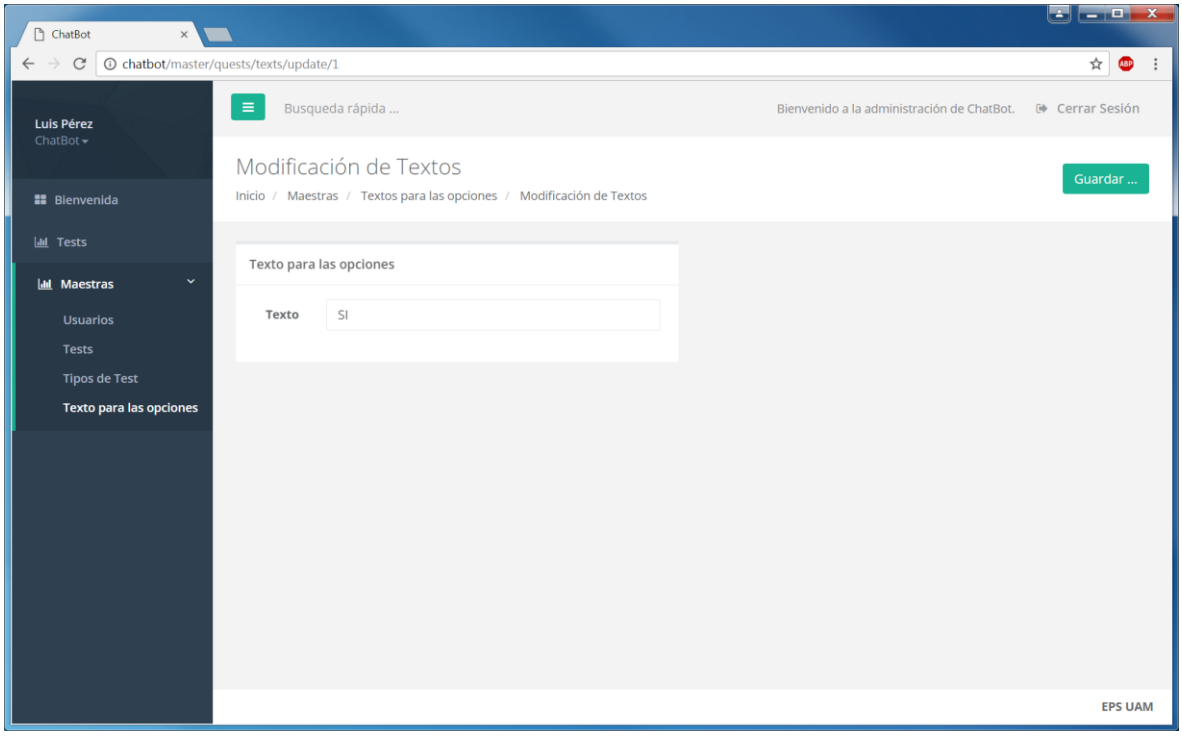

Figura 4-12: Pantalla Edición de Maestra Texto para opciones. Fuente: propia.

4.1.3.4 Test

La pantalla "Test" se ha creado sobre un solo fichero, se ha utilizado un condicional de Twig sobre la plantilla, si el controlador no recibe un tipo de test la vista con un selector de tipos, **s**ino con un panel de gestión de los test.

En la ***Figura 4-13*** se muestra la pantalla de selección de tipo, contiene un selector asociado a un formulario. Es necesario escoger un tipo para poder pasar a la edición.

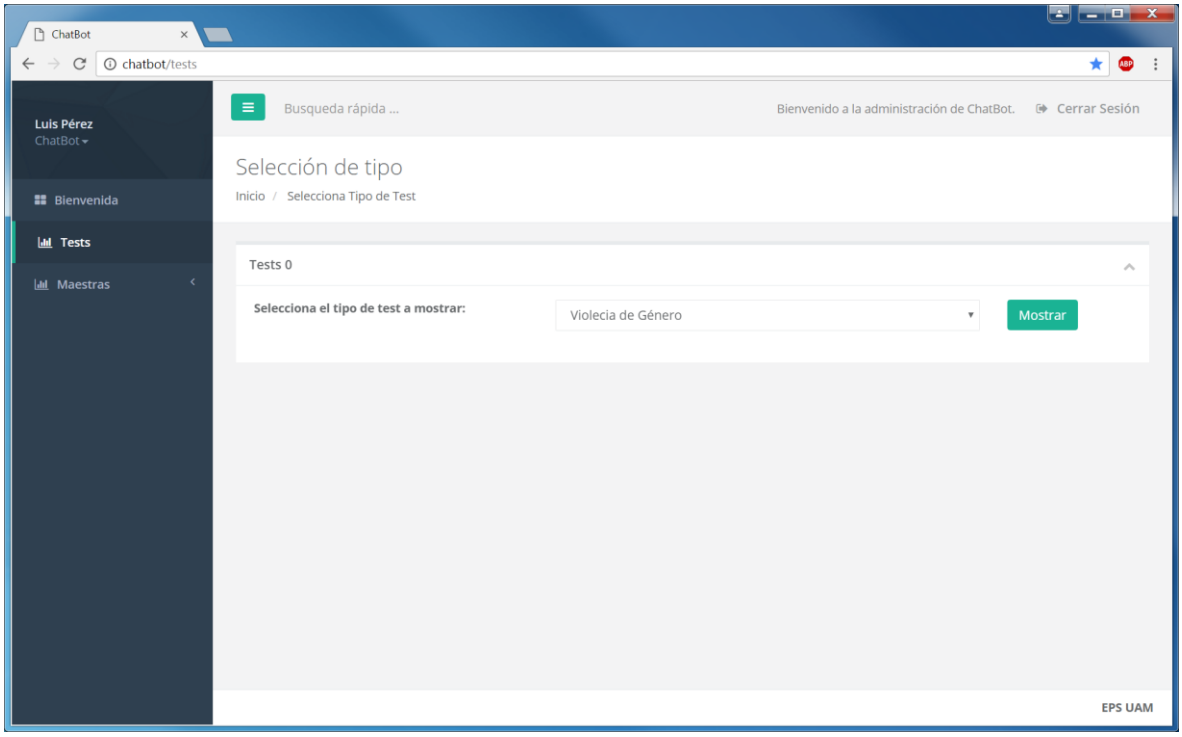

Figura 4-13: Pantalla selección de tipo. Fuente: propia.

La pantalla de gestión de Test corresponde a la ***Figura 4-14***. La parte superior la forman un contador con el número de test existentes para el tipo seleccionado, seguido de una lista con elementos seleccionables con el nombre de los test y un botón que redirecciona a la edición de maestras de test.

En la parte media nos encontramos un formulario que permite la edición de el nombre, la descripción y si se trata del test activo (el que se sincronizará con la aplicación). El botón para lanzar el formulario se encuentra en la parte superior.

En la parte inferior nos encontramos con un botón que permite crear nuevas preguntas y una tabla con todas las preguntas relacionadas. La tabla, al igual que todas las demás presentes en la aplicación, está generada con "Datatable".

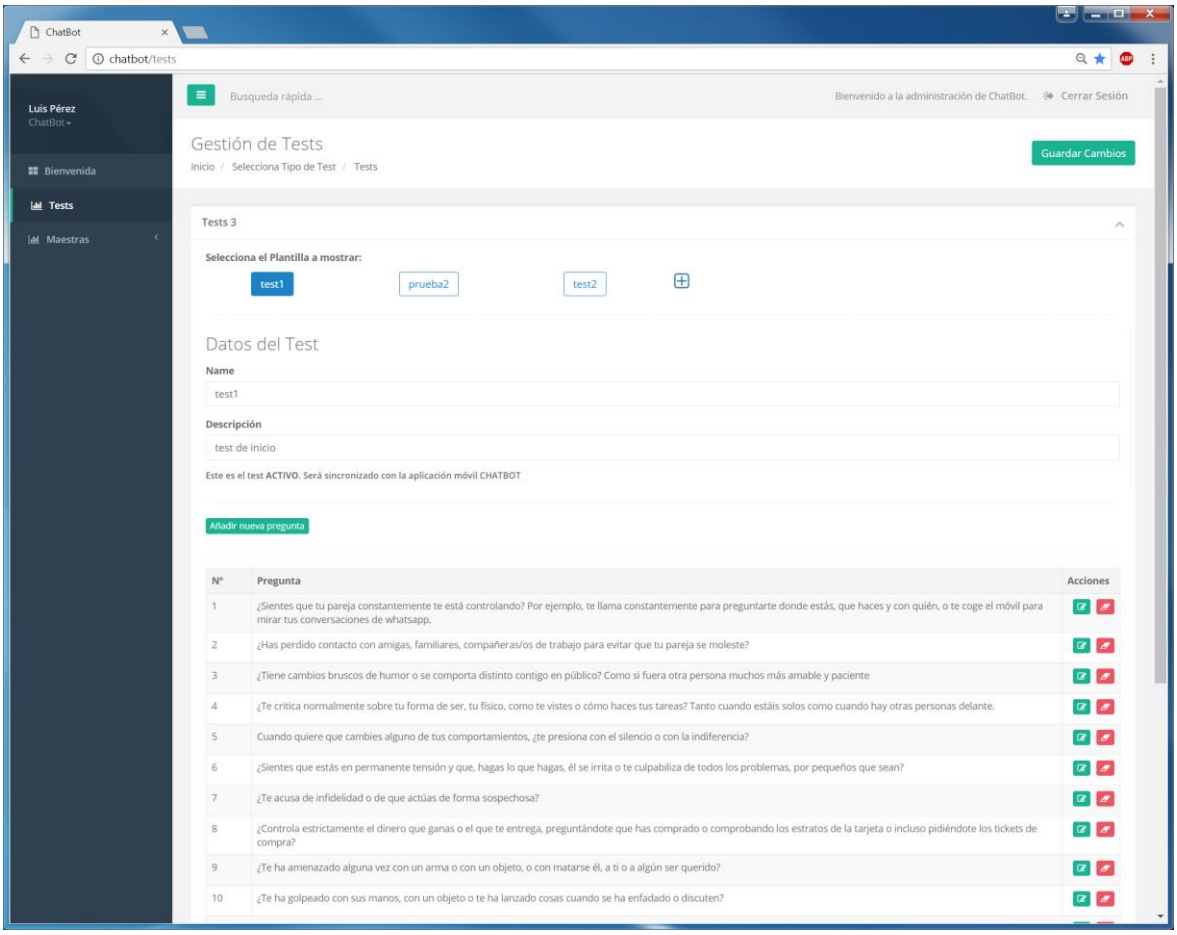

Figura 4-14: Pantalla Gestión de Test. Fuente: propia.

4.1.3.5 Quest

A Quest accederemos al pulsar sobre "Añadir nueva pregunta" o sobre la acción editar de la tabla. La ***Figura 4-15*** contiene la pantalla de Edición de preguntas en sus versión editar, en modo creación solo se muestra el primer panel sin opciones, una vez guardado se recarga la página y se muestran las opciones.

La vista está formada por un formulario completo, pero éste se muestra en dos paneles independientes, el botón guardar cambios almacenará todo lo contenido en la vista. La parte superior está formada por dos entradas para los atributos de la pregunta y la parte inferior contienen un panel para la creación y edición de opciones.

Este editor está configurado para que muestre solamente el número de opciones vinculadas a la pregunta, al pulsar sobre el botón "Añadir opción", se generará un nuevo cuadro con una opción vacía de forma dinámica, para ello se utilizando una función JS que modifica los atributos de elementos ocultos para hacerlos visibles.

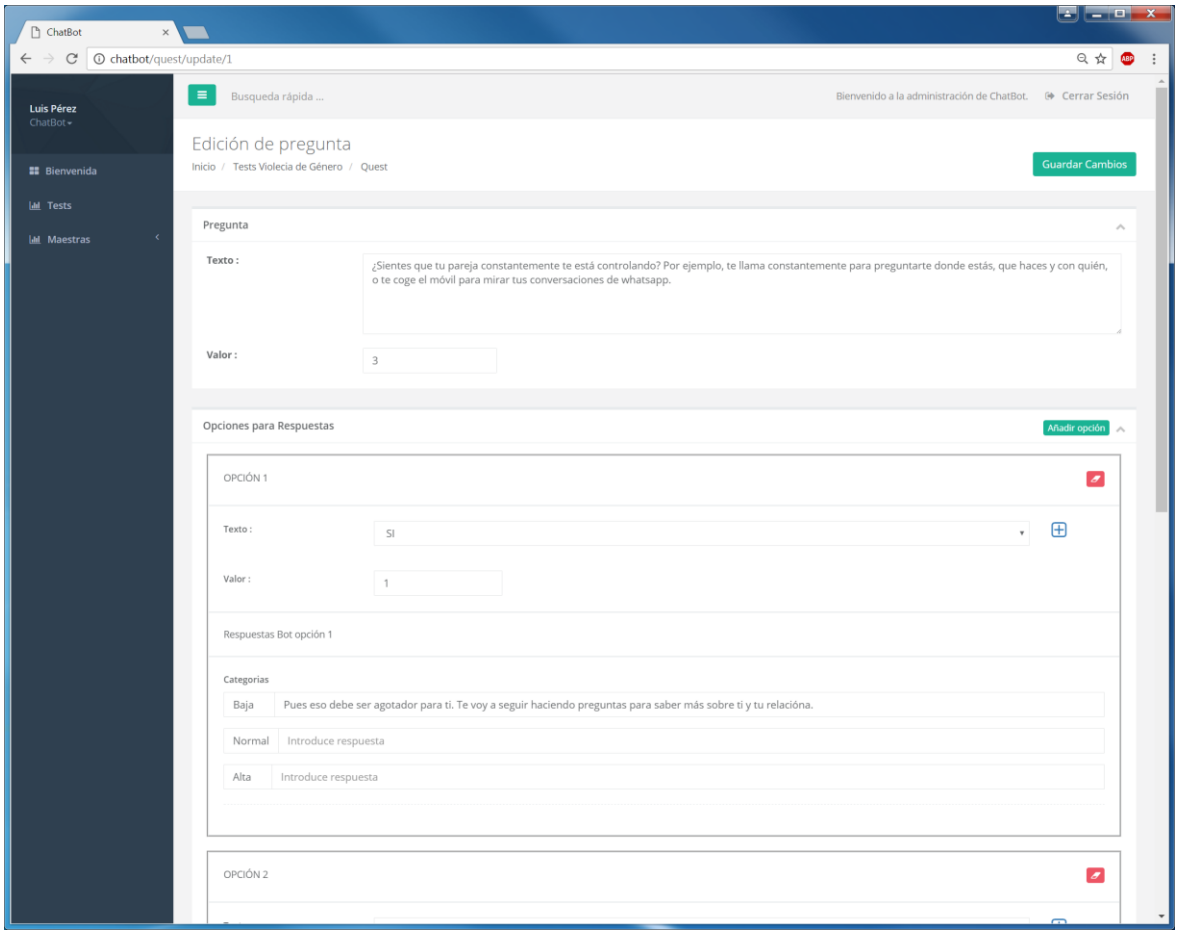

Figura 4-15: Pantalla edición de pregunta. Fuente: propia.

4.1.4 Modelos y Configuración del ORM.

Los modelos son necesarios para la utilización del ORM, ya que estos representan la clase con la que se extraerán los elementos de la base de datos.

Para el proyecto se han generado los ficheros de la ***Figura 4-16***, cada uno de ellos representa a una tabla de la base de datos y extienden de la clase "Model" de FuelPHP que proporciona los métodos para realizar consultas sobre la base de datos.

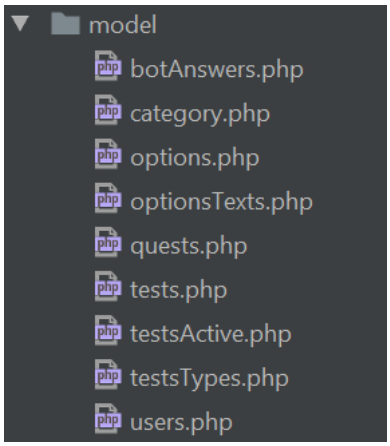

Figura 4-16: Modelos generados para el proyecto. Fuente: propia.

Para explicar la estructura de los modelos y mostrar el código desarrollado, la ***Figura 4-17*** contiene al fichero options.php. Se explica solo la clase "Model_Options", porque todas contienen la misma estructura y este fichero es el más extenso.

```php
<?php

use Orm\Model;




class Model_Options extends Model{

    protected static $_table_name = 'options';

    protected static $_primary_key = array('id');

    protected static $_belongs_to = array(
        'quest' => array(
            'key_from' => 'quest_id',
            'key_to' => 'id',
            'model_to' => 'Model_Quests',
            'cascade_delete' => false
        ),

        'text' => array(
            'key_from' => 'text_id',
            'key_to' => 'id',
            'model_to' => 'Model_OptionsTexts',
            'cascade_delete' => false
        ),

    );

    protected static $_has_many = array(
        'answers' => array(
            'key_from' => 'id',
            'key_to' => 'option_id',
            'model_to' => 'Model_BotAnswers',
            'cascade_delete' => false
        ),

    );



}
```

Figura 4-17: Fichero modelo Options. Fuente: propia.

Como se muestra en la imagen anterior, para definir un modelo de necesario instanciar varias variables pertenecientes la clase de la que extiende:

- **$_tabla_name**: Nombre real de la tabla en la base de datos. Con este parámetro, la clase quedará completamente vinculada a nuestra tabla, no será necesario especificar los atributos de la tabla, el ORM los detectará de forma automática.

- **$_primary_key**: Clave primaria de la tabla. Es necesario definirla para que el módulo pueda trabajar con la tabla.

- **$_belong_to**: Representa las relaciones 1-1 de la tabla. Se definen a partir de un array indexado, donde cada clave del array indica una propiedad de la relación.

- **$_has_many**: Sirve para declarar las relaciones 1-N. Se declara con la misma estructura que la variable anterior.

Para conectar el ORM con nuestra base de datos, el framework contiene dos carpetas en su directorio de configuración "development" y "production". Los dos ficheros contienen la misma estructura, se utilizará uno u otro dependiendo del modo de ejecución de la aplicación. En la ***Figura 4-18*** se muestra el fichero que se utiliza en modo desarrollo.

```php
<?php

return array(
    'default' => array(
        'connection'  => array(
            'dsn'        => 'mysql:host=localhost;dbname=chatbot',
            'username'   => 'root',
            'password'   => '123456'
        ),
    ),
);
```

Figura 4-18: Fichero configuración ORM. Fuente: propia.

4.1.5 JavaScript

En este proyecto JS junto con JQuery se ha utilizado para desempeñar funciones como hacer que se envíe un formulario pulsando un botón externo a dicho formulario o modificar atributos de los componentes de la vista de forma dinámica.

Por otro lado, Bootstrap también utiliza JS para realizar acciones de forma automática y dinámica. Utilizando el atributo "class" sobre una etiqueta HTML se pueden crear animaciones como la de la ocultación del menú lateral o que el menú pueda expandirse y contraerse mostrando subcategorías. También es utilizado para realizar todas las utilidades que ofrece "Datatable", pero estas deben de ser configuradas en un script.

4.1.6 Algoritmos destacables

Se ha intentado desarrollar el mayor número de funciones que faciliten extender las funcionalidades de la aplicación, ofreciendo herramientas que permitan modificar o añadir código de una forma sencilla.

La ***Figura 4-19*** muestra la función utilizada para completar las cadenas JSON que después son enviadas a la aplicación móvil. El método recibe dos argumentos, un objeto que contengan todos los elementos que se desean sincronizar y un JSON transformado en array indexado que será completado con los valores del objeto. Adicionalmente el segundo parámetro viene dentro de un array, esto se ha realizado así por motivos de diseño.

En el primer paso se obtiene la estructura que será rellenada con los datos pasados, esta variable sirve como plantilla que duplicaremos en cada iteración de elementos. El método contiene dos bucles anidados, el externo recorre los elementos (preguntas, resultados, opciones, etc), el interno recorre las propiedades de cada objeto individual obteniendo en cada iteración el valor y la clave de indexado.

Al tratarse de objetos con relaciones, muchos de estos contienen relaciones a otros objetos de la tabla, estas relaciones también tienen que ser expresadas en el JSON como se puede observar en el ***sección 4.1.2.6***. Cuando se detecta una relación de este tipo en el objeto, se realiza una llamada recursiva utilizando como parámetros el contenido de la relación y el fragmento JSON correspondiente.

Tanto el valor devuelto por la recursividad, como el valor de elemento (en el caso de que no se trate de una relación), se almacena el valor en la clave correspondiente dentro de la variable "$array_element". Esta variable se insertará en el segundo parámetro tantas veces como elementos contenga el primero.

Al final lo que obtenemos es un objeto que contiene indexado como queríamos todos los valores requeridos por el JSON. Esta variable al ser codificada con la función "json_encode" se transformará en la cadena con la información que se desea sincronizar.

```php
public function setQueryInArrayJson($elements ,$arrayJson)
{
    $array_aux = array_pop($arrayJson);
    foreach ($elements as $element)
    {
        $array_element = clone $array_aux;

        //LOOP OPTIONS KEY => VALUE
        foreach ($element as $key => $value)
        {

            if(is_array($value))
            {
                //EXTRAC THE EMPTY OBJECT
                $foreing_element_array = array(array_pop($array_element->$key));
                $array_element->$key = $this->setQueryInArrayJson($value,$foreing_element_array);
                continue;
            }

            //CHECK IF KEY IS SET IN ARRAY JSON, IF IT IS TRUE, THE VALUE WILL BE SAVED IN JSON
            if(isset($array_aux->$key))
            {
                $array_element->$key = $value;
            }
        }
        $arrayJson[]= $array_element;
    }
    return $arrayJson;

}
```

Figura 4-19: Función setQueryInArrayJson de fichero webservices/test.php. Fuente: propia.

4.2 Aplicación Móvil

4.2.1 Introducción

En este bloque del documento se muestra el desarrollo realizado en la aplicación móvil. Se describirán los ficheros principales: controladores, vistas, modelos y módulos.

Actualmente solo se ha desarrollado la versión para Android, pero gracias a Xamarin, gran parte del código es reutilizable para generar versiones para otras plataformas.

4.2.2 Controladores

Los controladores están creados sobre *Activities* de Android, las cuales nos permiten cargar vistas, realizar acciones sobre ellas y gestionar la información que introduce el usuario.

A continuación se describen los ficheros creados para esta aplicación.

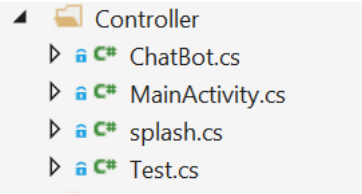

Figura 4-20: Controladores creados para la aplicación móvil. Fuente: propia.

4.2.2.1 Splash.cs

En esta actividad se produce la carga de la aplicación, se muestra una animación de carga, mientras se intenta producir la sincronización con el servidor para obtener el cuestionario. El código de la clase se puede ver en la ***Figura 4-21.***

En el primer método se realiza la carga inicial, durante este proceso se establece el contenido de la vista, se igualan las variables de la clase con los elementos de la pantalla y por último se anima la imagen.

Durante la ejecución de la actividad se llama a la función "OnResume", en este método se ejecuta una tarea que nos permita controlar mejor los tiempos de la aplicación sin perjudicar al hilo principal. En esta tarea primero se intenta obtener un usuario de la base de datos, si existe se registra la sesión, si no existe se crea uno nuevo. En ambos caso la información será enviada al servidor.

El siguiente paso durante la ejecución, será realizar la actualización de la información de la base de datos. Cada vez que se completa una etapa se introduce en un campo informativo de la pantalla en estado de la carga. Si no se tuviera conexión a internet, se trabajaría con los datos por defecto.

Por último se lanza la actividad MainActivity y se finaliza la actual.

```
using System;
using Android.App;
using Android.Content;
using Android.OS;
using Android.Views;
using Android.Widget;
using Android.Util;
using System.Threading.Tasks;
using System.IO;
using Android.Graphics;
using System.Threading;
using chatbot;
using ORM.Models;

namespace chatBot.Controller
{
    [Activity(MainLauncher = true)]
    public class SplashActivity : App
    {
        private AnimationView animate;
        public TextView textinfo;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            SetContentView(Resource.Layout.splash);
            animate = FindViewById<AnimationView>(Resource.Id.imgAnimate);
            Animate();
            textinfo = FindViewById<TextView>(Resource.Id.textInfo);
        }

        protected override void OnResume()
        {
            Task.Run(() => {
                try
                {
                    User user = CurrentTest.Model.Get<User>();
                    if (user != null)
                    {
                        user.sessions = user.sessions + "," + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                        CurrentTest.Model.Update(user);
                    }
                    else
                    {
                        user = new User();
                        user.created_at = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                        user.device_identifiquer = Build.Serial;
                        user.sessions = user.created_at;
                        CurrentTest.Model.Insert(user);
                    }

                    CustomWebClient customWebClient = new CustomWebClient(this, CurrentTest.Model, "test");

                    CurrentTest.User = user;

                    customWebClient.SendUserInfo(user);

                    RunOnUiThread(() => {
                        textinfo.Text = "Preparando datos...";
                    });

                    customWebClient.masterSync();
                    Thread.Sleep(500);

                    RunOnUiThread(() => {
                        textinfo.Text = customWebClient.output;
                    });

                    Thread.Sleep(1000);

                    RunOnUiThread(() => {
                        StartActivity(new Intent(Application.Context, typeof(MainActivity)));
                        Finish();
                    });
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message + "; Line => " + e.StackTrace);
                }

            });
            base.OnResume();
        }

        private async void Animate()...
    }

    public class AnimationView ...

}
```

Figura 4-21: controlador splash.cs aplicación móvil. Fuente: propia.

4.2.2.2 MainActivity.cs

Esta clase gestiona la pantalla donde se oculta la aplicación, si detecta que se ha iniciado por primera vez, envía al usuario directamente al controlador ChatBot, sino mostrará una serie de noticias a modo de distracción.

En la pantalla de notición también se encontrará una entrada texto con un botón, para introducir una contraseña. Adicionalmente se ha creado un registrador de eventos "Click" sobre el botón, donde si se introduce la contraseña correcta se iniciará el controlador ChatBot. La contraseña tiene que haber sido previamente configurada una primera ejecución de la aplicación.

4.2.2.3 ChatBot.cs

ChatBot se encarga de la presentación de la aplicación y el registro inicial. Toda la actividad se realiza sobre una misma vista, que se instancia en la función de carga al igual que el resto de variables que se utilizarán a lo largo de la clase.

En este controlador no utilizamos el método "OnResume" ya que el usuario va navegando por la pantalla a través de un botón. Este controlador muestra hasta diez pantallas distintas sobre la misma vista, para lograrlo se ha implementado un "Switch" que función a como un selector de estados, dependiendo en qué estado te encuentres se mostrará una pantalla con unas funcionalidades.

Esta gestión por estados, no permite volver a la misma pantalla en la que nos encontramos cuando abandonamos la aplicación. Cada estado activa y desactiva sus propiedades. Debido a que el código es muy largo, solo se muestran un par de casos del conmutador en la ***Figura 4-22***.

El código mostrado pertenece al estado cinco y seis de la función "BtnNext_Click" del controlador.

Durante el caso cinco se valida el nombre introducido, se almacena en la base de datos y en las clase Preferences (sección 4.2.5), se desactivan los elementos de la anterior pantalla y se activan los necesarios para recoger la edad. En caso de cualquier error se repetiría el estado.

En el caso seis, se valida la edad introducida, se almacena en la tabla de usuarios, se hacen invisibles los elementos de este estado y se preparan los del estado siguiente, que será la pantalla para registrar los datos sobre la pareja.

Una vez acabado todo el cuestionario de registro, se ofrece la posibilidad de pasar al test o salir de la aplicación, si elige salir, se cerrará la aplicación, si se escoge continuar se llamará a la siguiente actividad y se dará esta por finalizada.

```
case 5:
    //VALID OF THE USERNAME
    if (editText.Text != "" || username_saved!="")
    {
        warning.Text = "";
        //Remove Special Characters
        editText.Text = RemoveSpecialCharacters(editText.Text);

        //2º VALIDATION OF THE USERNAME
        if (editText.Text.Length == 0 && username_saved == "")
        {
            //REPEAT STATE
            setWarning();
        }
        else
        {
            //CHECK IF NOT STORE USER IN PREFERENCES
            if (username_saved == "")
            {
                Preferences.setValueUser(this, editText.Text);
                CurrentTest.User.name = editText.Text;
                CurrentTest.Model.Update(CurrentTest.User);
            }

            /*********DISABLE LAST STATE**********/
            //HIDE KEYBOARD
            inputMethodManager.HideSoftInputFromWindow(editText.WindowToken, 0);
            //HIDE INPUT, MEDIUM TEXT AND BUTTON NEXT
            editText.Visibility = ViewStates.Invisible;
            mediumContentText.Visibility = ViewStates.Invisible;


            /*********ACTIVE NEXT STATE**********/

            //SET LARGE CONTENT TEXT AND DISABLE TEXT MANAGER
            isActiveAutoTexts = false;
            largeContentText.Text = GetString(title) + " " + editText.Text + " " + GetString(body);

            //ACTIVE BUTTONS
            age.Visibility       = ViewStates.Visible;
            btnsYesNo.Visibility = ViewStates.Visible;
            btnNo.Visibility     = ViewStates.Gone;
            btnYes.Visibility    = ViewStates.Gone;

            //REMOVE EDIT TEXT
            editText.Text = "";
        }
    }
    else
    {
        //REPEAT STATE
        setWarning();
        return;
    }
    break;
case 6:

    /********************** GET AGE **********************/
    warning.Text = "";

    if (age.Value == 0)
    {
        //REPEAT STATE
        setWarning("*Por favor, introduzca una edad mayor que 0.");
        return;
    }

    CurrentTest.User.age = age.Value;
    CurrentTest.Model.Update(CurrentTest.User);

    /********************** SELECTOR MAN OR WOMAN **********************/

    /*********DISABLE LAST STATE**********/

    //HIDE CONTAINER BUTTONS YES/NO
    btnsYesNo.Visibility = ViewStates.Gone;
    exitButtonContent.Visibility = ViewStates.Gone;

    /*********ACTIVE NEW STATE**********/
    //ACTIVE CONTAINER BUTTONS YES/NO
    mediumContentText.Visibility = ViewStates.Visible;
    btnNext.Visibility           = ViewStates.Visible;
    genderSelector.Visibility    = ViewStates.Visible;
    editText.Visibility          = ViewStates.Visible;
    selectorContent.Visibility   = ViewStates.Visible;

    break;
```

Figura 4-22: Código del caso 5 y 6 de la función BtnNext_Click de ChatBot.cs. Fuente: propia.

4.2.2.4 Test.cs

Test es la última actividad de nuestra aplicación, está diseñada bajo la misma estructura que el resto. La clase va mostrando una a una las preguntas de test con las posibles opciones planteadas.

El controlador está diseñado para que sea independiente al número de preguntas que contiene, éste las irá mostrando hasta completar todas, una vez terminado obtiene el resultado de la base de datos y se muestra al usuario.

El código de la ***Figura 4-23*** se ejecuta cada vez que se pulsa el botón "Siguiente" en la pantalla, esta función distingue entre si tiene que mostrar una nueva pregunta o almacenar la opción seleccionada y mostrar la respuesta relacionada con la opción.

```
private void NextButton_Click(object sender, EventArgs e)
{
    if (newQuest)
    {
        //IF ANSWERED ALL QUESTIONS SHOW FINAL CONCLUSION
        if(questionNumber == quests.Count())
        {
            Preferences.setValueTestNumber(this,test_number+1);
            //SET FINAL BOT TEXT
            largeContentText.SetText(Resource.String.final_bot);
            //FIRST TIME NOT SHOW TEST RESULT
            if(showTestResult)
            {
                results = CurrentTest.Model.GetWhere<Results>((d) => (d.min <= testScore && d.max >= testScore));
                largeContentText.Text = results.ToArray()[0].desc;
            }
            showTestResult = true;

        }
        else
        {
            SetLayoutTexts();
            newQuest = false;
        }

    }
    else
    {
        RadioButton optionSelected = FindViewById<RadioButton>(options.CheckedRadioButtonId);
        if (optionSelected != null)
        {
            //SAVE USER ANSWER
            UserAnswers answer = new UserAnswers();
            answer.quest_id = currentQuest.id;
            answer.test_number = test_number;
            answer.option_selected_id = questOptions[(int)optionSelected.Tag].id;
            answer.value = questOptions[(int)optionSelected.Tag].value;
            CurrentTest.Model.Insert(answer);

            //INCREMENT TESTSCORE
            testScore += questOptions[(int)optionSelected.Tag].value;

            //SET ANSWER BOT
            SetAnswerBotTexts(questOptions[(int)optionSelected.Tag].id);

            //PREPARE THE NEXT SCREEN
            questionNumber++;
            options.ClearCheck();
            warningText.Text = "";
            newQuest = true;

        }
        else
        {
            warningText.Text = "*Porfavor, selecciona una opción";
        }
    }

}
```

Figura 4-23: Función NextButton_Click del fichero Test.cs. Fuente: propia.

4.2.3 Vistas

En este bloque se explicarán las vistas que se han implementado. Las capturas que se muestran pertenecen al prototipo inicial de la aplicación.

En Xamarin las vistas funcionan prácticamente igual que en Android, la única diferencia es que la extensión varía de XML a AXML. Esta última extensión es la que utiliza el framework para las interfaces nativas de Android.

Las vistas que se muestran en la ***Figura 4-24*** son las utilizadas por los controladores. Con estos cuatro ficheros se crearán todas las pantallas.

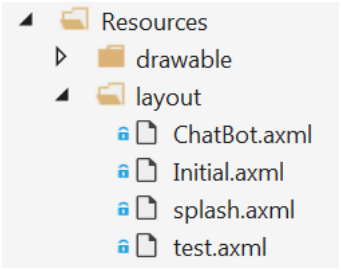

Figura 4-24: Ficheros generados para las vistas de Aplicación Móvil. Fuente: propia.

4.2.3.1 Splash.axml

Esta vista está compuesta por una animación de carga y un texto descriptivo del proceso. La ***Figura 4-25*** contiene una captura de esta pantalla inicial.

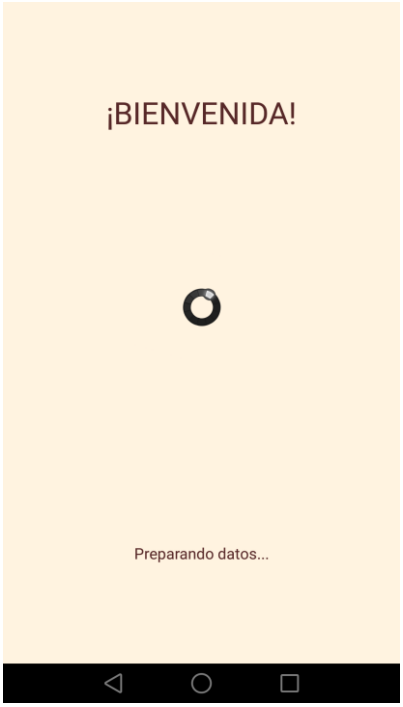

Figura 4-25: Pantalla inicial Aplicación Móvil. Fuente: propia.

4.2.3.2 ChatBot.axml

La primera vez que la aplicación cargue nos encontraremos con la asistenta virtual, ella será la encargada de guiarnos a través de la aplicación. En las pantallas iniciales nos explica quién es ella y de que trata el test que realiza. Se muestra como ejemplo la pantalla de presentación en la siguiente imagen, ***Figura 4-26***.

Figura 4-26: Pantalla de presentación de aplicación Móvil. Fuente: propia.

En las siguientes pantallas la asistente virtual solicita que se introduzca una contraseña, explica claramente para que sirve y donde habrá que introducirla. Después da las explicaciones necesarias para entender cómo se maneja la aplicación y las características especiales que tiene, como por ejemplo, salir rápido de la aplicación pulsando sobre su imagen en cualquier momento de los test.

Una vez explicado el funcionamiento, la asistenta comienza a solicitar datos personales a la usuaria para poder dar un trato más personalizado y poder clasificarla mejor dentro de grupos de riesgo. A continuación se muestran dos de estas pantallas, las que no aparecen en este documento, preguntan por: edad, estado actual con la pareja o expareja y sobre si tiene hijos.

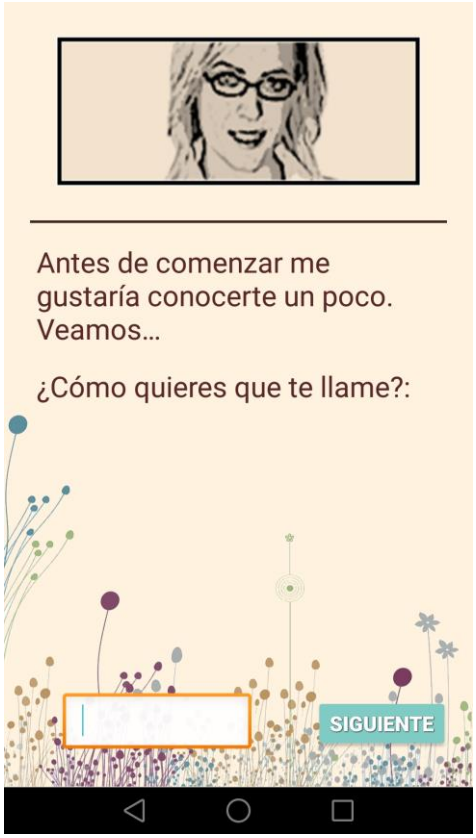

Figura 4-27: Pantalla registro de nombre Aplicación Móvil. Fuente: propia.

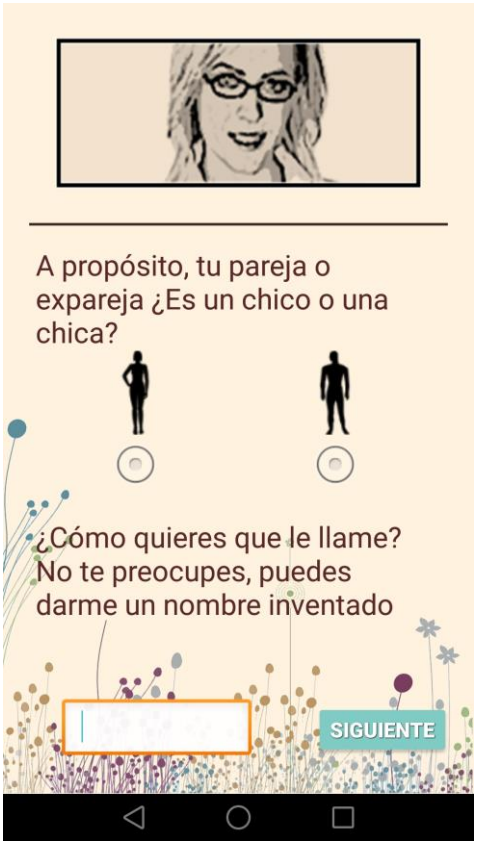

Figura 4-28: Pantalla registro de datos pareja Aplicación Móvil. Fuente: propia

En las siguientes pantallas se dan consejos para realizar el test y para que la persona se sienta segura utilizando la aplicación. Después se da la opción de abandonar la aplicación o continuar en ella para realizar el test.

Si se decide salir o en cualquier momento se minimiza, la imagen que se muestra en el registro de aplicaciones de Android, no será diferenciable, como se enseña en la ***Figura 4-29***. Para proteger al máximo la intimidad de la usuaria, la ocultación sucederá en todas las pantallas menos en la de noticias. Si se vuelve a lanzar la aplicación, se accede a este apartado de noticias.

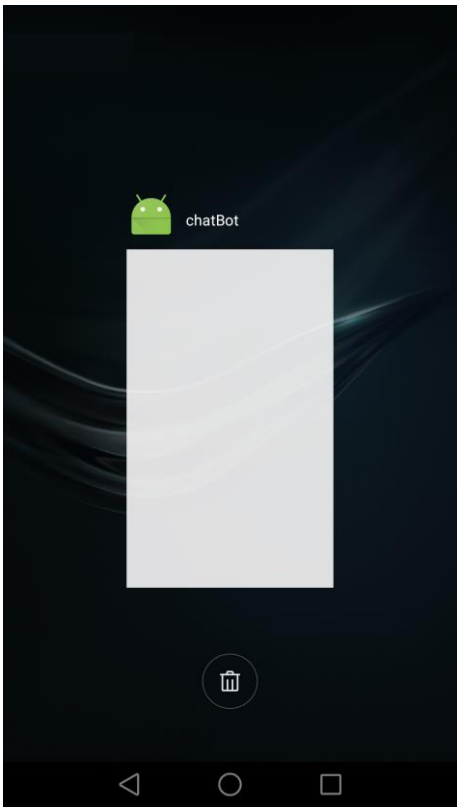

Figura 4-29: Aplicación Móvil oculta en el registro de actividades de Android. Fuente: propia.

4.2.3.3 Intial.axml

Esta pantalla se mostrará una vez completado el cuestionario inicial y a partir de la segunda ejecución de la aplicación. Su funcionalidad es única y exclusivamente ocultar la aplicación a usuarios ajenos.

Para esconder el verdadero contenido de la aplicación, esta pantalla simula una simple aplicación de noticias. En la parte inferior de la pantalla hay un cuadro de entrada donde se deberá introducir la contraseña para acceder al contenido real. La ***Figura 4-30*** muestra la el final de la pantalla, donde se encuentra el falso formulario.

Si se introduce un correo válido, salta un mensaje como si se hubiera realizado la suscripción correctamente, si se introduce cualquier texto que no sea la contraseña, aparece un mensaje de correo no válido. Al introducir la contraseña correctamente, se pasará la pantalla de test.

Figura 4-30: Pantalla noticias, ocultación de aplicación móvil. Fuente: propia.

4.2.3.4 Test.axml

Al test se puede acceder desde el registro inicial o desde las noticias. Un vez que entremos en los test, la aplicación continuará hasta que se finalice, si salimos, al volver entrar se retomara desde el último punto donde lo dejaras.

Las siguientes imágenes muestran una de las preguntas y la respuesta que da la asistenta, en este caso, después de haber contestado "Sí" a la pregunta.

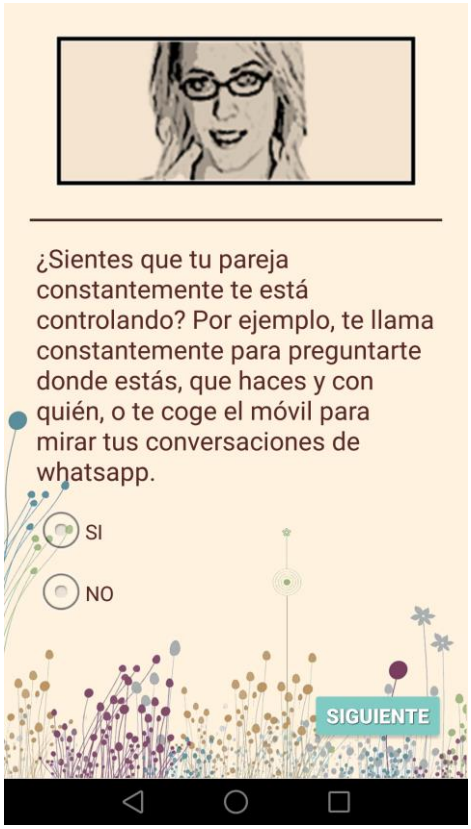

Figura 4-31: Ejemplo de pregunta del test, Aplicación Móvil. Fuente: propia.

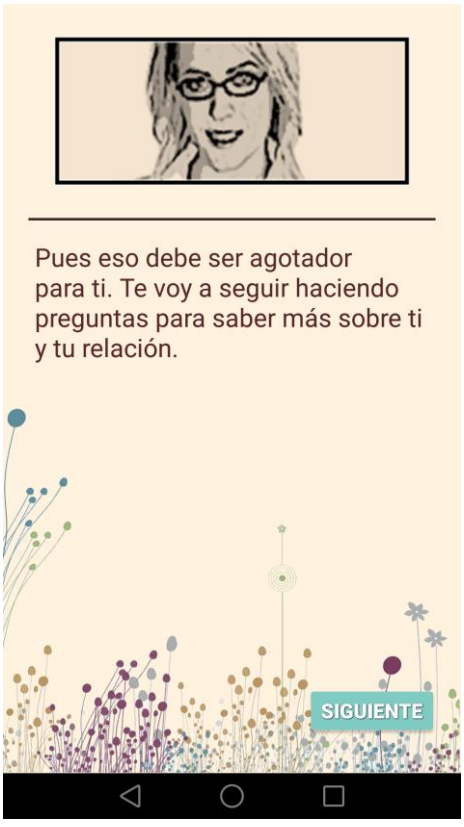

Figura 4-32: Ejemplo de respuesta de la asistenta durante el test, Aplicación Móvil. Fuente: propia.

Una vez contestado a todos los test, se muestra el resultado del test y consejos de ayuda que varían según el nivel de riesgo obtenido, estos resultados son definidos por los psicólogos en la aplicación Web.

4.2.4 Modelos

Los modelos tienen la misma funcionalidad que en FuelPHP. Xamarin no trae un ORM para el manejo de la base datos, pero si tiene muchos paquetes para gestionar estas clases.

En nuestro caso hemos utilizado herramientas que son descargables desde el administrador de paquetes NuGet o instaladas de forma externa. Los tres ficheros que aparecen fuera de la carpeta "Models" se utilizan para realizar las conexiones sobre la base de datos.

Los ficheros que se encuentran dentro de "Models" representan a cada tabla existente en la base de datos de la aplicación. Igual que en otros ORM sirven como traductores de los registros de la tabla a objeto.

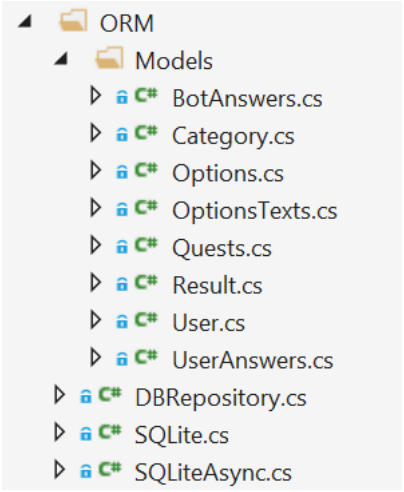

Figura 4-33: Ficheros creados para los modelos de la base de datos de la aplicación móvil. Fuente: propia.

"SQLlite.cs" contiene todas las funciones de bajo nivel necesarias para conectarse a la base de datos y realizar consultas. En "DBRepository.cs", utilizando la librería anterior, se han desarrollado funciones más avanzadas, proporcionando métodos para realizar consultas de selección con predicado, consultas de inserción o copias de seguridad de la base de datos.

En la ***Figura 4-34*** se pueden observar el fichero "BotAnswer.cs" perteneciente a los modelos de la aplicación. Como se observa, consta de una estructura muy simple, ni si quiera implementa relaciones entre tablas. Los demás modelos serán muy similares a este.

```
using SQLite;

namespace ORM.Models
{
    [Table("bot_answers")]
    public class BotAnswers
    {
        [PrimaryKey, AutoIncrement, Column("id")]
        public int id { get; set; }

        [Column("option_id")]
        public int option_id { get; set; }

        [Column("text")]
        public string text { get; set; }

        [Column("category_id")]
        public int category_id { get; set; }

    }


}
```

Figura 4-34: Fichero BotAnswer.cs perteneciente a la carpeta "Models" de la aplicación móvil. Fuente: propia

4.2.5 Módulos

Los módulos se generan para poder reutilizar la funcionalidad que proporcionan a lo largo de toda la herramienta.

En este proyecto hemos creado los módulos que se observan en la siguiente imagen. En este bloque daremos una pequeña explicación de para qué sirve cada uno.

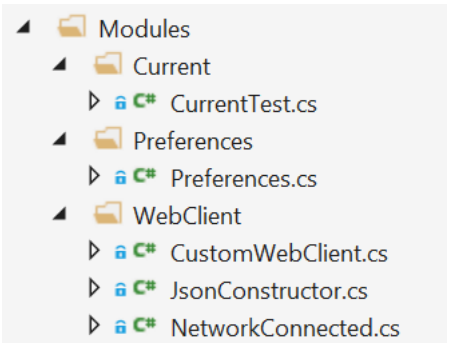

Figura 4-35: Carpeta Modules de la aplicación Móvil. Fuente: propia.

CurrentTest.cs:

Esta clase estática está constituida únicamente por dos variables, se instancia al inicio de la aplicación y se reutiliza a lo en todos los controladores. Su función principal es proveer a las demás clases de la variable "Model" que pertenece a "DBRepository", que se utilizará para realizar consultas sobre la base de datos, y "User" que representa al usuario y contendrá todos sus datos.

```
using ORM;
using ORM.Models;

namespace chatBot
{
    public static class CurrentTest
    {
        public static DBRepository Model;
        public static User User;
    }
}
```

Figura 4-36: Fichero CurrentTest.cs Aplicación Móvil. Fuente: propia.

Preferences.cs:

Esta clase se ha creado utilizando la librería Android.Preferences que permite almacenar información en el dispositivo de forma sencilla y persistente, aunque se cierre la aplicación no perderemos la información almacenada en estas variables.

En la aplicación se utiliza para comprobar si es la primera vez que se inicia la aplicación y para guardar pequeños registros, a los que recurriremos para no acceder a la base de datos.

NetworkConnected.cs:

Módulo muy simple que solo implementa una función la cual devuelve un valor booleano dependiendo del estado de la conexión.

CustomWebClient.cs:

Mediante está clase se realizarán todas las tareas de comunicación con el servidor, por este motivo será el fichero que más se detalle. A continuación se muestran algunas de las funciones más importantes.

En la ***Figura 4-37*** se observan la función de construcción "CustomWebClient" que es instancia las principales variables y comprueba la conexión del móvil utilizando el módulo NetworkConnected. En la variable output registra el valor de salida, que refleja el estado actual del conector.

La otra función importante que también se ve en la imagen es “MasterRequest”, este método realiza la petición al servidor para que le envíe una cadena en formato JSON con el nuevo contenido de la base de datos. Como se observa, la petición se realiza de forma síncrona, por lo tanto habrá que esperar a la respuesta del servidor para continuar con la ejecución de la aplicación. Si la respuesta no se recibe correctamente o se agota el tiempo de espera, la función devolverá error y no se actualizarán los datos. Si la respuesta se recibe de forma correcta, se llama a la función “UpdateMasters” pasando como parámetro el contenido de la respuesta, ésta transforma el JSON y actualiza la base de datos.

```
/************************************************************
 * Sincronizacion.
 *
 * Este modulo sicroniza automaticamente el test
 *
 *
 *
 ************************************************************/
public CustomWebClient(Context c, DBRepository m)
{
    _context = c;
    Model = m;
    wcData = new WebClient();

    if (!NetworkHelper.NetworkConnected(c))
    {
        is_ready= false ;
        output = "No se detecta conexión a internet, la aplicación se puede utilizar normalmente ";
    }
    else
    {
        is_ready = true;
        output = "Conexion preparada";
    }


}


public void masterSync()
{
    //INICIAMOS LA SYNC
    MastersRequest();
}

public void MastersRequest()
{
    try
    {
        var request = HttpWebRequest.Create("http://192.168.0.161:8080/chatbot/public/webservices/test");
        request.ContentType = "application/json";
        request.Method = "GET";
        request.Timeout = 10000;
        using (HttpWebResponse response = request.GetResponse() as HttpWebResponse)
        {
            if (response.StatusCode != HttpStatusCode.OK)
                System.Console.Out.WriteLine("Error fetching data. Server returned status code: {0}", response.StatusCode);
            using (StreamReader reader = new StreamReader(response.GetResponseStream()))
            {
                var content = reader.ReadToEnd();
                if (string.IsNullOrWhiteSpace(content))
                {
                    System.Console.Out.WriteLine("Response contained empty body...");
                }
                else
                {
                    UpdateMasters(content);
                    System.Console.Out.WriteLine("Response Body: \r\n {0}", content);
                }
            }
        }
        output = "Sincronización correcta";


    }
    catch (Exception e)
    {
        System.Console.WriteLine(e.Message);
        output = "No se pudo concectar al sevidor";

    }

}
```

Figura 4-37: Constructor y función MasterRequest del fichero CustomWebClient.cs Fuente: propia.

Además de recibir información como se ha mostrado en la función anterior, este módulo también contiene métodos para enviar información del usuario, la forma de comunicación será la misma, pero en el sentido inverso.

Para realizar estos envíos de datos se ha creado la función "SendUserInfo". El funcionamiento del método es el siguiente, inicializa una variable con la cadena que va a enviar al servidor, se van comprobando las variables del usuario, si ya está generadas se introduce el valor en su clave correspondiente. Una vez construido la cadena de texto con los valores, se llama a la función "SendJson".

La función "SendJson" utiliza la clase "Webclient" de la librería System.Net y llama al método "UploadValues". Esta última función envía una petición HTTP con método POST al servidor, con un formulario que se construye en la función "PrepareContent" y que contiene la cadena a sincronizar.

Tanto "SendUserInfo" como "SendJson" se puede consultar en la ***Figura 4-38***.

```
public void SendUserInfo(User user, bool test = false)
{
    string jsonUser = "{\"device_identifiquer\":\"#device_identifiquer#\",\"name\":\"#name#\",\"age\":\"#age#\","+
                      "\"options\":\"#options#\",\"partner_name\":\"#partner_name#\",\"partner_gender\":\"#partner_gender#\","+
                      "\"created_at\":\"#created_at#\",\"sessions\":\"#sessions#\", userAnswers}";

    if (user.age > 0)
    {
        jsonUser = jsonUser.Replace("#age#", user.age.ToString());
    }
    if (user.device_identifiquer != null)
    {
        jsonUser = jsonUser.Replace("#device_identifiquer#", user.device_identifiquer);
    }
    if (user.name != null)
    {
        jsonUser = jsonUser.Replace("#name#", user.name);
    }
    if (user.options != null)
    {
        jsonUser = jsonUser.Replace("#options#", user.options);
    }
    if (user.partner_name != null)
    {
        jsonUser = jsonUser.Replace("#partner_name#", user.partner_name);
    }
    if (user.partner_gender != null)
    {
        jsonUser = jsonUser.Replace("#partner_gender#", user.partner_gender);
    }
    if (user.created_at != null)
    {
        jsonUser = jsonUser.Replace("#created_at#", user.created_at);
    }
    if (user.sessions != null)
    {
        jsonUser = jsonUser.Replace("#sessions#", user.sessions);
    }

    if(test)
    {
        string usersAnswersJson = "";
        TableQuery<UserAnswers> userAnswers = Model.GetWhere<UserAnswers>((d) => (d.test_number == Preferences.getValueTestNumber(_context)));
        foreach(UserAnswers userAnswer in userAnswers )
        {
            usersAnswersJson += (usersAnswersJson != "") ? "," : "";
            usersAnswersJson += "{\"test_id\":\"" + Preferences.getTestIdKey() + "\",\"option_selected_id\":\"" + userAnswer.option_selected_id  +
                                "\",\"test_number\":\"" + userAnswer.test_number+ "\",\"value\":\"" + userAnswer.value +"\"}";
        }

        usersAnswersJson += "\"userAnswers\":["+ usersAnswersJson + "]";
        jsonUser = jsonUser.Replace("userAnswers", usersAnswersJson);
    }
    else
    {
        jsonUser = jsonUser.Replace(", userAnswers", "");
    }

    SendJson(jsonUser);
}

public void SendJson(string content, string url = "chatbot/public/webservices/test", string location = "http://192.168.0.161:8080/")
{

    NameValueCollection dataBytes = PrepareContent(content);

    try
    {
        Task.Run(() =>
        {
            wcData.UploadValues(new Uri(location + url), dataBytes);
        });
    }
    catch (Exception e)
    {

        System.Console.WriteLine(e.Message);
    }

    System.Console.WriteLine("data upload is complete." + content);

}
```

Figura 4-38: Funciones SendUserInfo y SemdJson pertenecientes al fichero WebClient.cs. Fuente: propia.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

La idea inicial de este proyecto surge de una presentación del ICFS que realizó un amigo psicólogo, en principio se quería pasar a plataforma digital unos test orientados a evaluar el riesgo de maltrato que puede estar sufriendo una mujer.

Durante la primera reunión para analizar cuáles eran las necesidades reales que implicaban el proyecto, se dio a conocer que el instituto poseía muchos otros test de categorías distintas que podrían aprovecharse del desarrollo de esta aplicación.

Con la tecnología actual y los medios de los que disponíamos, nos dimos cuenta de que la idea podía ir más lejos. El desarrollo móvil inicial podría ser reutilizado como plantilla para aplicaciones futuras sobre otros tipos de test. Además se podría proporcionar una herramienta con la que mantener todos los test actualizados.

Por estos motivos decidimos ampliar el planteamiento original y crear una aplicación web que pudiera cubrir las dos necesidades principales:

- Mantener la aplicación móvil actualizada obteniendo *feedback* de los usuarios y pudiendo modificar los test.

- Aunar todos los tipos de test en una misma aplicación web, para que posteriormente pueda cubrir la primera necesidad al resto de aplicaciones móviles que se generen de este tipo.

Con el objetivo de que la aplicación llegara al mayor número de personas posibles, se buscaron las tecnologías más adecuadas para todas las situaciones que planteaba el proyecto.

Por esto último, se decidió usar herramientas de desarrollo que permitieran la reutilización de código entre las plataformas móviles existentes. También se decidió usar un sistema de comunicación a través de peticiones web, así una sola versión web proveería a todas las plataformas.

Como conclusión, finalmente después de llevar a cabo todo el desarrollo, se han obtenido dos sistemas intercomunicados que permiten la realización de test de evaluación de riegos, la gestión de la información recibida sobre los resultados y el manejo de la información contenida en los test.

Adicionalmente, se han diseñado las herramientas pensando en el resto de aplicaciones que podrán aprovecharse de este proyecto. La parte web ya está preparada actualmente para desarrollar test para otras aplicaciones y el código de la parte móvil modularizado para facilitar el desarrollo de una aplicación base.

Lo que inicialmente era un proyecto que solo implicaba una categoría de preguntas en una aplicación, finalmente tiene el potencial para llegar a ofrecer una gran variedad de categorías en distintas aplicaciones.

5.2 Trabajo futuro

5.2.1 Aplicación Web

La aplicación web ya puede proveer a distintas aplicaciones móviles con los tipos de test que tenga generados, relacionando el tipo de test con la aplicación y enviándole así la información correspondiente a cada una.

Pero este servicio solo se extiende a los móviles y teniendo ya desarrollado un API que provee los test, se plantea crear una web donde se puedan realizar estos test desde cualquier navegador.

Adicionalmente, también habrá que realizar el mantenimiento de la aplicación y el servidor, manteniéndolos actualizados y añadiendo nuevas funcionalidades a la herramienta que vayan surgiendo.

5.2.2 Aplicación Móvil

Este apartado se divide en dos, debido a que existen dos frentes bastante destacados en el desarrollo futuro de esta aplicación

5.2.2.1 Cross-Plataform

Actualmente el código solo se ha desarrollado en Xamarin.Android, pero como ya hemos comentado anteriormente, este framework nos permite la portabilidad del código a otros sistemas.

Según cálculos de la empresa Xamarin, el 75% del código generado con su framework es reutilizable en las demás plataformas.

Por lo tanto, a la hora de portar la aplicación a iOS o a Windows Phone, se podrá reutilizar la mayor parte del código lógico generado.

Para las interfaces habría dos posibilidades, diseñar una interfaz para cada dispositivo o utilizar la herramientas de interfaces que ofrece Xamarin.Forms, que permite desarrollar una interfaz válida para todas las plataformas.

Queda por tanto pendiente la portabilidad del código a otros sistemas y la decisión sobre el cómo generar las interfaces.

5.2.2.2 Aplicación Base

La mejor opción para poder generar aplicaciones tipo, es crear una aplicación plantilla, que contenga las funcionalidades básicas necesarias y las vistas preparadas para que apenas tengan que sufrir modificaciones.

De esta forma partimos de un desarrollo inicial muy avanzado, que nos facilita generar nuevas aplicaciones mucho más rápido.

Además, gracias a los módulos implementados, el sistema de comunicación de las futuras aplicaciones apenas tendrá que sufrir cambios.

Como objetivo futuro cercano, se pretende reutilizar el código de la aplicación actual para generar una aplicación base que facilite el desarrollo de futuras aplicaciones.

Referencias

[1] Fragmento extraído de *https://es.wikipedia.org/wiki/Framework*.

[2] Información obtenida en https://store.unity.com/es/products/unity-personal.

[3] Dato obtenido de *https://en.wikipedia.org/wiki/Appcelerator_Titanium*. *Julie Bort (1 de febrero de 2013). "Microsoft podría comprar una startup que Powers 10 por ciento de los teléfonos inteligentes del mundo" . Business Insider . Consultado el 11 de julio de 2013.*

[4] Frases extraídas de https://laravel.com/ (*12 de enero de 2017*)

[5] Dato obtenido de https://es.wikipedia.org/wiki/SQLite#cite_note-2.

Glosario

Activity	Clase de Android utilizada para la gestión de las pantallas de la aplicación.
Bootstrap	Framework para la creación de páginas y aplicaciones web.
Composer	Manejador de dependencias para PHP, esto lo que permite es poder controlar mejor las versiones y mantener siempre actualizado a la versión requerida. Se trata de un simple fichero en formato JSON que contiene la información necesaria de las versiones que queremos tener de cada módulo utilizado en nuestro proyecto. Sin duda facilita el control de versiones para poder realizar la instalación en diferentes plataformas o versiones de PHP.
Cross-Plataform	También llamado multiplataforma o software independiente de la plataforma. Se utiliza para desarrollar un único código ejecutable o compilable para varias plataformas.
CRUD	*Create, Read, Update and Delete*. Expresión utilizada en la programación web, se utiliza para referirse a un módulo que cumpla las funciones de listar, registrar, actualizar y eliminar un registro de la base de datos.
CSV	*comma-separated values.* Documento de formato abierto que se utiliza para representar datos en tablas.
JS y jQuery	JS (JavaScript) lenguaje de programación scripting. jQuery es un framework que extiende las librerías DOM de JS para hacerlas más sencillas de utilizar.
ORM	*Object-Relational Mapping,* Son un conjunto de librerías de funciones que suelen tener implementados todos los frameworks, Su principal utilidad es la de realizar consultas sobre la base de datos y extraer los registros transformados en objetos de PHP. Pueden ser compartidos por varios frameworks o creados exclusivamente para uno en concreto.
MCV	*Model-Controller-View*. Modelo vista controlador.
Migas de Pan	Palabra procedente del término inglés ***breadcrumb.*** Es una técnica de navegación utilizada en las aplicaciones. Se basa en una línea de texto que indica el recorrido seguido en la aplicación y permite regresar a vistas anteriores.