UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DESARROLLO EN ANDROID DE UN SISTEMA DE RESPUESTA INMEDIATA CON CLICKERS PARA USO EDUCATIVO

Mario Antonio Polo Daza
Tutora: Estrella Pulido Cañabate
Junio 2017

DESARROLLO EN ANDROID DE UN SISTEMA DE RESPUESTA INMEDIATA CON CLICKERS PARA USO EDUCATIVO

AUTOR: Mario Antonio Polo Daza
TUTORA: Estrella Pulido Cañabate

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2017

Resumen

Con la irrupción de los *smartphones* y la tecnología táctil se cambió la percepción del mundo, hoy día podemos desde comprar una entrada de cine hasta jugar a cortar fruta tan solo gracias a nuestros dedos y nuestro móvil. Por ello, el objetivo de este Trabajo de Fin de Grado será aplicar esa utilidad táctil con su facilidad de uso y rápida respuesta en el **mundo educativo**.

Es una realidad que en numerosas ocasiones los alumnos no participan en las clases por temor a cometer fallos, con lo que la labor del docente se complica. Con esta aplicación permitiremos eliminar esa barrera.

Podremos complementar las clases teóricas con **preguntas** que elaborará el **profesor** gracias a una herramienta web, en la cual se clasificará por asignatura o grupo y se filtrará por tema. La web se ha desarrollado con las tecnologías más punteras, como son un servidor **Node.js** y el *framework* **Angular.js**.

Una vez la pregunta sea lanzada al servidor Node, el alumno se conectará a la aplicación **Android** y de nuevo filtrará el contenido hasta llegar a la cuestión a contestar. Finalmente el **estudiante** responde desde la intimidad y su **respuesta** se almacena en la base de datos **MySQL**. Más tarde, dichos datos servirán para modelar los **gráficos estadísticos** en la web de los profesores.

El alumno será mucho más partícipe en las clases, además así la docencia se hará más amena, interactiva y entretenida. Por otro lado, el profesor podrá llevar un seguimiento de grano grueso por asignatura o grupo, y más de grano fino consultando las estadísticas de cada alumno en concreto, lo que le permitirá reforzar a aquellos que lo necesiten, o incluso montar grupos de apoyo con los más adelantados.

En definitiva con esta aplicación mejoraremos ambos frentes de la enseñanza, facilitando la labor de los docentes y llevando un seguimiento por alumno más personal.

Este TFG comprende por tanto el desarrollo de 2 aplicaciones (web y android), llevando a cabo el modelo en cascada para su implementación. Mediante pruebas reales en clase se ha comprobado la robustez y se garantiza el cumplimiento de los requisitos funcionales de la aplicación.

Palabras clave

Android, XML, Node.js, Express.js, NPM, Angular.js, ECMAScript, MySQL, framework, Aplicación web, Modelo en cascada, Gráficos estadísticos, clickers

Abstract

The break-in of smartphones and touchscreen technology has changed how we interact with the world. Nowadays, we can buy cinema tickets or even play cutting fruit using our smartphone with just our fingers. The statement of this thesis explains how we can apply the usage and rapid response of the touchscreen technology to the educational world.

It is a reality that students fear making mistakes, and because of this, participation in class decreases. This causes the teaching process to get harder. With this app we will delete this barrier and solve the problem for both, teachers and students.

We will be able with this app to support and upgrade the quality of theorical learning with questions made by the professor with the help of a web application. This web application will classify questions by subject, chapter, and also by student groups. This tool has been developed with the most advanced technologies such as Node.js server and Angular.js framework.

The tool works as follows. Once the question is uploaded or launched to the Node server, the student will connect to the Android app and will filter the content until he finds the question he really wants to know about. Finally, the student answers and his/her response will be stored in MySQL database. Later on, this data is used to build and modulate the graphs located in the teacher's private web.

With this tool, the student's participation will increase and will help improve the teaching and learning experience, making it richer in content and more entertaining. On the other hand, the teacher will be able to follow-up the subject or group of students from a broad point of view. And more exhaustively follow-up consulting statistical records from each of the students. This results in an easier and faster process to reinforce individual students or creating student groups with detailed needs. This needs can be either for the most advanced students or for students falling behind.

In conclusion, the learning process and experience will improve, making it easier for teachers and students to teach and be taught.

This thesis comprises the development of two apps (web and Android) carrying out the waterfall model of its implementation. Through real tests inside theory classes we have proved and we ensure the compliance of the app's functional requirements.

Keywords

Android, XML, Node.js, Express.js, NPM, Angular.js, ECMAScript, MySQL, framework, Web application, Waterfall Model, Statistics Charts, clickers

Agradecimientos

A mi tutora Estrella Pulido Cañabate, por su apoyo y colaboración en el trabajo desarrollado.

A mis compañeros de estudio, por todo lo aprendido en la universidad con ellos.

A mis compañeros de trabajo por su ayuda para mejorar la implementación del proyecto.

A mi familia y en especial a mi pareja por su esfuerzo y empuje cuando más lo necesitaba.

Mario Antonio Polo Daza
Junio 2017

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE FRAGMENTOS

1. Introducción

En los siguientes apartados se detallarán la motivación de este Trabajo de Fin de Grado, así como el alcance del proyecto, los objetivos del mismo y la organización de la memoria.

Motivación

A día de hoy la docencia poco a poco ha ido beneficiándose de los grandes avances de la tecnología. Aún así, en la mayoría de aulas se nos hace común ver la presencia de la pizarra clásica en la que los profesores se apoyan para impartir conocimiento. Con este Trabajo de Fin de Grado buscamos impulsar el uso del *smartphone* para aprender en clase.

Mediante el uso de clickers [1] ya se comenzó a realizar el proceso de aprendizaje de una manera más interactiva, lo cual motiva al alumno a participar. Estos dispositivos permiten al estudiante contestar las cuestiones que se planteen de una forma rápida y sencilla.

Con el tiempo se han desarrollado aplicaciones web enfocadas en esta materia, explotando el uso del móvil y su potencial. Sin embargo estas herramientas tienen limitaciones, como por ejemplo la necesidad de usar periféricos extra (proyectores, clickers, ordenadores) mientras se lleva a cabo la prueba, por ello la principal motivación de implementación de este Trabajo de Fin de Grado es mejorar y complementar estas herramientas.

Desligando la creación y publicación de preguntas, permitiendo incluso realizar las pruebas de manera totalmente remota, conseguimos una comunión completa entre docencia y análisis de resultados.

Los resultados estadísticos asociados a las respuestas dadas por los alumnos, son otro de los fundamentos de este trabajo. La capacidad de poder visualizar de manera fácil y gráfica el progreso de los estudiantes, permite al docente reforzar las áreas con más dudas e incluso dedicar más tiempo a los alumnos con más dificultades.

Por todo ello se ha llevado a cabo el desarrollo de ClickEPS, una aplicación desarrollada en uno de los lenguajes más extendidos de la tecnología móvil como es Android, con una interfaz simple e intuitiva; y una aplicación web implementada complementaria con uno de los *frameworks* más punteros como es Angular.js.

Objetivos

El objetivo de este Trabajo de Fin de Grado es el desarrollo en Android de un sistema de respuesta inmediata para uso educativo. El ámbito en el que se utilice podría ser cualquier aula de docencia, aunque en el caso que nos atañe, las pruebas se han llevado a cabo en clases de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

Es por ello que se definen los siguientes objetivos:

- Acceso mediante autenticación, con un email de usuario y contraseña. La creación y gestión de usuarios queda fuera del alcance de la aplicación. Además el acceso a la interfaz Android queda restringido a los alumnos, mientras que la aplicación web solo permite el acceso del profesorado.
- Crear un modelo de datos que permita gestionar los usuarios, las preguntas y respuestas asociadas a cada asignatura mediante una base de datos relacional.
- Desarrollar una interfaz Android para consultar la base de datos de preguntas, que permita filtrarlas por asignatura y tema, y finalmente almacene las respuestas de los estudiantes.
- Recibir un *feedback* instantáneo al responder las preguntas, es decir, dar la opción al estudiante de consultar la pregunta recién contestada (para que en caso de error se aprenda del fallo). Esto conlleva permitir consultar todas las preguntas del tema estén o no contestadas.
- Restringir el acceso a las preguntas en función de las necesidades del profesor mediante una fecha de activación de la pregunta, y una fecha de expiración de la misma.
- Desarrollar una interfaz web para consultar la base de datos que permita visualización de estadísticas así como tratamiento de preguntas en el sistema. La gestión de asignaturas y grupos queda fuera del alcance de la aplicación.
- Acompañar los datos estadísticos con gráficos intuitivos que permitan evaluar el rendimiento ya sea por grupo de estudiantes o individualmente.
- Permitir crear preguntas de tipo test, con un enunciado y 4 respuestas posibles, marcando la respuesta correcta, y fijando las fechas ya mencionadas (de activación y expiración). Permitir editar, borrar o duplicar preguntas del sistema.

Organización de la memoria

La memoria sigue los patrones en base a la normativa del Trabajo de Fin de Grado constando de los siguientes capítulos:

- **Capítulo 2 (Estado del Arte)**: En este capítulo analizaremos las aplicaciones que ya existen para facilitar la docencia online, evaluaremos sus virtudes y sus defectos y mostraremos una comparativa de cara a extraer conclusiones para el desarrollo de nuestra aplicación. Finalmente expondremos los puntos fuertes de nuestro desarrollo en comparación con lo que ya existe.
- **Capítulo 3 (Tecnologías y Frameworks)**: En este apartado explicaremos brevemente cada una de las tecnologías que se usarán en nuestra implementación. Además definiremos los *frameworks* y librerías en las que nos hemos apoyado.
- **Capítulo 4 (Diseño)**: En la fase de diseño detallaremos el ciclo de vida escogido, acotaremos el alcance de nuestra aplicación y expondremos los requisitos funcionales y no funcionales. Explicaremos los casos de uso y mostraremos los diagramas entidad-relación de la base de datos.
- **Capítulo 5 (Desarrollo)**: En este bloque explicaremos las dos aplicaciones implementadas, detallando los apartados más relevantes en ambas y cómo se ha llevado a cabo la implementación.
- **Capítulo 6 (Integración, pruebas y resultados)**: En el capítulo de integración y pruebas hablaremos de cómo se integran ambas aplicaciones, así como las pruebas que se han realizado tanto en el ámbito de desarrollo como en el de producción.
- **Capítulo 7 (Conclusiones y trabajo futuro)**: Finalmente en el último apartado resumiremos la aplicación desarrollada y propondremos futuros desarrollos para la mejora y mantenimiento de la misma.
- **Glosario, referencias y anexos**: Los últimos 3 bloques finalizan con el glosario, en el que mencionaremos los principales tecnicismos. El apartado de referencias, dónde detallaremos las fuentes que se han consultado en el desarrollo. Y los anexos, en los cuáles encontraremos la guía de uso, las pruebas de usabilidad llevadas a cabo en clase, adjuntaremos fragmentos de código, mostraremos las maquetas presentadas en las primeras fases y veremos las capturas reales de cómo ha quedado la aplicación.

2. Estado del arte

En este capítulo realizaremos un estudio sobre las aplicaciones que ya existen y tratan de cubrir el mismo ámbito abarcado por nuestra aplicación, explicando sus características y detallando los aspectos positivos y negativos de cada una.

Soluciones Actuales del Problema

En la actualidad encontramos herramientas y aplicaciones para resolver problemas similares en el ámbito de la docencia. En concreto nos centraremos en: *Kahoot* y *Plickers*.

Kahoot

Kahoot [2] es una aplicación web cuya finalidad es aplicar técnicas de **gamificación** [3] en la enseñanza. En su interfaz podemos crear preguntas (o juegos), para más tarde responder desde cualquier dispositivo.

La web se presenta solo en inglés y aun se encuentra en fase beta. En la vista principal nos permitirá crear 4 tipos de interacciones con los alumnos: *Quiz*, *Jumble*, *Discussion* y *Survey*. Además nos muestra un resumen de estadísticas y resultados recientes. En nuestro caso nos centraremos en el modo *Quiz* (**Figura 1**).

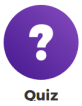


Figura 1 – Botón Quiz de Kahoot

Podremos crear nuestro propio grupo de preguntas (**Figura 2**), detallarlo mediante una descripción y añadirle parámetros de filtrado según visibilidad, idioma y audiencia (en nuestro caso universidad).

Figura 2 – Formulario de Creación de Grupo de Preguntas en Kahoot

A continuación podremos crear preguntas (**Figura 3**), marcando el tiempo límite de respuesta, y detallando enunciado y cuatro respuestas, con una única como correcta. Además podremos dar recompensas, dando más peso a unas preguntas que a otras.

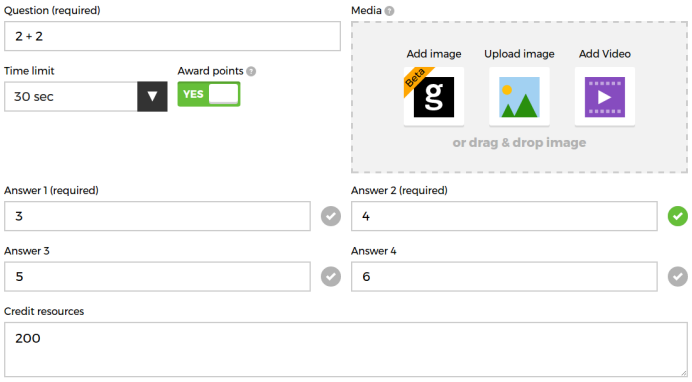


Figura 3 – Formulario de Creación de Preguntas en Kahoot

Finalmente podremos lanzar el test para que los alumnos contesten. Mediante un código PIN, los estudiantes buscan el test en el navegador de su dispositivo y acceden directamente a las preguntas, las cuales se muestran en la pantalla de clase (**Figura 4**), ya que a los alumnos solo les salen las opciones de respuesta en el *smartphone* (**Figura 5**).

Figura 4 – Respuestas de Kahoot en Proyector del Profesor

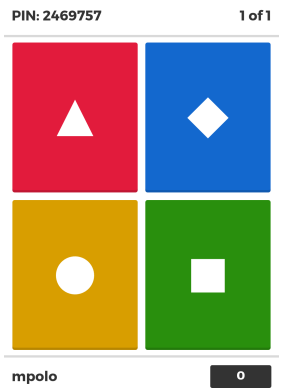


Figura 5 – Respuestas en Dispositivo Móvil del Alumno

En conclusión, *Kahoot* es una aplicación bastante completa que nos permite contestar preguntas mediante dispositivos móviles, en la que las limitaciones pueden ser que los estudiantes no tienen acceso a *posteriori* a los resultados, ni pueden ver las preguntas y sus respuestas en su propio dispositivo, dependiendo siempre de un proyector en clase.

Plickers

Plickers [4] es una aplicación web enfocada en la enseñanza online y la interacción con los alumnos. Esta aplicación no requiere el uso de ningún dispositivo móvil, ya que se basa en un código QR (*Quick Response*) contenido en una hoja impresa que le servirá al alumno para contestar las preguntas.

Esta aplicación también se nos muestra en inglés, y permite crear clases y preguntas desde su interfaz web (**Figura 6**).

Una vez creada la clase, creamos un grupo de preguntas para dicha clase. En este caso podremos crear pregunta con multirrespuesta (dando cuatro respuestas posibles) y del tipo Verdadero / Falso.

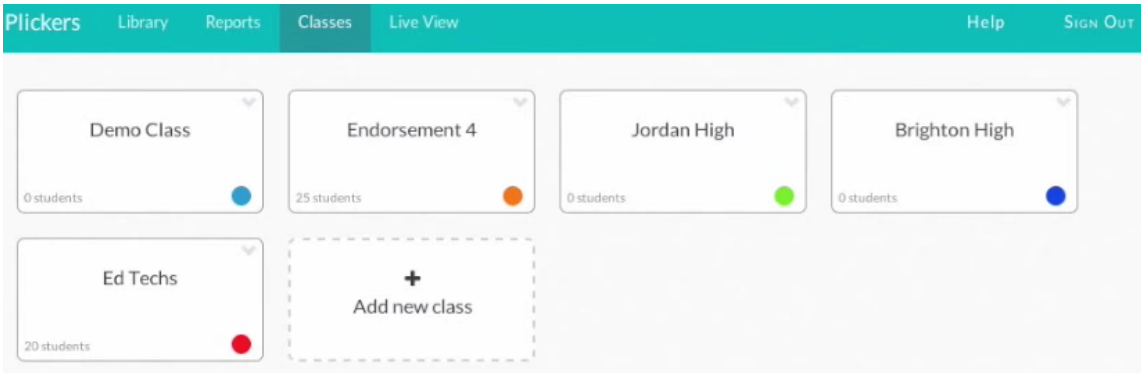


Figura 6 – Dashboard de Plickers con las Clases creadas

Además se pueden comprobar datos estadísticos y establecer fechas en la configuración de las preguntas (**Figura 7**).

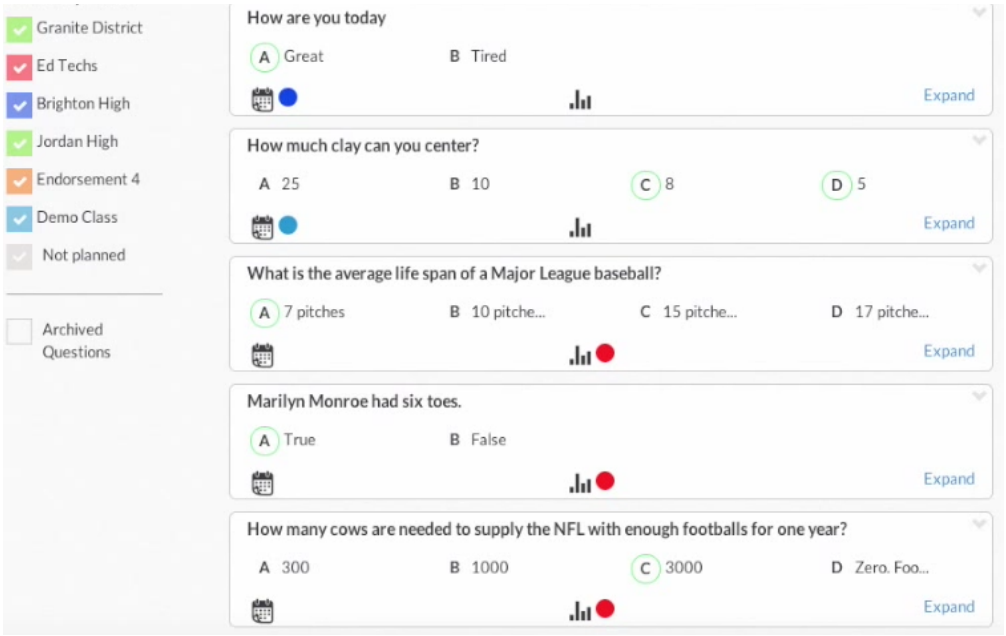


Figura 7 – Listado de Preguntas de Plickers

Ahora solo quedaría definir los alumnos de nuestra clase, y aportarles a cada uno los códigos QR para poder contestar las preguntas (**Figura 8**). Cada código QR contiene información sobre el alumno que contesta y la opción que quiere responder. En función de la posición en la que el QR es mostrado, el alumno contesta una opción u otra (siendo el lado superior el que marca la respuesta).

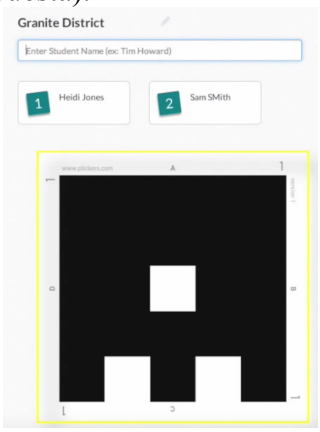


Figura 8 – Plantilla de Respuesta de Alumno

Finalmente al lanzar la pregunta y mostrarla por un proyector a los estudiantes, estos alzan la respuesta que ellos consideran correcta (**Figura 9**), y desde la aplicación móvil (Android o iOS) del profesor, se captan las respuestas.

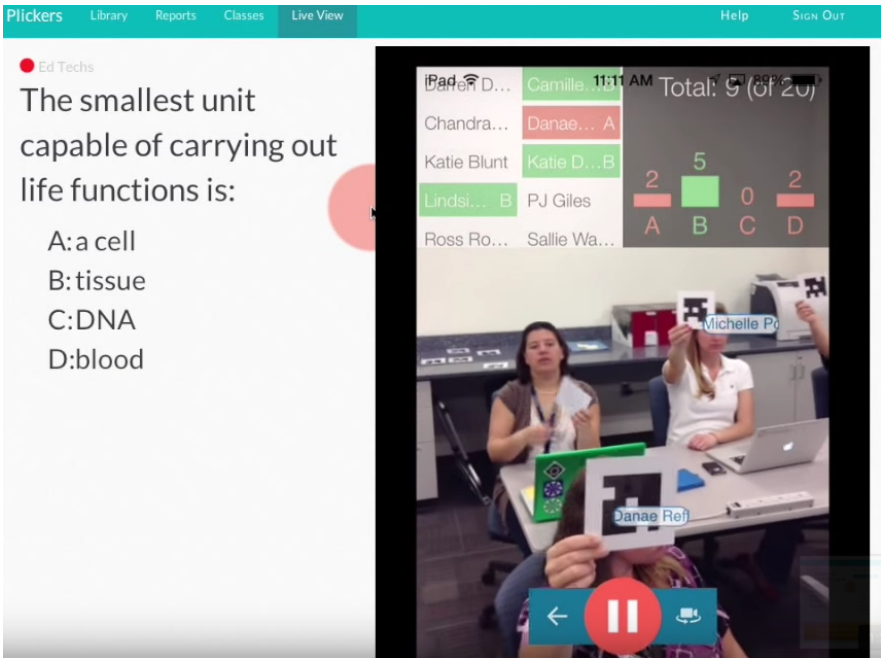


Figura 9 – Ejemplo real de captación de códigos QR con Plickers

Como conclusión destacamos que su uso resulta más interactivo, puede ser utilizada sin una conexión a internet por parte de los estudiantes, pero sin embargo el número de códigos QR para los alumnos es limitado.

De nuevo el alumno no puede consultar a *posteriori* resultados y preguntas antiguas, ya que la aplicación está más centrada en el profesor, el cual sí puede llevar el seguimiento de los estudiantes gracias a las estadísticas que presenta la web.

Comparativa

Las aplicaciones analizadas presentan unos objetivos muy similares a los de nuestra aplicación ClickEPS. A continuación presentamos una tabla comparativa de las aplicaciones analizadas:

	Kahoot!	plickers clickers, simplified.
Facilidad de Uso e Interacción	Fácil, aplicación simple y de rápido aprendizaje (tanto para profesores como estudiantes)	Fácil, interacción muy intuitiva mediante códigos QR. Interfaz web sencilla e intuitiva.
Tipo de Interacción	Clicks (en navegador web)	Códigos QR (en papel)
Variedad de Tipos de Preguntas	Presenta cuatro tipos de juegos, entre ellos el multirrespuesta (4)	Solo permite multirrespuesta (4) y preguntas del tipo Verdadero / Falso
Contiene Estadísticas	Sí, aunque bastante escuetas	Sí
Plataformas	Se maneja mediante navegador web, tanto el profesor como el alumno	Mediante interfaz web para definir las preguntas y con aplicación móvil (Android / iOS) para captar respuestas
***Feedback* al Estudiante**	Notifica del fallo o del acierto, pero no permite consultas de respuestas a pasado	El estudiante únicamente recibe información gracias al proyector en clase, ya que no interactúa con ningún dispositivo inteligente.

Tabla 1 – Comparativa de Aplicaciones Kahoot y Plickers

Una vez analizada la comparativa, hemos extraído los objetivos esenciales a cumplir por la aplicación a desarrollar, priorizando el uso de *clicks* y el *feedback* al estudiante.

De esta manera buscamos una aplicación igualmente usable y completa tanto a nivel del profesorado como a nivel del alumnado.

Es por esto que ClickEPS se distingue de estas aplicaciones mediante una aplicación Android muy usable que además de permitir al alumno responder preguntas, le aportará información de resultados, aciertos, errores y permitirá a los profesores fijar tiempos de activación y caducidad de las preguntas (haciéndola una aplicación independiente de la web en tiempo real).

Además la interfaz web permite crear, editar y eliminar preguntas según la necesidad del profesorado, clasificando por tema, asignatura y grupo. También ofrece mayor precisión en los resultados estadísticos pudiendo filtrar por pregunta o alumno.

3. Tecnologías y Frameworks

En este apartado realizaremos un análisis de las tecnologías y *frameworks* [5] que utilizaremos para la implementación de nuestra aplicación y expondremos las razones que nos han llevado a ello. Detallaremos los lenguajes utilizados en el desarrollo, el patrón de arquitectura MVC (Modelo-Vista-Controlador) y el uso de una base de datos relacional (MySQL).

Android

Android [6] es un sistema operativo de código abierto basado en núcleo **Linux** (**Figura 10**). Desarrollado por la empresa **Android Inc** que fue comprada en 2005 por **Google** e impulsó su implementación publicando la primera versión en el año 2007. A día de hoy existen más terminales vendidos con este sistema operativo que con los principales competidores juntos (Windows Phone e iOS).

Figura 10 – Logo de Android

El lenguaje utilizado para implementar es **Java** [7], uno de los lenguajes más extendidos a nivel mundial. En cuanto al soporte y versionado, Google garantiza un alto nivel de soporte para las **versiones** más recientes de Android (**Figura 11**), ello permite que existan todavía millones de usuarios con versiones más retrasadas pero en ningún momento obsoletas.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.9%
4.1.x	Jelly Bean	16	3.5%
4.2.x		17	5.1%
4.3		18	1.5%
4.4	KitKat	19	20.0%
5.0	Lollipop	21	9.0%
5.1		22	23.0%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	4.5%
7.1		25	0.4%

Figura 11 – Versiones de Android

En su implementación las vistas se definen en código **XML**, y su depuración y desarrollo se facilita enormemente gracias a la comunidad y a los **IDE** [8] (Integrated Development Environment), como Eclipse o **Android Studio**.

Node.js

Node.js [9] (**Figura 12**) es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del **servidor** basado en el lenguaje de programación **ECMAScript** [10] (especificación estándar de Javascript). Desarrollado en 2009, funciona sobre **un solo hilo de ejecución** usando entradas y salidas asíncronas, lo cual provoca que toda operación deba tener una función (de retorno).

Figura 12 – Logo de Node.js

Combinado con el uso de **Express.js** [11] (**Figura 13**) (*framework* de aplicaciones web de Node.js) permite desarrollar un **Backend** muy robusto y en un lenguaje muy afín al **Front-End** [12]. Con ello permite construir un **API REST** [13] para futuras consultas desde cualquier interfaz (web o android en nuestro caso).

Figura 13 – Logo de Express.js

Finalmente gracias a **NPM** [14] (**Figura 14**) (gestor de paquetes de Node.js), podemos descargarnos fácilmente todas las **dependencias** de nuestro proyecto llevando un control de versiones. Además también recibe un muy alto nivel de **soporte** y tiene detrás una **comunidad** inmensa de contribuidores.

Figura 14 – Logo de NPM

Angular.js

Angular.js [15] es un *framework* de **JavaScript** [16] (**Figura 15**) de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones *singlepage* (web de una sola página) basado en la arquitectura Modelo-Vista-Controlador. Orientado a la **programación imperativa** [17], resulta una herramienta muy completa para desarrollar la lógica de negocio de una aplicación web.

Figura 15 – Logo de Angular.js

Actualmente es uno de los *frameworks* más extendidos por todo el mundo, tanto es así que recientemente se ha impulsado el desarrollo de **Angular 2** [18] (publicado en el año 2016). Sin embargo cabe destacar que Angular 2 es un nuevo *framework* que poco tiene que ver con Angular.js. Implementado en **TypeScript** [19] y escrito desde cero, el salto de un *framework* a otro no resulta directo y conlleva un aprendizaje.

Gracias a Angular.js la aplicación web se organiza mediante **módulos** (*modules*) diferenciados, en los cuales se cargan **modelos** (*data models*) para representar los datos, y se ejecutan peticiones al servidor gracias a los **servicios** (*data services*). Por último el **controlador** (*controller*) de cada vista comprende la lógica de la misma.

Por último, de entre todos los paquetes y *plugins* que podemos usar con Angular.js, destacamos **Angular UI-Router** [20] (**Figura 16**), que nos permitirá realizar el enroutamiento de la aplicación de una forma más completa incluso que el *router* nativo de Angular.js, ya que gracias al uso de vistas anidadas y estados, podremos completar una aplicación modular.

Figura 16 – Logo del UI-Router

Modelo-Vista-Controlador

El MVC [21] (Modelo-Vista-Controlador) es un patrón de **arquitectura de software** que separa los datos y la lógica de negocio, la interfaz de usuario y el control de eventos y las comunicaciones de una aplicación. Gracias a ello la aplicación resulta más modular, fácil de mantener y permite una mejor **escalabilidad**. A continuación definiremos sus componentes:

- El **modelo** contiene los datos de la lógica de negocio, define los objetos que se pedirán al servidor y cobrarán sentido en la interfaz.
- La **vista** representa los modelos de datos de manera visual en la interfaz de usuario.
- El **controlador** contiene la lógica de negocio en si misma, es decir, realizará las peticiones al servidor y completará los modelos que se sirven a la interfaz.

Este patrón será el adoptado en la aplicación Android de este Trabajo de Fin de Grado (**Figura 17**):

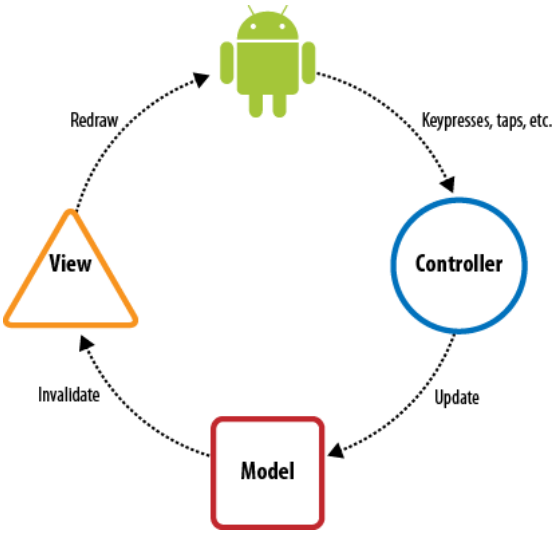


Figura 17 – Patrón MVC en Aplicación Android

Y en la aplicación web complementaria del proyecto (**Figura 18**), la cual se apoya en Angular.js:

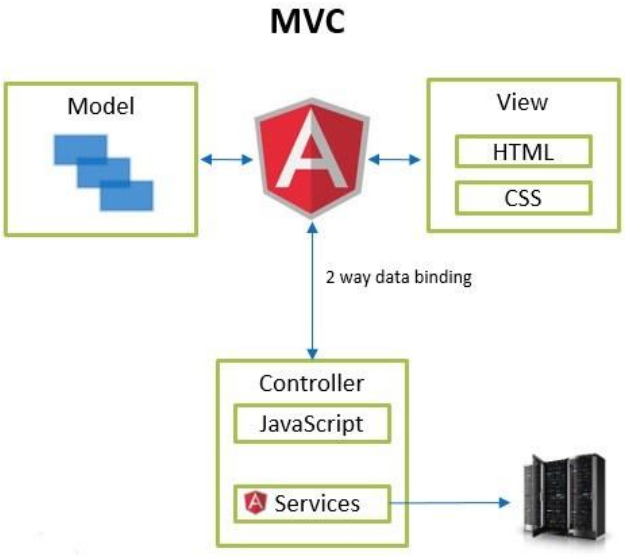


Figura 18 – Patrón MVC en Aplicación Web

MySQL

MySQL [22] (**Figura 19**) es un sistema de gestión de **bases de datos relacional** [23] de código abierto, considerado uno de las más populares del mundo (junto a Oracle y Microsoft SQL Server).

Incorpora un amplio subconjunto del lenguaje **SQL** [24], garantiza una conectividad segura y permite llevar a cabo transacciones en nuestra base de datos. Gracias al uso de claves primarias y claves foráneas, permite una gran consistencia entre los datos (realizando comprobaciones en cascada).

Figura 19 – Logo de MySQL

4. Diseño

En este apartado analizaremos todo lo relacionado con el diseño de las aplicaciones implementadas. Detallaremos el ciclo de vida escogido para el desarrollo, así como el alcance de las aplicaciones y requisitos funcionales y no funcionales. Explicaremos los casos de uso y mostraremos la evolución del modelo de datos, para finalmente visualizar las primeras maquetas y exponer la estructura de los proyectos.

Ciclo de Vida

Para la realización de ambas aplicaciones (Android y Web) se escogió el **ciclo de vida en cascada** [25] (**Figura 20**) basado en ordenar rigurosamente las etapas de un proyecto, sin poder comenzar la siguiente etapa hasta haber finalizado la anterior.

Se trata de un modelo de desarrollo clásico y muy común en la ingeniería del software que se caracteriza por llevar a cabo una revisión al final de cada proceso para decidir si afrontar la siguiente etapa o volver a la anterior. Como inconveniente principal destaca que sufrir un error de Análisis o de Diseño y no darse cuenta hasta una etapa tardía, provoca la reimplementación de varios bloques de desarrollo.

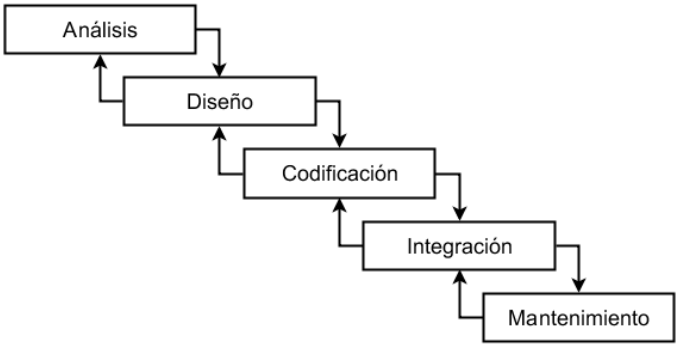


Figura 20 – Ciclo de Vida en Cascada

El modelo elegido es el que más cuadra con nuestro proyecto ya que tenemos una idea muy bien definida y sabemos cómo queremos implementarla.

Alcance de la Aplicación

El alcance de la aplicación se enfoca en acercar el uso de la tecnología táctil a la docencia en clase, centrándonos en un desarrollo Android que permita al alumno hacer *login* en el sistema con su usuario, filtrar las preguntas expuestas por el profesor y contestarlas. Incluyendo un *feedback* instantáneo acerca del resultado y aportando información sobre tiempo restante para resolver las preguntas.

Todo ello apoyado en una herramienta web que permita crear, editar y eliminar preguntas, así como filtrar estudiantes o cuestiones para evaluar los conocimientos adquiridos.

Por ello, vamos a definir las necesidades de los 2 grandes bloques a implementar:

- Aplicación **Android ClickEPS**
  - Entrar al sistema mediante usuario y contraseña (exclusivo alumnos)
  - Filtrar entre las asignaturas en las que el alumno está matriculado
  - Filtrar entre los temas de la asignatura seleccionada
  - Mostrar el listado de preguntas asociadas al tema
  - Permitir consultar el detalle de una pregunta (esté contestada, haya expirado o siga en periodo vigente de respuesta)
  - Aportar información de tiempos en preguntas sin contestar y en periodo vigente
  - Aportar información de resultados en preguntas ya contestadas
  - Aportar información de resultados en preguntas expiradas
  - Permitir contestar una pregunta seleccionando una de las cuatro opciones posibles
  - Permitir contestar una pregunta como “No sabe / No contesta”

- Aplicación **Web ClickEPS**
  - Entrar al sistema mediante usuario y contraseña (exclusivo profesores)
  - Crear, editar, duplicar y eliminar preguntas de un tema en una asignatura
  - Consultar resultados estadísticos clasificando por tema y pregunta
  - Consultar resultados estadísticos clasificando por tema y alumno (dando también opción de filtrar por una única pregunta)
  - Permitir modificar tiempos de vigencia de preguntas ya existentes

Análisis de Requisitos

En el análisis de requisitos expondremos todos los requisitos funcionales por aplicación, así como los requisitos no funcionales, detallando la entrada y salida en cada uno de ellos.

Requisitos Funcionales Aplicación Android

(RF1) Rol de Usuario

Descripción: Se define un único rol de usuario que será el del alumno. Identificado con su correo universitario podrá acceder a la aplicación. No se permite el acceso a profesores ya que no tiene sentido contestar sus propias preguntas.

(RF2) Acceso a la Aplicación

Descripción: Es necesario un proceso de autenticación mediante usuario y contraseña.

Entrada: El alumno introduce su email y contraseña.
Proceso: La aplicación consultará al API (y ésta a la base de datos) si los credenciales son correctos, en cuyo caso permitirá el acceso a la misma. En caso contrario reportará un aviso al usuario.
Salida: Acceso a la aplicación si el usuario es correcto, en otro caso recibirá un mensaje de error.

(RF3) Consulta de Grupos matriculados

Descripción: La aplicación permitirá obtener el listado de grupos asociados a las asignaturas en las que el estudiante esté matriculado.

Entrada: Identificador de alumno.
Proceso: La aplicación consultará al API por el listado de asignaturas en las que esté dado de alta el usuario.
Salida: En caso de éxito se mostrará un selector con los grupos disponibles al filtrado, en cualquier otro caso se reportará un error.

(RF4) Filtrado por Grupo matriculado

Descripción: Se permitirá el filtrado por grupo para obtener los temas asociados a un grupo en concreto.

Entrada: Selección de un grupo del listado en pantalla.
Proceso: La aplicación lanza la siguiente actividad con el identificador del grupo seleccionado.
Salida: El sistema mostrará el listado por temas, en caso contrario reportará un error.

(RF5) Consulta de Temas

Descripción: La aplicación permitirá obtener el listado de temas asociados al grupo seleccionado.

Entrada: Identificador del alumno e identificador del grupo seleccionado.
Proceso: La aplicación consultará al API por el listado de temas asociados al grupo en el que está matriculado el usuario.
Salida: En caso de éxito se mostrará un selector con los temas disponibles al filtrado, en cualquier otro caso se reportará un error.

(RF6) Filtrado por Tema

Descripción: Se permitirá el filtrado por tema para obtener las preguntas asociadas a un tema en concreto.

Entrada: Selección de un tema del listado en pantalla.
Proceso: La aplicación lanza la siguiente actividad con el identificador del tema seleccionado y el identificador del grupo anteriormente seleccionado.
Salida: El sistema mostrará el listado de preguntas, en caso contrario reportará un error.

(RF7) Consulta de Preguntas

Descripción: La aplicación permitirá obtener el listado de preguntas de un tema asociado al grupo seleccionado en los primeros pasos.

Entrada: Identificador del alumno, identificador del grupo e identificador del tema seleccionado.
Proceso: La aplicación consultará al API por el listado de preguntas asociadas al tema del grupo en el que está matriculado el usuario.
Salida: En caso de éxito se mostrará el listado de preguntas, en cualquier otro caso se reportará un error.

(RF8) Estado de las Preguntas

Descripción: En el listado de preguntas se deberá aportar información sobre el estado de la pregunta, si ya ha sido (o no) respondida, así como el tiempo restante de vigencia de la pregunta o el resultado de la misma (si fue correcta o no).

(RF9) Detalle de Pregunta

Descripción: Al seleccionar una pregunta del listado se mostrará el detalle de la misma, destacando el enunciado y las cuatro respuestas posibles. También se mantendrá el contador de tiempo, o en caso de estar ya respondida o caducada, información acerca del resultado de la pregunta, marcando aciertos y errores.

Entrada: Identificador de pregunta e identificador de alumno.
Proceso: La aplicación consulta al API si existe respuesta para dicha pregunta, en caso negativo evalúa el tiempo restante y si sigue vigente muestra las opciones posibles y habilita los botones de *submit*. Si la pregunta ya no está vigente, se marcará la respuesta correcta y se inhabilitarán los botones, y en caso de que la pregunta ya haya sido resuelta, se evaluará si ha sido contestada correctamente, mostrando el resultado.
Salida: Pantalla con el detalle de la pregunta y las opciones pertinentes, en caso de error se reportará al usuario.

(RF10) Contestar Pregunta

Descripción: En caso de estar en disposición de contestar una pregunta, se mostrará un botón para finalizar la cuestión, con su correspondiente diálogo de confirmación (ya que la acción es irreversible).

Entrada: Identificador de pregunta, identificador de alumno y respuesta seleccionada.
Proceso: El sistema procesa la petición POST y almacena en la base de datos la respuesta dada por el usuario.
Salida: Vuelta al listado de preguntas mostrando un mensaje en pantalla con la confirmación del envío y la opción seleccionada. En caso de error se reportará al usuario.

(RF11) Contestar Pregunta sin opción seleccionada

Descripción: Se deberá dar la posibilidad al estudiante de no responder una pregunta por decisión propia o por desconocimiento de la misma. Por ello se habilitará un botón de envío de respuesta vacía (con su correspondiente diálogo de confirmación).

Entrada: Identificador de pregunta, identificador de alumno y respuesta vacía.
Proceso: El sistema procesa la petición POST y almacena en la base de datos la respuesta vacía.
Salida: Vuelta al listado de preguntas mostrando un mensaje en pantalla con la confirmación del envío y la opción seleccionada. En caso de error se reportará al usuario.

Requisitos Funcionales Aplicación Web

(RF1) Rol de Usuario

Descripción: Se define un único rol de usuario que será el del profesor. Identificado con su correo universitario podrá acceder a la aplicación. No se permite el acceso a alumnos ya que no tiene sentido, puesto que la aplicación web permite administrar las preguntas de la aplicación Android.

(RF2) Acceso a la Aplicación

Descripción: Es necesario un proceso de autenticación mediante usuario y contraseña.

Entrada: El profesor introduce su email y contraseña.
Proceso: La aplicación consultará al API (y ésta a la base de datos) si los credenciales son correctos, en cuyo caso permitirá el acceso a la misma. En caso contrario reportará un aviso al usuario.
Salida: Acceso a la aplicación si el usuario es correcto, en otro caso recibirá un mensaje de error.

(RF3) Filtrado por Grupos

Descripción: El menú principal deberá permitir siempre el filtrado de información por grupos del docente (se entiende como grupo la docencia de una asignatura para un conjunto concreto de alumnos).

Entrada: Identificador de profesor.
Proceso: La aplicación consultará al API por el listado de grupos asociados al profesor.
Salida: Se muestra en el menú izquierdo el listado de grupos, en caso de error se reporta al usuario.

(RF4) Filtrado por Temas

Descripción: Además, el menú principal deberá permitir siempre el filtrado de información por temas asociados al grupo seleccionado.

Entrada: Identificador de profesor, identificador de grupo.
Proceso: La aplicación consultará al API por el listado de temas asociados al grupo seleccionado del profesor logueado.
Salida: Se muestra en el menú izquierdo el listado de temas (debajo del listado de grupos), en caso de error se reporta al usuario.

(RF5) Estadísticas por Pregunta

Descripción: Se permitirá el filtrado por pregunta (asociada al tema seleccionado del grupo filtrado inicialmente), mostrando información del enunciado, las respuestas y gráficos estadísticos que evalúen el ratio de respuesta y la calidad de las mismas.

Entrada: Identificador de tema.
Proceso: La aplicación consultará al API por el listado de preguntas asociadas al tema seleccionado (del grupo y del profesor logueado).
Salida: Se muestra el listado de preguntas y las estadísticas mencionadas, en caso de error se reporta al usuario.

(RF6) Estadísticas por Estudiante

Descripción: Se permitirá el filtrado por estudiante (perteneciente al tema seleccionado del grupo filtrado inicialmente), mostrando información general acerca de su balance de respuestas, y dando la opción de filtrado por grano fino en función de una pregunta en concreto de las contenidas en el tema seleccionado.

Entrada: Identificador de tema, identificador de alumno.
Proceso: La aplicación consultará al API por el listado de alumnos asociados al tema seleccionado (del grupo y del profesor logueado). Además consultará una pregunta en concreto para evaluar al estudiante.
Salida: Se muestra el listado de estudiantes y las estadísticas mencionadas, además del detalle de la pregunta filtrada, en caso de error se reporta al usuario.

(RF7) Nueva Pregunta

Descripción: Se permitirá crear nuevas preguntas en el sistema, asociándolas al tema y al grupo seleccionados. Consistirá en un formulario que permita introducir enunciado, cuatro respuestas, marcar la respuesta correcta, seleccionar una dificultad y marcar las fechas de activación y expiración de la pregunta.

Entrada: Identificador de tema, enunciado de pregunta, respuesta A, respuesta B, respuesta C, respuesta D, respuesta correcta, dificultad, fecha de activación y fecha de expiración.
Proceso: La aplicación comprueba que todos los campos sean correctos y estén rellenos, los valida y persiste en la base de datos.
Salida: Se muestra un aviso de pregunta creada con éxito, en caso de error se reporta al usuario.

(RF8) Editar Pregunta

Descripción: Se permitirá editar preguntas ya creadas con anterioridad, así como duplicar una pregunta ya existente o incluso borrar la pregunta. El formulario ha de ser muy parecido al de creación de pregunta.

Entrada: Identificador de tema, enunciado de pregunta, respuesta A, respuesta B, respuesta C, respuesta D, respuesta correcta, dificultad, fecha de activación y fecha de expiración.
Proceso: La aplicación comprueba que todos los campos sean correctos y estén rellenos, los valida y persiste en la base de datos.
Salida: Se muestra un aviso de pregunta editada con éxito, en caso de error se reporta al usuario.

(RF9) Cerrar sesión

Descripción: Se dará la opción de finalizar la sesión con un proceso de *logout*.

Entrada: El usuario pulsa el botón de cerrar sesión.
Proceso: La aplicación elimina la sesión de usuario y redirecciona a la página principal.
Salida: Se muestra la pagina principal, en caso de error se reporta al usuario.

Requisitos No Funcionales

(RNF1) Requisitos de Disponibilidad

(RNF 1.1) Durante labores de mantenimiento o mejora de las aplicaciones, éstas permanecerán inoperativas el tiempo de la intervención.

(RNF2) Requisitos de Seguridad

(RNF 2.1) La aplicación será accesible mediante usuario y contraseña por cada usuario.
(RNF 2.2) Las contraseñas se encriptarán mediante un algoritmo de encriptación y así se guardarán en base de datos.
(RNF 2.3) La sesión del usuario se destruye al finalizar sesión.

(RNF3) Requisitos de Usabilidad

(RNF 3.1) La aplicación tendrá unos tiempos de respuesta menores a los 10 segundos, y en caso de que alguna petición se exceda, se realizará un reintento.
(RNF 3.2) El usuario siempre será informado en caso de que algún proceso no pueda ser concluido satisfactoriamente.

(RNF4) Requisitos de Backup

(RNF 4.1) Toda la información se persistirá en base de datos.

(RNF5) Requisitos de Plataformas

(RNF 5.1) La aplicación móvil será accesible desde cualquier dispositivo Android con versión mayor o igual a 4.4.2.
(RNF 5.2) La aplicación web será accesible desde cualquier navegador web tales como Chrome, Firefox, Opera, Safari, Internet Explorer.
(RNF 5.3) El API residirá en un servidor Node.js, el cual a su vez sirve el servicio web.
(RNF 5.4) La base de datos utilizada será MySQL.

Diagrama de Casos de Uso

En la siguiente imagen mostramos el diagrama de **casos de uso** [26] (**Figura 21**) de ambas aplicaciones:

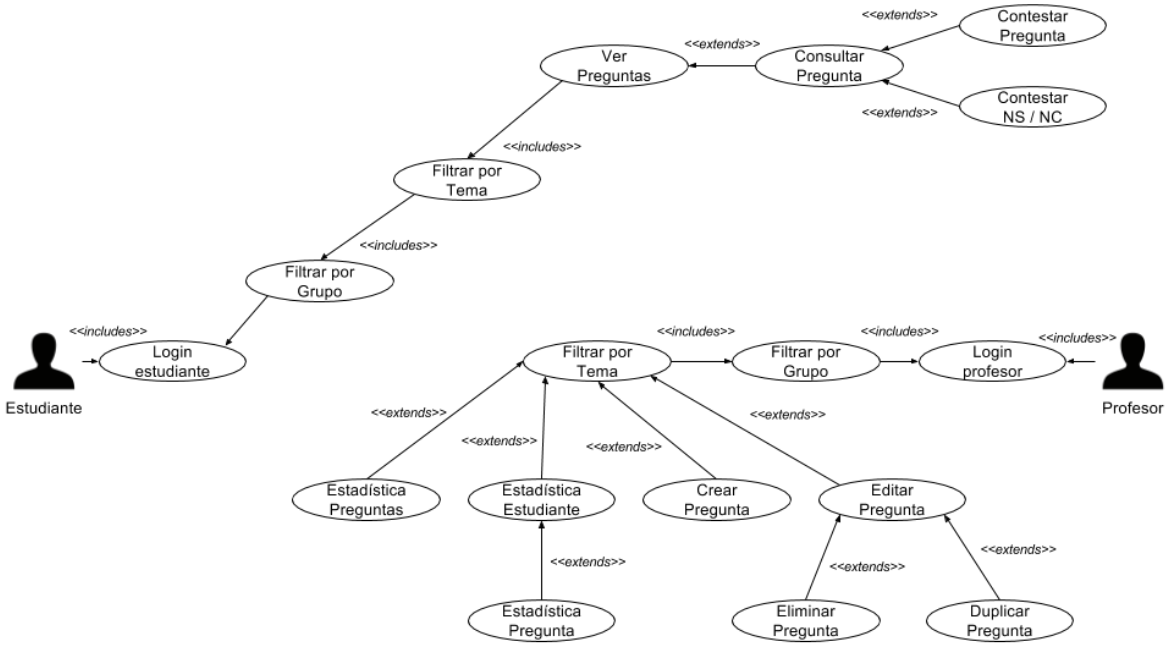


Figura 21 – Diagrama de Caso de Uso de la Aplicación

Como se puede observar, existen dos roles de usuario y cada uno podrá acceder a aplicaciones diferentes, las cuales consultarán los recursos compartidos para que cada usuario pueda consultar o crear preguntas según su perfil.

Modelo de datos

Seguidamente mostramos la evolución del diagrama Entidad-Relación ya que en una primera instancia se llevó un diseño adelante (**Figura 22**), para en siguientes iteraciones realizar sucesivas mejoras que permitieran mejorar los flujos de las aplicaciones.

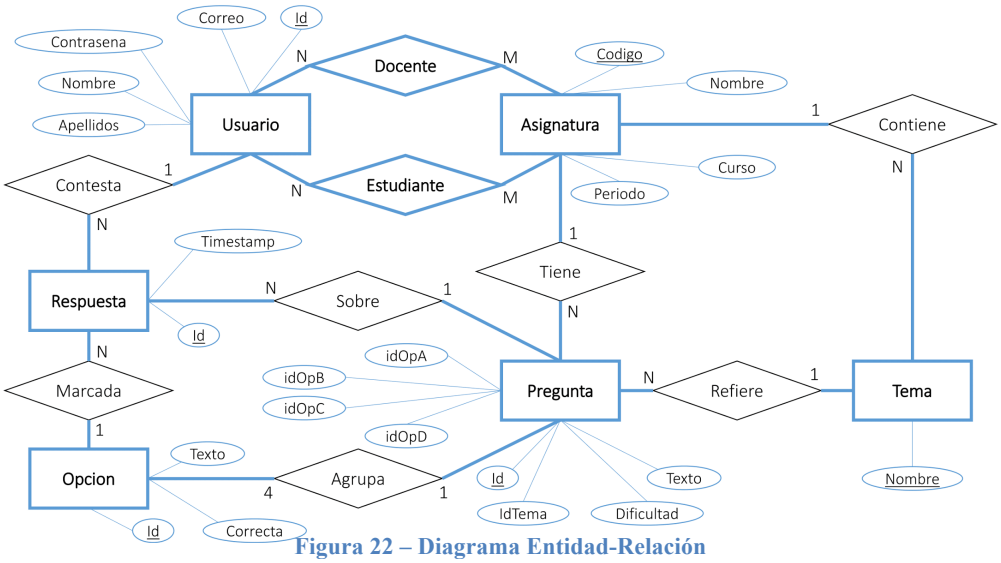


Figura 22 – Diagrama Entidad-Relación

Más adelante se llegó al modelo definitivo (**Figura 23**), el cual se muestra a continuación:

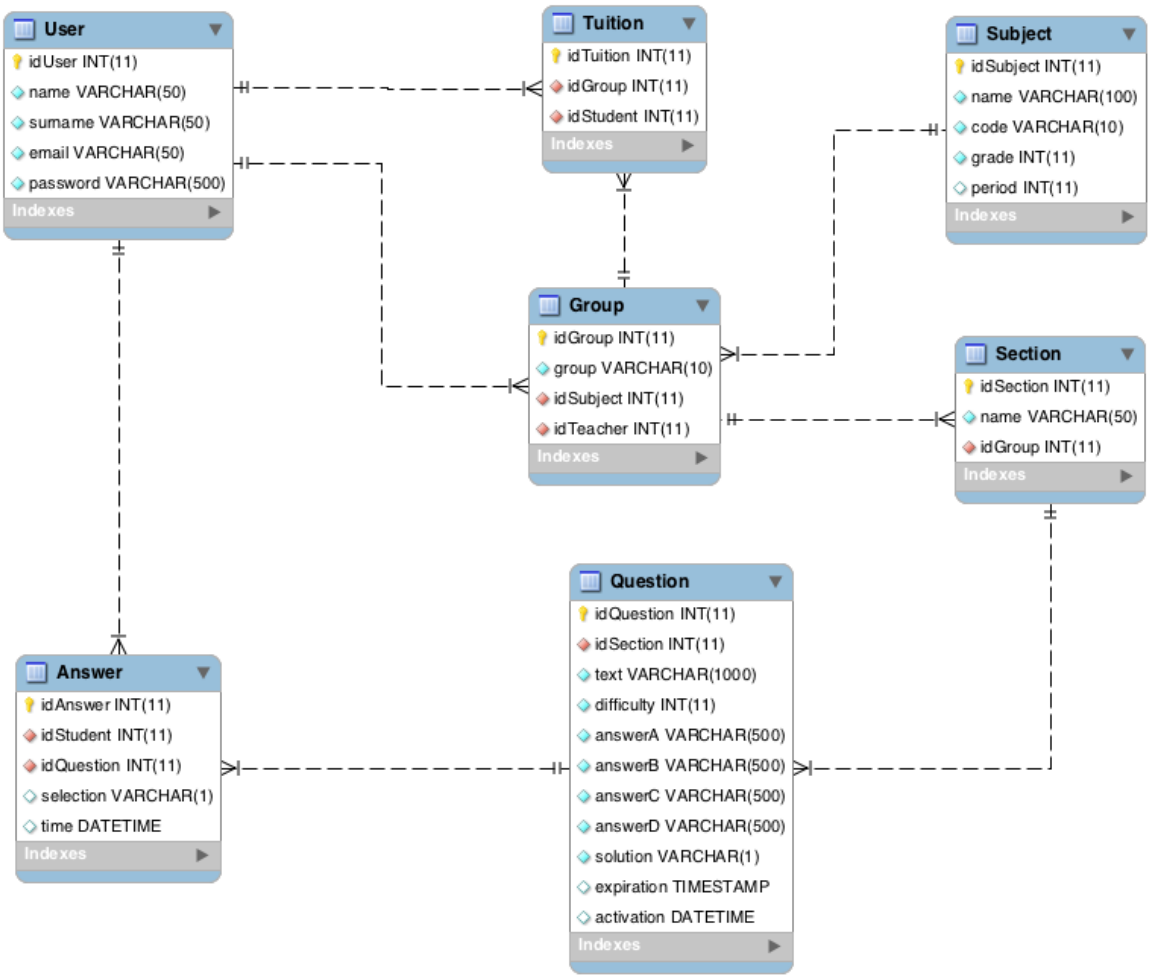


Figura 23 – Modelo de Datos de la Aplicación

Como se puede apreciar, se incluyeron las tablas de Group y Tuition para poder gestionar el hecho de que un profesor puede impartir una misma asignatura en varios grupos a diferentes clases de alumnos, así como cualquier alumno puede estar matriculado en uno de esos grupos (que representa el enlace con la asignatura).

Vamos a detallar cada una de las entidades:

- **User**: Es la entidad que representa a cualquier usuario de la aplicación (ya sea estudiante o profesor). Se identifica mediante correo y contraseña, y además lleva información asociada como son nombre y apellidos. Los profesores se relacionan con las asignaturas mediante los grupos, y los estudiante mediante las matrículas. Además las respuestas de cada alumno quedan asociadas a su identificador.
- **Group**: Representa un grupo de una asignatura en concreto y se identifica mediante la asignatura y el profesor que la imparte. Además agrupa las matrículas de los alumnos que pertenezcan al mismo. También se puede subdividir en los temas que lo conformen (de cara a clasificar las preguntas con más facilidad).
- **Tution**: Entidad que identifica la relación entre un alumno y la asignatura que está cursando.
- **Subject**: Representación de la asignatura, la cual contiene atributos tales como nombre, código, curso y periodo en el que se imparte.
- **Section**: Es la entidad que divide un grupo en secciones para poder clasificar su contenido de una manera más intuitiva.
- **Question**: Es la principal identidad y sobre la que giran las aplicaciones, representa las preguntas de la aplicación y tiene como atributos el enunciado, las respuestas posibles, la respuesta correcta, la dificultad y los tiempos de activación y expiración. Cada tema contiene un listado de preguntas y cada respuesta irá asociada a la pregunta que le corresponde.
- **Answer**: Entidad que almacena la respuesta de un alumno a una pregunta concreta, formada por la selección del estudiante y el momento en el que se contestó.

Estructura de las Aplicaciones

En este apartado describiremos la estructura de los proyectos, diferenciando la aplicación Android de la aplicación Web.

Respecto a la aplicación Android, hemos seguido la estructura sugerida por Android Studio, los ficheros fuentes se encuentran en *app/src/main* separados en dos rutas:

- *java/es/uam/eps/tfg17846/mariopolo2805/clickeps*: Donde se encuentran todos los **Activity** de la aplicación. En la carpeta *helpers* hallaremos los **Class** necesarios para la serialización de los objetos.
- res: Donde encontraremos los recursos para la aplicación tales como imágenes, logos y correspondientes **Layout** para la visualización de cada una de las actividades.

En cuanto a la aplicación Web, nos hemos basado en una aplicación que utiliza **Node.js** con **Express.js** como *framework* principal para el servidor, por lo que todas las dependencias del proyecto se encontrarán en *node_modules* y el API queda definida en el fichero *server.js*. A partir de entonces, en la ruta *public* podremos encontrar:

- *main.js:* Encargado de cargar, con *require.js* [27] según el patrón Asynchronous Module Definition (AMD) [28], todas las dependencias y hacer *bootstrapping* [29] de la aplicación angular.
- *app.js*: Donde se define nuestra aplicación angular, cargando cada módulo y generando el estado inicial de la aplicación.
- *index.html*: Vista principal de la aplicación, la cual cargará los estilos compilados con **LESS** [30] del *stylesheets/main.css* y el fichero *main.js*.
- *modules*: Donde se definen los módulos de cada pantalla de la aplicación.
- *commons*: Directorio donde definiremos los modelos angular (*models*) que servirán para representar los objetos que nos intercambiaremos con **Backend** mediante los servicios de datos (*services*), que también se encuentran en este directorio.
- *resources*: Ruta en la que se encuentran los recursos tales como imágenes o iconos.
- *less*: Donde encontramos los estilos en LESS.
- *stylesheets*: Directorio que almacenará el fichero *main.css* con todos los estilos compilados (después de pasar por el preprocesador LESS gracias a un *plugin* de Node.js).
- *wrapper*: Carpeta que define el *header* y *footer* de la aplicación.

5. Desarrollo

En esta sección explicaremos las diferentes fases del desarrollo, así como la lógica asociada a cada aplicación y cómo se comunican entre ellas. Para el desarrollo de la aplicación Android ya hemos mencionado que utilizaremos Android Studio, mientras que para el desarrollo Web no nos apoyaremos en ningún IDE y mediante Sublime Text y Atom (y numerosos *plugins* que nos facilitarán el trabajo) llevaremos a cabo la implementación.

Aplicación Android

Para implementar la aplicación Android vamos a seguir el patrón más frecuente por el cual cada *Activity* contiene la lógica de un *Layout*. De esta manera, iremos pasando por diferentes actividades para completar el flujo que se desea para el usuario. Gracias a Android Studio la implementación resultará más amena, así como la simulación de la aplicación, ya que nos dará opción de correrla en un dispositivo android virtualizado, o mediante conexión USB lanzarla en nuestro propio dispositivo personal (haciéndose así más fácil la labor de depuración).

Conexión con el Servidor

Para realizar la conexión con el servidor y la API, nos apoyaremos en Android Volley [31], librería HTTP para Android fundamental para conectar con la API y poder realizar las peticiones de nuestra aplicación.

En el **Anexo C – Fragmento 1** podemos ver cómo se realiza la configuración para la conexión. También podemos consultar en el **Anexo C – Fragmento 2** cómo declarar un fichero de constantes, donde se almacenarán todas las direcciones a las que se atacará para hacer las peticiones.

Finalmente se implementa la interfaz del servidor, la cual detalla las rutas a las que se ataca, así como los parámetros necesarios para obtener respuesta de las mismas. Consultar **Anexo C – Fragmento 3** para más información.

Activities

Mediante el uso de actividades, daremos lógica a cada una de las pantallas de la aplicación, en el **Anexo C – Fragmento 4** podemos ver un ejemplo de ello.

Todas las actividades se caracterizan por un método **onCreate** encargado de inicializar el *layout* y un método **onResume**, el cual contiene la lógica.

Layouts

La representación de la interfaz se realiza mediante su fichero de *layout* correspondiente, en el cual se establecen propiedades de anchura, altura, orientación, estilos y los componentes necesarios para la pantalla. En el **Anexo C – Fragmento 5** mostramos un ejemplo con la vista de grupos.

La representación de los datos es posible gracias a las clases que representan cada entidad, también mostradas en el **Anexo C – Fragmento 6**.

Aplicación Web

La aplicación web se ejecutará en un servidor Node.js que hará las veces también de API gracias al *framework* Express.js, el cual servirá las rutas para que consulten tanto la web como la aplicación Android.

Además implementaremos la aplicación web con otro *framework*, Angular.js, el cual nos provee de funcionalidades múltiples tales como **controllers** para manejar las vistas, **models** para representar los datos y **data services** para consultar al API, todo ello bajo el patrón MVC ya explicado anteriormente.

Conexión con el Servidor

Express.js nos proporciona una configuración sencilla y cumple con todo lo necesario para poder ejecutar nuestra API. En el **Anexo C – Fragmento 7** se muestra cómo se realiza la conexión con la base de datos y cómo se exponen las rutas de la API.

API

La totalidad de peticiones se describen en el mismo fichero de configuración, justo a continuación del mismo.

Como podemos ver en el **Anexo C – Fragmento 8**, diferenciamos las peticiones por bloques, separando en el primero las de la aplicación Web, y a su vez separando en submódulos. En el ejemplo se muestran las de *login* y a continuación los apartados de estudiantes y de asignaturas.

Gracias a la conexión con la base de datos MySQL, las consultas se declaran en este mismo fichero, recibiendo como parámetros los datos necesarios para las mismas.

En el segundo bloque se declaran las peticiones de la aplicación móvil, las cuales siguen el mismo patrón, solo que en este caso la consulta no se realiza sobre el mismo servidor (ya que con la aplicación web, API y aplicación residen bajo un mismo dominio). Consultar el **Anexo C – Fragmento 9** para ver el ejemplo.

Es mediante vía remota como los dispositivos android consultarán las rutas mencionadas, de nuevo pasando los parámetros indicados para resolver las peticiones. No obstante su implementación es prácticamente la misma que el resto de peticiones, a los efectos la API es transparente en todos sus métodos a la hora de atacarla.

Enrutado y Estados

Para resolver el enrutado de la aplicación hemos optado por no utilizar el *router* nativo de angular y usar el UI-Router, el cual es respaldado por una gran comunidad y permite numerosas funcionalidades como cargar estados asociados a vistas, o por ejemplo cargar varias vistas bajo un mismo estado maestro (como será el caso de nuestros *tabs*).

Para ver cómo se define el árbol de estados en la aplicación angular, consultar el **Anexo C – Fragmento 10**.

En el ejemplo, se define un estado principal **wrapper** con la vista *wrapper.html* que contiene tres partes diferenciadas, la barra superior, el contenedor y la barra inferior.

Además podemos diferenciar diferentes cargas de contenido, ya que en la barra superior se carga un *template* externo y se hace referencia a un controlador angular que lo gobierna. Sin embargo el contenido se define como una vista que será sobrescrita posteriormente en función de la pantalla a la que naveguemos. Finalmente la barra inferior carga un *template* estático sin lógica de controlador asociada.

A continuación definimos el módulo principal de la aplicación, en el cual se declara un estado con subestados hijos del primero, para establecer cada una de las cuatro vistas del menú principal. Consultar **Anexo C – Fragmento 11** para mayor información.

En este caso, el estado padre se define como abstracto, por lo que no podrá ser accesible directamente y precisará de la carga de uno de sus estados hijos. Sin embargo, toda su lógica gobernará cada una de las pantallas que se carguen en niveles inferiores.

De esta manera tenemos centralizada toda la lógica asociada al menú principal y podremos cambiar de vista sin necesidad de cargar nuevos datos.

Módulos Angular

La base de toda aplicación angular radica en sus módulos, cada uno de los cuales define todo lo necesario para que una vista tenga sentido. Se declara el controlador que gobernará la vista, así como los modelos de datos que se representarán y los servicios de datos necesarios para realizar las peticiones al servidor.

Para ver un ejemplo del módulo angular de la vista principal de la aplicación, consultar el **Anexo C – Fragmento 12**. Aquí se inyectan todas las dependencias necesarias para que el controlador pueda trabajar. Además se declaran los módulos que también es necesario precargar para el correcto funcionamiento (en el caso del ejemplo, el módulo de gráficos y de enrutado).

Cada controlador recibe las dependencias mencionadas e inicializa los parámetros necesarios para el correcto funcionamiento de su pantalla asociada, como podemos ver en el **Anexo C – Fragmento 13**.

El controlador establece toda la lógica de la vista, peticiones y construcción de modelos. Se puede consultar el **Anexo C – Fragmento 14** para ver el ejemplo (en este caso acerca de las pantallas de creación y edición de preguntas).

Todos los modelos de datos se definen de manera común en el mismo módulo angular (ver **Anexo C – Fragmento 15**). En el mismo subdirectorio se declaran cada uno de los modelos de datos definiendo los parámetros que lo conforman. Se muestra el modelo de grupos en el **Anexo C – Fragmento 16**.

De igual manera todos los servicios de datos se definen de manera conjunta (ver **Anexo C – Fragmento 17**). E igualmente se implementa cada uno de ellos en el mismo directorio. Consultar el **Anexo C – Fragmento 18** para ver el servicio de datos de grupos.

Vistas

A continuación mostraremos cómo se han implementado las vistas apoyándonos en el sistema de *templating* de angular y sus múltiples directivas junto con la sintaxis típica de **HTML** [32]. Gracias a angular, la vista se enriquece con lógica que tratará el **DOM** [33] (Document Object Model) para una mejor representación de los datos.

Como veremos en cada *template*, la manera de enlazar con el controlador es mediante **vm** (*view model*), el cual será referenciado en el UI-Router para enlazar cada vista con su controlador correspondiente.

Consultar el **Anexo C – Fragmento 19** para ver la vista del módulo estadísticas por pregunta.

Estilos

Para la carga de estilos hemos usado un *plugin* de node, que transpila el código LESS a **CSS** [34]. Hemos elegido LESS porque facilita enormemente el trabajo con las hojas de estilo gracias a la anidación. Además haremos uso de sus ***mixins*** [35] para declarar los *vendor-prefixes* [36] de una manera más modulable.

Por otro lado también cabe destacar el uso de **Flex-Model** [37] para la composición de las diferentes vistas, mucho más intuitivo que el ya conocido Box-Model [38].

Podemos ver un ejemplo de estilos sobre la vista de estadísticas por pregunta en el **Anexo C – Fragmento 20**. En el mismo ejemplo podremos ver la utilidad de los *mixins*, los cuales se definen aparte (ver **Anexo C – Fragmento 21**) de manera que puedan ser reutilizados en cualquier punto.

Finalmente destacamos el uso de un grupo de tipografías concreto (consultar **Anexo C – Fragmento 22**), todas ellas agrupadas y definidas igualmente de manera modular. Así como los colores (ver **Anexo C – Fragmento 23**), otro ejemplo de declaración como bloque único y reusable explotando el uso de las constantes de LESS.

6. Integración, pruebas y resultados

En este apartado describiremos brevemente cómo se realizó la integración de las dos aplicaciones en un entorno real, cómo se realizó la fase de pruebas y mostraremos los resultados de la aplicación.

Pruebas Unitarias

Las pruebas unitarias se han ido comprobando a la par que el desarrollo, analizando cada controlador y testeando que cada función recibiera unos parámetros para devolver los esperados.

Además en ambas aplicaciones se podía comprobar rápidamente, gracias a la interfaz, si los valores devueltos correspondían con los adecuados. Las llamadas a la API se comprobaron vía Postman [39] y mediante la terminal de Linux con Curl [40].

Pruebas Funcionales

En el momento en el que las aplicaciones se encontraron en un punto de madurez apto, se comenzó con la fase de pruebas funcionales. Para ello se realizó un análisis concreto sobre un flujo relativo a un requisito funcional.

A continuación mostramos algunas de las pruebas en la aplicación Android:

Requisito Funcional	**RF2**
Descripción	Vamos a comprobar el acceso a la aplicación
Entrada	Correo del alumno y contraseña
Resultados	El alumno se loguea en la aplicación
Prueba con éxito	Sí

Tabla 2 – Prueba Funcional sobre RF2 de Aplicación Android

Requisito Funcional	**RF7**
Descripción	Comprobaremos si la consulta de preguntas asociadas a un tema y un grupo, funciona correctamente
Entrada	Identificador de alumno, identificador de tema e identificador de grupo
Resultados	El alumno es dirigido a la vista del listado de preguntas
Prueba con éxito	Sí

Tabla 3 – Prueba Funcional sobre RF7 de Aplicación Android

Y seguidamente detallamos también pruebas en la aplicación web:

Requisito Funcional	RF4
Descripción	Vamos a comprobar el filtrado por temas según el profesor y el grupo seleccionado, se muestra correctamente
Entrada	Identificador de profesor e identificador de grupo
Resultados	El profesor visualiza en el panel izquierdo el listado de temas (clickables) correspondientes al grupo seleccionado
Prueba con éxito	Sí

Tabla 4 – Prueba Funcional sobre RF4 de Aplicación Web

Requisito Funcional	RF7
Descripción	Nos cercioramos de que la creación de preguntas funciona como se espera y persiste todos los datos
Entrada	Identificador de tema, enunciado de pregunta, respuesta A, respuesta B, respuesta C, respuesta D, respuesta correcta, dificultad, fecha de activación y fecha de expiración
Resultados	El profesor recibe un aviso de que la pregunta se ha creado con éxito pudiéndose visualizar ahora en el listado de preguntas
Prueba con éxito	Sí

Tabla 5 – Prueba Funcional sobre RF7 de Aplicación Web

Por último resaltar que también se han realizado pruebas funcionales en un entorno real de producción con profesores y alumnos reales (para ver más información, ir al **Anexo B**).

Como conclusión de las pruebas realizadas destacamos que la aplicación Android se testeó en una clase con alrededor de 30 alumnos, los cuales completaron preguntas reales realizadas por la profesora Estrella Pulido. El servidor persistió en base de datos todas las respuestas (si bien con algún error que en su correspondiente reintento finalizó con éxito).

Para realizar las pruebas de la aplicación web, se dio de alta en el sistema a 3 profesores que exploraron todas las vistas e interactuaron con las mismas, sacando conclusiones y puntos a mejorar de la interfaz (las cuales mencionaremos en el apartado de conclusiones y futuras mejoras).

En general los resultados han sido satisfactorios.

Integración en Producción

Una vez ambas aplicaciones habían sido probadas por separado y en un entorno local, se pasó a montar el entorno de producción.

Para sacar una *release* de la aplicación Android nos apoyamos en la herramienta de **Android Studio** que nos permite firmar una distribución con nuestros credenciales de cara a ser distribuida.

Además cerraremos un *Tag* en el proyecto personal de **GitHub** [41] (repositorio privado que funciona con Git), de cara a que en el futuro podamos llevar un control de versiones acorde con la desplegada en producción.

Con todo ello solo debemos distribuir el fichero **APK** para facilitar la instalación de la aplicación en dispositivos Android con versión igual o superior a la 4.4.2.

Montaremos la aplicación Web, la cual también proveerá del servidor Node.js que expondrá la API públicamente (únicamente accesible con cifrado seguro), en **Heroku** [42] conectado también al repositorio personal en GitHub.

De una manera rápida y fácilmente configurable, Heroku nos aporta las herramientas necesarias para desplegar el servidor en remoto, con lo que ya tendremos la web accesible mediante cualquier navegador, y el API estará lista para ser consultada por la aplicación Android.

7. Conclusiones y trabajo futuro

Conclusiones

La pretensión de este trabajo era conseguir una herramienta que facilitase la docencia online, tanto para el profesor como para el alumno. Se ha desarrollado una aplicación Android que cumple con los requisitos requeridos y una aplicación Web que complementa a la anterior, para que el docente pueda gestionar las preguntas y los resultados.

Respecto a las tecnologías con las que se ha trabajado, podemos considerarlas de las más punteras ya que Android está sumamente extendido a nivel mundial y Angular.js es uno de los *frameworks* más usados para aplicaciones Web. Todo ello complementado con el API expuesto con Node.js y las múltiples librerías usadas en ambas aplicaciones, conforman una base muy notable de cara a mantener y mejorar ambas implementaciones.

Finalmente se ha conseguido el objetivo perseguido, aprendiendo cada día en el ciclo de desarrollo de ambas aplicaciones.

Trabajo futuro

Esta primera versión de las aplicaciones es totalmente funcional y ya ha sido probada en entornos reales. Gracias a dichas pruebas (ver **Anexo B**) hemos podido extraer criterios de mejora en la **usabilidad** de ambas aplicaciones, mejorando las interfaces para un uso más fácil para el usuario.

Algunas de estas propuestas en la aplicación Android fueron, destacar más el **enunciado** de las **preguntas** al consultar el detalle o poder **corregir** una **respuesta** ya dada. Respecto a la aplicación Web se recalca la posibilidad de mantener un **tema seleccionado** por omisión, **diferenciar** mejor las preguntas sin contestar de las marcadas como "No sabe/No contesta", dar la posibilidad de crear preguntas **sin fecha de expiración** y clarificar el campo de **dificultad** de las preguntas.

Una herramienta útil de cara a facilitar el flujo correcto de las aplicaciones, podría ser el desarrollo de algún **módulo** que permitiese la **gestión de las entidades** que se manejan. Es decir, actualmente toda la gestión de usuarios, asignaturas, grupos y demás entidades de alto nivel, se tratan vía INSERT en la base de datos. ClickEPS se ciñe a la creación de preguntas, respuestas y datos estadísticos (apoyándose en las entidades mencionadas).

Respecto a la aplicación móvil se podría analizar el uso de los nuevos *frameworks* de **Front-End** que permiten desarrollar aplicaciones móviles para Android e iOS utilizando un único lenguaje. Algunos de los proyectos más interesantes son **Ionic**, **React Native** o **Framework 7** (por mencionar algunos).

Finalmente el desarrollo web igualmente podría ser mejorado desarrollándolo con nuevas tecnologías tales como **Angular 2**, **React** o incluso **Vue.js**. Uno de los *stacks* más extendidos es **MEAN Stack**, formado por **MongoDB**, Express.js, **Angular 2** y Node.js.

8. Referencias

[1] Derek Bruff (2017) - Classroom Response Systems (“Clickers”) - Vanderbilt University, *https://cft.vanderbilt.edu/guides-sub-pages/clickers/*

[2] Kahoot - How it works, *https://getkahoot.com/how-it-works*

[3] Virginia Gaitán (2013) - Gamificación: el aprendizaje divertido - Educativa, *http://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/*

[4] Plickers vs Kahoot, *http://mimochiladocente.es/?p=276*

[5] Jordi Sánchez (2006) - ¿Qué es un Framework? – Jordisan Blog, *http://jordisan.net/blog/2006/que-es-un-framework/*

[6] Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura (2012) Programming Android - O'Reilly Media

[7] Java Programming Language, *https://en.wikipedia.org/wiki/Java_(programming_language)*

[8] IDE, *http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment*

[9] Tom Hughes-Croucher, Mike Wilson (2012) - Node: Up and Running - O'Reilly Media

[10] Ecma International (2016) - Standard ECMA-262 (ECMAScript® 2016 Language Specification), *https://www.ecma-international.org/publications/standards/Ecma-262.htm*

[11] Ethan Brown (2014) - Web Development with Node and Express - O'Reilly Media

[12] Alber Pedraza (2014) - ¿Qué es desarrollo Front-End?, *https://desarrollofrontend.com/que-es-desarrollo-frontend/*

[13] Asier Marqués (2013) - Conceptos sobre API REST, *http://asiermarques.com/2013/conceptos-sobre-apis-rest/*

[14] What is NPM?, *https://docs.npmjs.com/getting-started/what-is-npm*

[15] Todd Motto (2016) - AngularJS styleguide (ES2015), *https://github.com/toddmotto/angularjs-styleguide*

[16] David Flanagan (2011) - JavaScript: The Definitive Guide - O'Reilly Media

[17] Programación Imperativa, *https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa*

[18] John Papa (2016) – Angular 2, *https://angular.io/https://github.com/johnpapa/angular-styleguide/blob/master/a2/README.md*

[19] TypeScript, *https://www.typescriptlang.org/*

[20] UI-Router, *https://ui-router.github.io/about/*

[21] MVC, *https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador*

[22] Seyed M.M. Tahaghoghi, Hugh E. Williams (2006) - Learning MySQL - O'Reilly Media

[23] Base de datos relacional, *http://searchdatacenter.techtarget.com/es/definicion/Base-de-datos-relacional*

[24] SQL, *http://searchsqlserver.techtarget.com/definition/SQL*

[25] Metodología en cascada, *http://metodologiaencascada.blogspot.com.es/*

[26] Caso de uso, *https://es.wikipedia.org/wiki/Caso_de_uso*

[27] Require.js, *http://requirejs.org/*

[28] Addy Osmani (2015) - Learning JavaScript Design Patterns - O'Reilly Media

[29] Angular Bootstrapping, *https://docs.angularjs.org/guide/bootstrap*

[30] LESS, *http://lesscss.org/*

[31] Android Volley, *https://developer.android.com/training/volley/index.html*

[32] Jeremy Keith, Rachel Andrew (2016) - HTML5 for Web Designers - O'Reilly Media

[33] DOM, *https://www.w3.org/TR/DOM-Level-2-Core/introduction.html*

[34] Eric A. Meyer, Estelle Weyl (2016) CSS: The Definitive Guide - O'Reilly Media

[35] Mixins, *http://lesscss.org/features/#mixins-as-functions-feature*

[36] Vendor-Prefixes, *https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix*

[37] Estelle Weyl (2017) - Flexbox in CSS - O'Reilly Media
[38] Box-Model, *https://www.w3schools.com/css/css_boxmodel.asp*
[39] Postman, *https://www.getpostman.com/*
[40] Curl, *https://curl.haxx.se/*
[41] GitHub, *https://es.wikipedia.org/wiki/GitHub*
[42] Heroku, *https://es.wikipedia.org/wiki/Heroku*

9. Glosario

MVC	Model View Controller
XML	eXtensible Markup Language
IDE	Integrated Development Environment
SQL	Structured Query Language
API	Application Programming Interface
REST	REpresentational State Transfer
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
CSS	Cascading StylesheetS
APK	Android aPplication pacKage

Anexos

A. Manual de Usuario

Este anexo contiene una **guía de uso básica** para el manejo de la aplicación ClickEPS.
La guía detalla 2 interfaces:

- Interfaz Web (para el uso por parte de los profesores)
- Interfaz Android (para el uso por parte de los alumnos)

Interfaz Web

Para hacer *login* debemos introducir nuestro usuario (correo) y contraseña. Únicamente pueden loguearse profesores. Una vez dentro, el menú principal se compone de 4 secciones:

- Estadísticas por Pregunta
- Estadísticas por Estudiante
- Nueva Pregunta
- Editar Pregunta

Además a la izquierda se muestran los **grupos** en los cuales somos docentes y los temas asociados a cada grupo. Es **importante** saber que los temas se resetean cada vez que cambiamos de sección (no conservan estado). A la derecha se muestra el usuario logueado y la opción de cerrar sesión.

Estadísticas por Pregunta

En esta sección podremos seleccionar los **temas** deseados y comprobar estadísticas por **pregunta** . Se mostrará el enunciado de la misma y las 4 opciones de respuesta (marcando la correcta). Además se aporta información acerca de la fecha de expiración de la pregunta (el límite válido para que los alumnos la contesten), una vez expirado el tiempo, la pregunta se le marcará a cada alumno que no responda como “No sabe / No contesta”.

En caso de que la pregunta tenga respuesta registradas, se mostrarán en un **gráfico** inferior las estadísticas mencionadas.

Estadísticas por Estudiante

Esta sección es similar a la anterior, solo que ahora clasificaremos por **estudiante** de manera más concreta. Igualmente habrá que seleccionar el tema deseado y podremos ver los totales del alumno en el **gráfico** de estadísticas.

En el bloque inferior se muestra en **detalle** cada pregunta, pudiendo ver la respuesta dada por el estudiante (marcando en cada caso si acertó o falló).

Nueva Pregunta

En este apartado podremos crear nuevas preguntas para los alumnos. Igual que antes, hemos de seleccionar el **tema** en el que queremos crear la pregunta, rellenar su enunciado y respuestas y marcar la correcta.

Además tendremos que establecer el momento de **activación** de la pregunta (a partir del cual los estudiantes ya la tendrán visible en sus dispositivos) y la fecha de **expiración** (marcando el límite de tiempo que tendrán para contestar).

Editar Pregunta

En este último apartado se da la opción de editar preguntas ya creadas en el sistema. Una vez más hemos de seleccionar el **tema** del que queremos cargar preguntas y seleccionamos la que queramos editar.

Además se aportan las opciones de **eliminar** una pregunta o incluso **duplicarla** , de cara a editar la pregunta duplicada y aprovechar los datos sin tener que crearlos partiendo de cero.

Interfaz Android

Para hacer *login* debemos introducir nuestro usuario (correo) y contraseña. Únicamente pueden loguearse alumnos.

Una vez dentro, la primera pantalla nos muestra los **grupos** de las asignaturas en las que estamos matriculados. Seleccionamos el grupo deseado y pulsamos “Siguiente”.

Ahora se nos mostrará el listado de **temas** de la asignatura seleccionada. De manera similar seleccionamos el tema y pulsamos de nuevo en “Siguiente”.

El sistema nos mostrará entonces el listado de **preguntas** asociadas a dicho tema y asignatura, aportando además información de aquellas preguntas que ya han sido contestadas (marcadas con un candado) y detallando la respuesta que se dio en su momento.

Además también podremos ver las preguntas disponibles (marcadas con una interrogación), las cuales informan del **tiempo restante** que queda para contestar la pregunta antes de que ésta se de por finalizada.

Al pulsar sobre una pregunta accederemos a su **detalle**, el cual muestra más información, como enunciado, cuatro opciones de respuesta y botones de acción para contestar.

Igualmente, en caso de que la pregunta ya fuera contestada, solo mostrará información sobre la misma, siendo ya **imposible** modificar la respuesta dada.

B. Informes de Pruebas y Usabilidad

Las pruebas se han realizado en un entorno de producción con alumnos y profesores reales. Como podremos comprobar los resultados son razonablemente satisfactorios.

Resultados test de usabilidad aplicación Android

Figura 24 – Encuesta sobre Filtrado de Grupos en Aplicación Android

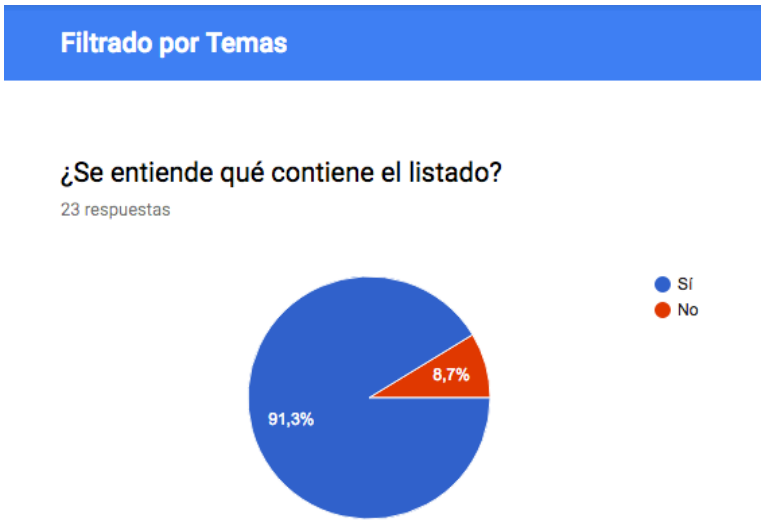


Figura 25 – Encuesta sobre Filtrado de Temas en Aplicación Android

Figura 26 – Encuesta sobre Listado de Preguntas en Aplicación Android

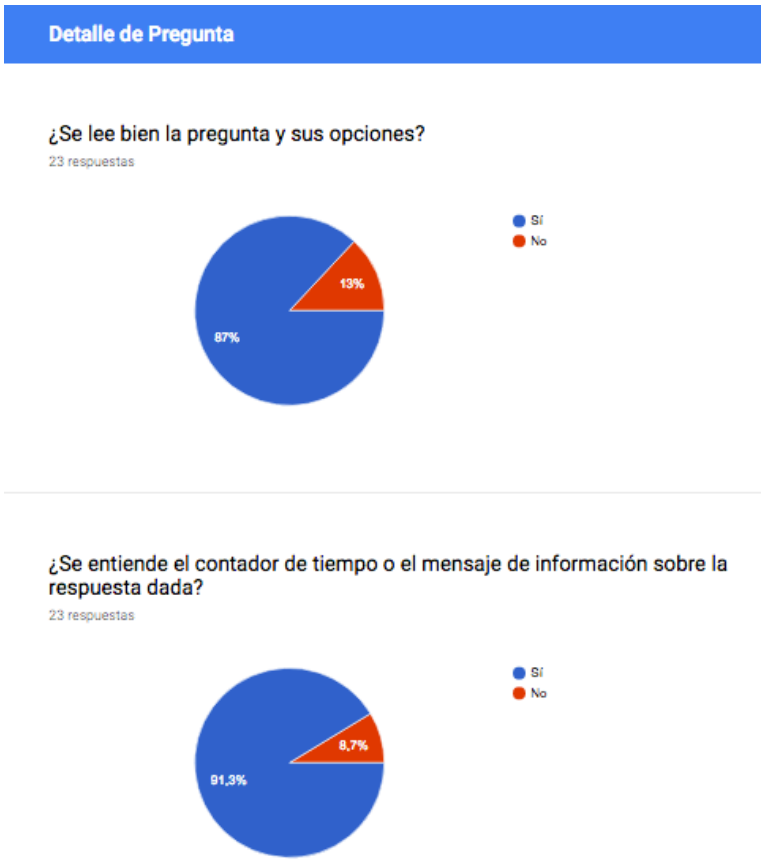


Figura 27 – Encuesta sobre Detalle de Pregunta en Aplicación Android (1)

Figura 28 – Encuesta sobre Detalle de Pregunta en Aplicación Android (2)

Resultados test de usabilidad aplicación Web

Figura 29 – Encuesta sobre Estadísticas de Pregunta en Aplicación Web (1)

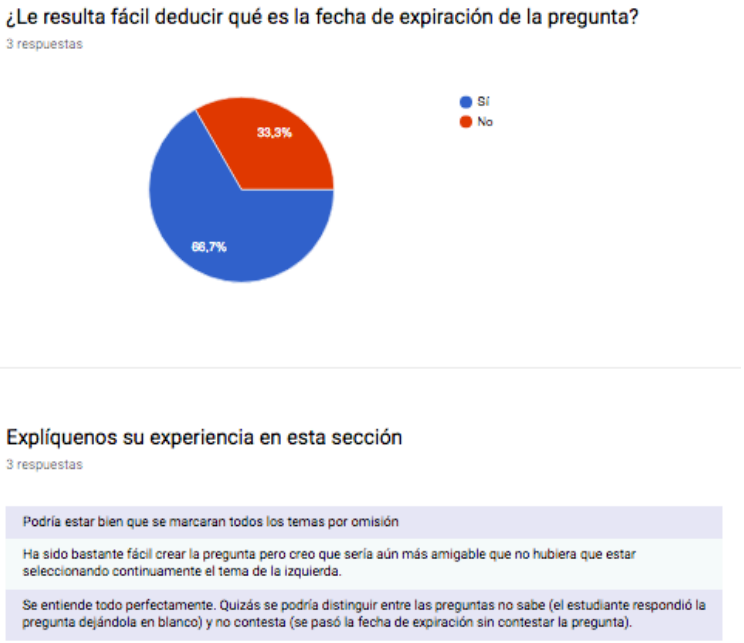


Figura 30 – Encuesta sobre Estadísticas de Pregunta en Aplicación Web (2)

Figura 31 – Encuesta sobre Estadísticas de Pregunta en Aplicación Web (3)

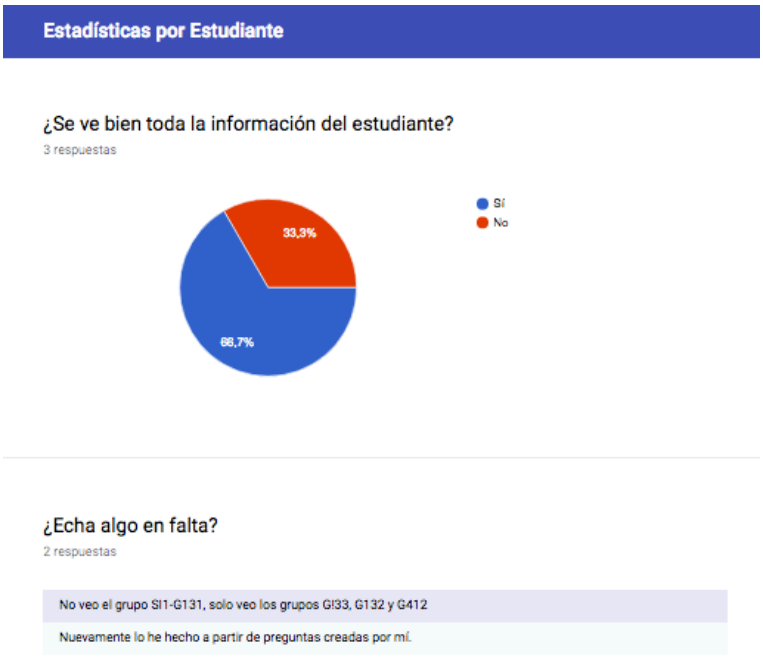


Figura 32 – Encuesta sobre Estadísticas de Estudiante en Aplicación Web (1)

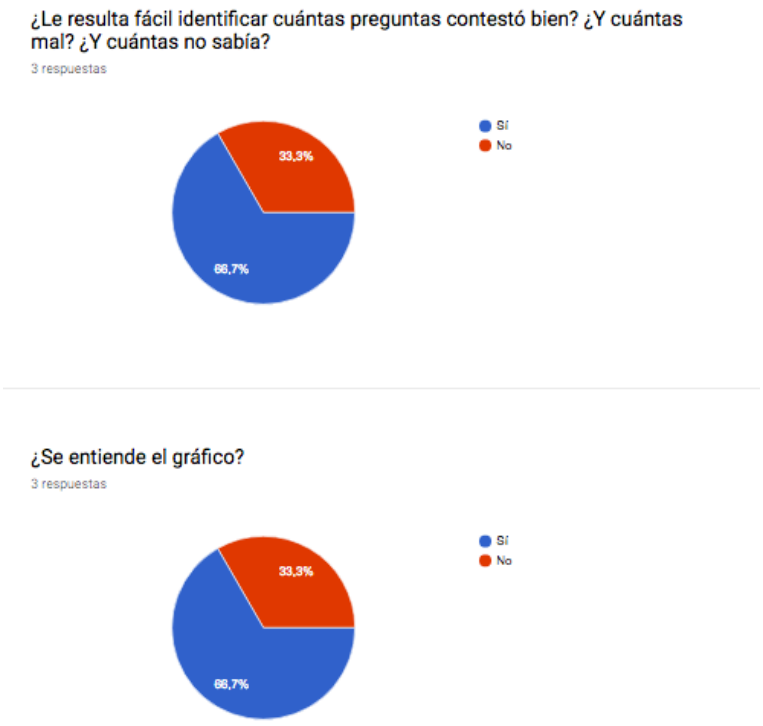
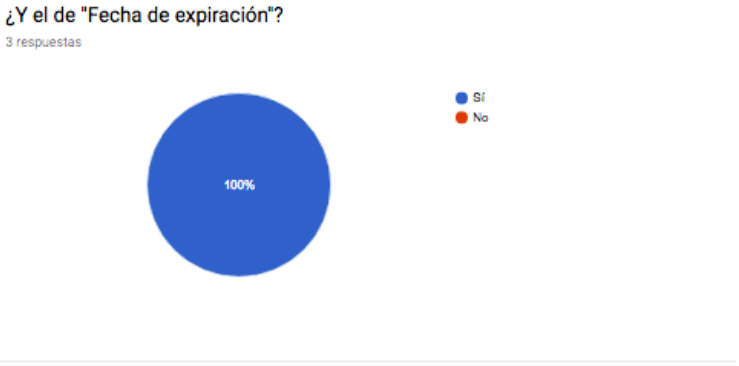


Figura 33 – Encuesta sobre Estadísticas de Estudiante en Aplicación Web (2)

Figura 34 – Encuesta sobre Nueva Pregunta en Aplicación Web (1)



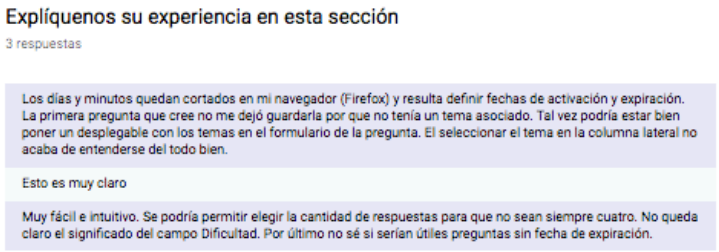


Figura 35 – Encuesta sobre Nueva Pregunta en Aplicación Web (2)

Figura 36 – Encuesta sobre Editar Pregunta en Aplicación Web

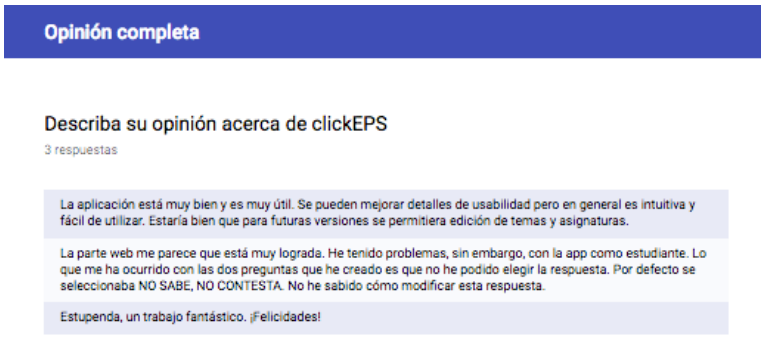


Figura 37 – Encuesta sobre Opinión Completa de la Aplicación Web

C. Fragmentos de Código

En este anexo mostraremos algunos fragmentos del código implementado:

```java
package es.uam.eps.tfg17846.mariopolo2805.clickeps.helper;

import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;

public abstract class Connection {

    public static boolean isConnected(Context context) {
        ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
        return networkInfo != null && networkInfo.isConnected();
    }

}
```

Fragmento 1 – Configuración para la Conexión con el Servidor

```java
package es.uam.eps.tfg17846.mariopolo2805.clickeps.helper;

public abstract class Constants {

    final public static String BASE_URL = "http://tfg17846.herokuapp.com";
    final public static String LOGIN_URI = "/login";
    final public static String GROUPS_URI = "/groupsOfStudent";
    final public static String SECTIONS_URI = "/sectionsOfGroup";
    final public static String QUESTIONS_URI = "/questionsOfStudent";
    final public static String ANSWERS_URI = "/answersOfStudent";
    final public static String NEW_ANSWER_URI = "/newAnswer";

    final public static String USERID_KEY = "user_id";
    final public static String GROUP_KEY = "group_id";
    final public static String SECTION_KEY = "section_id";
    final public static String QUESTION_KEY = "question_data";

}
```

Fragmento 2 – Constantes para el manejo del Servidor

```java
package es.uam.eps.tfg17846.mariopolo2805.clickeps.helper;

import android.content.Context;
import android.util.Log;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response.ErrorListener;
import com.android.volley.Response.Listener;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import java.util.HashMap;
import java.util.Map;

public class ServerInterface {

    private static ServerInterface serverInterface;
    private RequestQueue queue;

    private ServerInterface(Context context) {
        queue = Volley.newRequestQueue(context);
    }

    public static ServerInterface getServer(Context context) {
        if (serverInterface == null)
            serverInterface = new ServerInterface(context);
        return serverInterface;
    }

    public void login(final String username,
                      Listener<String> callback,
                      ErrorListener errorCallback) {
        String url = Constants.BASE_URL + Constants.LOGIN_URI + "/" + username;

        StringRequest request = new StringRequest(Request.Method.POST, url, callback, errorCallback);
        queue.add(request);
    }

    public void groupsOfStudent(final String userId, Listener<String> callback,
                          ErrorListener errorCallback) {
        String url = Constants.BASE_URL + Constants.GROUPS_URI + "/" + userId;

        StringRequest request = new StringRequest(Request.Method.GET, url, callback, errorCallback);
        queue.add(request);
    }

    public void sectionsOfGroup(final String groupId, Listener<String> callback,
                                ErrorListener errorCallback) {
        String url = Constants.BASE_URL + Constants.SECTIONS_URI + "/" + groupId;

        StringRequest request = new StringRequest(Request.Method.GET, url, callback, errorCallback);
        queue.add(request);
    }
```

Fragmento 3 – Interfaz del Servidor

```
public class GroupActivity extends AppCompatActivity implements OnClickListener {

    private ProgressBar progressBar;
    private Spinner groupSpinner;
    private ArrayAdapter<Group> groupAdapter;
    private List<Group> groups;

    private String userId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_group);

        progressBar = (ProgressBar) findViewById(R.id.progress_bar);
        groupSpinner = (Spinner) findViewById(R.id.group);
        groups = new ArrayList<>();

        userId = getIntent().getExtras().getString(Constants.USERID_KEY);
    }

    @Override
    protected void onResume() {
        super.onResume();

        ServerInterface server = ServerInterface.getServer(GroupActivity.this);
        server.groupsOfStudent(
                userId,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String s) {
                        progressBar.setVisibility(View.INVISIBLE);
                        try {
                            groups.clear();
                            JSONArray groupJSONList = new JSONArray(s);
                            for (int i = 0; i < groupJSONList.length(); i++) {
                                JSONObject groupJSON = groupJSONList.getJSONObject(i);
                                Group g = new Group(
                                        groupJSON.getString("idGroup"),
                                        groupJSON.getString("code"),
                                        groupJSON.getString("group"));
                                groups.add(g);
                            }

                            groupAdapter = new ArrayAdapter<>(GroupActivity.this, android.R.layout.simple_spinner_dropdown_item,
                            groupSpinner.setAdapter(groupAdapter);
                        } catch (Exception e) {
                            progressBar.setVisibility(View.INVISIBLE);
                            Toast.makeText(GroupActivity.this, "Hubo un problema al solicitar los grupos", Toast.LENGTH_SHORT).s
                        }
                    }
                }
                , new Response.ErrorListener() {
```

Fragmento 4 – Actividad de Grupos

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_tuition"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="es.uam.eps.tfg17846.mariopolo2805.clickeps.GroupActivity">

    <Spinner
        android:id="@+id/group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/group"
        android:onClick="onClick"
        android:text="Siguiente" />

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:indeterminate="true"
        android:visibility="invisible" />

</RelativeLayout>
```

Fragmento 5 – Vista de Grupos

```java
package es.uam.eps.tfg17846.mariopolo2805.clickeps.helper;

import java.io.Serializable;

public class Group implements Serializable {

    private String id;
    private String code;
    private String group;

    public Group(String id, String code, String group) {
        this.id = id;
        this.code = code;
        this.group = group;
    }

    public String getId() {
        return id;
    }

    public String getCode() {
        return code;
    }

    public String getGroup() {
        return group;
    }

    @Override
    public String toString() {
        return code + " - " + group;
    }
}
```

Fragmento 6 – Clase de Grupo

```
"use strict";

var express = require('express');
var lessMiddleware = require('less-middleware');
var bodyParser = require('body-parser')
var app = express();

app.use( bodyParser.json() );       // to support JSON-encoded bodies
app.use(bodyParser.urlencoded({     // to support URL-encoded bodies
  extended: true
}));

/*****************************************************************************
 ************************** Routing configuration **************************
 *****************************************************************************/

/* Compile less and store CSS in target/public */
app.use(lessMiddleware('public', { dest : 'public' }));
/* Default static files are in /public folder */
app.use(express.static('public'));
/* Serve uncompressed NPM packages in /lib folder */
app.use('/lib', express.static('node_modules'));

/*****************************************************************************
 ************************** Server configuration **************************
 *****************************************************************************/

/* Create the server */
var server = app.listen(process.env.PORT || 9000, function () {
    var port = server.address().port;
    console.log('Server up!');
});

/*****************************************************************************
 ************************** MySQL configuration **************************
 *****************************************************************************/

var mysql = require('mysql');
var connection = mysql.createConnection({
    port     : '3306',
    host     : 'tfg.c2j7nutmc0hf.eu-central-1.rds.amazonaws.com', //localhost
    user     : 'mariopolo2805', //mariopolo2805
    password : 'tfg17846',
    database : 'tfg' //tfg
});
```

Fragmento 7 – Inicialización del Servidor y Conexión con la Base de Datos

```
app.get('/#/login', function(req, res) {
    res.sendfile('login/login.html');
});

app.post('/login/:user', function(req, res) {
    var user = req.params.user;
    var query = "SELECT * FROM tfg.User WHERE email='" + user + "'";
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows[0]);
            res.send(json);
        }
    });
});

/* Student Queries */

app.post('/studentsOfGroup/:id', function(req, res) {
    var id = req.params.id;
    var query = "SELECT tfg.User.idUser, tfg.User.name, tfg.User.surname FROM tfg.User INNER JOIN tfg.Tuition ON tfg.User.idUser
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows);
            res.send(json);
        }
    });
});

/* Subject Queries */

app.post('/subjects', function(req, res) {
    var query = "SELECT * FROM tfg.Subject";
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows);
            res.send(json);
        }
    });
});

app.post('/subject/:id', function(req, res) {
    var id = req.params.id;
```

Fragmento 8 – Peticiones de la API para la Aplicación Web

```
app.get('/groupsOfStudent/:id', function(req, res) {
    var id = req.params.id;
    var query = "SELECT * FROM tfg.Tuition INNER JOIN tfg.Group ON tfg.Group.idGroup=tfg.Tuition.idGroup INNER JOIN tfg.Subject
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows);
            res.send(json);
        }
    });
});

app.get('/sectionsOfGroup/:id', function(req, res) {
    var id = req.params.id;
    var query = "SELECT * FROM tfg.Section WHERE idGroup='" + id + "'";
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows);
            res.send(json);
        }
    });
});

app.get('/questionsOfStudent/:id/section/:idSection', function(req, res) {
    var id = req.params.id;
    var idSection = req.params.idSection;
    var query = "SELECT * FROM tfg.User INNER JOIN tfg.Tuition ON tfg.User.idUser=tfg.Tuition.idStudent INNER JOIN tfg.Group ON
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows);
            res.send(json);
        }
    });
});

app.get('/answersOfStudent/:id/section/:idSection', function(req, res) {
    var id = req.params.id;
    var idSection = req.params.idSection;
    var query = "SELECT * FROM tfg.Answer INNER JOIN tfg.User ON tfg.Answer.idStudent=tfg.User.idUser INNER JOIN tfg.Tuition ON
    connection.query(query, function(err, rows) {
        if(err) {
            console.error("Problem with MySQL" + err);
        } else {
            var json = JSON.stringify(rows);
            res.send(json);
        }
    });
});
```

Fragmento 9 – Peticiones de la API para la Aplicación Android

```
        .config(function($stateProvider, $urlRouterProvider) {

            $urlRouterProvider.otherwise("/landingPage");

            $stateProvider
                .state('wrapper', {
                    url: '',
                    abstract: true,
                    views: {
                        'wrapper': {
                            templateUrl: 'wrapper/wrapper.html'
                        },
                        'wrapper-top@wrapper': {
                            templateUrl : 'wrapper/wrapper-top.html',
                            controller: 'TopbarCtrl',
                            controllerAs: 'vm'
                        },
                        'wrapper-content@wrapper': {
                            template: '<ui-view></ui-view>'
                        },
                        'wrapper-foot@wrapper': {
                            templateUrl : 'wrapper/wrapper-foot.html'
                        }
                    }
                });
```

Fragmento 10 – Árbol de Estados del UI-Router para definir el 'wrapper'

```
.config(function($stateProvider) {
    $stateProvider
        .state('wrapper.mainMenu', {
            url: '/mainMenu',
            abstract: true,
            views: {
                '': {
                    templateUrl: 'modules/mainMenu/mainMenu.html',
                    controller: 'MainMenuCtrl',
                    controllerAs: 'vm'
                }
            }
        })
        .state('wrapper.mainMenu.questionStats', {
            url: '',
            views: {
                '': {
                    templateUrl: 'modules/mainMenu/childViews/questionStats.html'
                }
            }
        })
        .state('wrapper.mainMenu.studentStats', {
            url: '',
            views: {
                '': {
                    templateUrl: 'modules/mainMenu/childViews/studentStats.html'
                }
            }
        })
        .state('wrapper.mainMenu.newQuestion', {
            url: '',
            views: {
                '': {
                    templateUrl: 'modules/mainMenu/childViews/newQuestion.html'
                }
            }
        })
        .state('wrapper.mainMenu.editQuestion', {
            url: '',
            views: {
                '': {
                    templateUrl: 'modules/mainMenu/childViews/editQuestion.html'
                }
            }
        });
});
```

Fragmento 11 – Árbol de Estados del UI-Router para definir el Módulo Principal

```
angular
    .module('tfg.mainMenu', ['chart.js', 'ui.router'])
    .controller('MainMenuCtrl', [
        '$rootScope',
        '$timeout',
        '$scope',
        '$state',
        'UserDataSer',
        'UserDataModel',
        'GroupDataSer',
        'GroupDataModel',
        'SectionDataSer',
        'SectionDataModel',
        'QuestionDataSer',
        'QuestionDataModel',
        'AnswerDataSer',
        'AnswerDataModel',
        controller.MainMenuCtrl])
    .controller('EditableChecksCtrl', [controller.EditableChecksCtrl])
```

Fragmento 12 – Inyección de Dependencias Angular del Módulo Principal

```
        function MainMenuCtrl(
            $rootScope,
            $timeout,
            $scope,
            $state,
            UserDataSer,
            UserDataModel,
            GroupDataSer,
            GroupDataModel,
            SectionDataSer,
            SectionDataModel,
            QuestionDataSer,
            QuestionDataModel,
            AnswerDataSer,
            AnswerDataModel) {

            var vm = this;
            vm.user = null;
            vm.groups = [];
            vm.groupActive = 0;
            vm.onlyOneSection = false;
            vm.sections = [];
            vm.numChecked = 0;
            vm.isAnyChecked = false;
            vm.tabActive = 0;
            vm.colors = ['#44C767', '#F72F2F', '#F89B3A'];
            vm.questions = [];
            vm.questionSelected = null;
            vm.answers = [];
            vm.students = [];
            vm.studentSelected = null;

            /* User */
            vm.user = UserDataSer.getUserCookie();
            $rootScope.user = vm.user;
            /* User */
```

Fragmento 13 – Inicialización de variables en Controlador Angular del Módulo Principal

```
vm.options = ['A', 'B', 'C', 'D'];
vm.newQuestion = new QuestionDataModel.QuestionData();
delete vm.newQuestion.id;
vm.newQuestion.solution = vm.options[0];

vm.setMaxDate = function(dateActivation) {
    return QuestionDataModel.setMaxDate(dateActivation);
}

vm.sendQuestion = function(question) {
    if(angular.isDefined(question)) {
        vm.newQuestion = question;
    }
    if(vm.newQuestion.idSection === null) {
        alert("Valide haber seleccionado el tema (en el panel izquierdo) al que asignarle la pregunta");
    } else {
        var exchangeModel = QuestionDataModel.getExchangeModel(vm.newQuestion);
        if(exchangeModel === "errValidDates") {
            alert("La fecha de activación no puede ser posterior (o igual) a la fecha de expiración");
        } else {
            QuestionDataSer.createQuestion(exchangeModel).then(function(result) {
                if(result === 200) {
                    alert("Pregunta creada con éxito");
                }
            });
        }
    }
}
/* New question */

/* Edit question */
vm.submitQuestion = function() {
    switch(vm.submitType) {
        case 'edit':
            return editQuestion();
        case 'duplicate':
            return duplicateQuestion();
        case 'remove':
            return removeQuestion();
    }
}

function editQuestion() {
    var exchangeModel = QuestionDataModel.getExchangeModel(vm.questionSelected);
    if(exchangeModel === "errValidDates") {
        alert("La fecha de activación no puede ser posterior (o igual) a la fecha de expiración");
    } else {
        QuestionDataSer.editQuestion(exchangeModel).then(function(result) {
            if(result === 200) {
                alert("Pregunta editada con éxito");
            }
        });
    }
}
```

Fragmento 14 – Lógica del Controlador Angular del Módulo Principal

```
define([
    'angular',
    './userDataModel',
    './subjectDataModel',
    './groupDataModel',
    './sectionDataModel',
    './questionDataModel',
    './answerDataModel'
], function(angular) {
    'use strict';

    return angular.module('tfg.models', [
        'tfg.models.userDataModel',
        'tfg.models.subjectDataModel',
        'tfg.models.groupDataModel',
        'tfg.models.sectionDataModel',
        'tfg.models.questionDataModel',
        'tfg.models.answerDataModel'
    ]);
});
```

Fragmento 15 – Definición del Módulo Angular con los Modelos de Datos Angular

```
function GroupDataModel(SubjectDataModel) {

    function GroupData(json) {
        if(angular.isDefined(json)) {
            this.id = json.idGroup;
            this.group = json.group;
            this.idSubject = json.idSubject;
            this.subject = (json.code) ? new SubjectDataModel.SubjectData(json) : null;
            this.idTeacher = json.idTeacher;
        } else {
            this.id = null;
            this.group = null;
            this.idSubject = null;
            this.subject = null;
            this.idTeacher = null;
        }
    }

    return {
        GroupData: GroupData
    };
}

angular
    .module('tfg.models.groupDataModel',[])
    .factory('GroupDataModel', ['SubjectDataModel', GroupDataModel]);
```

Fragmento 16 – Definición del Modelo de Datos Angular de Grupos

```
define([
    'angular',
    './userDataSer',
    './subjectDataSer',
    './groupDataSer',
    './sectionDataSer',
    './questionDataSer',
    './answerDataSer'
], function(angular) {
    'use strict';

    return angular.module('tfg.services', [
        'tfg.services.userDataSer',
        'tfg.services.subjectDataSer',
        'tfg.services.groupDataSer',
        'tfg.services.sectionDataSer',
        'tfg.services.questionDataSer',
        'tfg.services.answerDataSer'
    ]);
});
```

Fragmento 17 – Definición del Módulo Angular con los Servicios de Datos Angular

```
function GroupDataSer($http) {

    this.getGroupsData = function() {
        return $http({
            url: '/groups',
            method: 'POST'
        }).then(function(result) {
            return result.data;
        });
    };

    this.getGroupData = function(id) {
        return $http({
            url: '/group/' + id,
            method: 'POST'
        }).then(function(result) {
            return result.data;
        });
    };

    this.getGroupsOfTeacherData = function(id) {
        return $http({
            url: '/groupsOfTeacher/' + id,
            method: 'POST'
        }).then(function(result) {
            return result.data;
        });
    };

    this.getGroupsWithSubjectOfTeacherData = function(id) {
        return $http({
            url: '/groupsWithSubjectOfTeacher/' + id,
            method: 'POST'
        }).then(function(result) {
            return result.data;
        });
    };
}

angular
    .module('tfg.services.groupDataSer', [])
    .service('GroupDataSer', ['$http', GroupDataSer]);
```

Fragmento 18 – Definición del Servicio de Datos Angular de Grupos

```
<div class="question-stats-layout" ng-show="vm.tabActive === 0">
    <div class="question-stats-select">
        <span class="question-stats-select-title">Seleccione Pregunta:</span>
        <select ng-show="vm.questions.length > 0" ng-options="item as item.title for item in vm.questions track by item.id" ng-m
        <select ng-show="vm.questions.length === 0"><option>Seleccione temas del menú izquierdo con preguntas asociadas</option>
    </div>

    <div class="question-stats-detail" ng-show="vm.questions.length > 0">
        <div class="question-stats-detail-question">{{vm.questionSelected.text}}</div>
        <div class="question-stats-detail-answer">
            A) {{vm.questionSelected.answerA}}
            <img src="resources/icons/tick.png" class="icon-tick" ng-show="vm.questionSelected.solution === 'A'">
        </div>
        <div class="question-stats-detail-answer">
            B) {{vm.questionSelected.answerB}}
            <img src="resources/icons/tick.png" class="icon-tick" ng-show="vm.questionSelected.solution === 'B'">
        </div>
        <div class="question-stats-detail-answer">
            C) {{vm.questionSelected.answerC}}
            <img src="resources/icons/tick.png" class="icon-tick" ng-show="vm.questionSelected.solution === 'C'">
        </div>
        <div class="question-stats-detail-answer">
            D) {{vm.questionSelected.answerD}}
            <img src="resources/icons/tick.png" class="icon-tick" ng-show="vm.questionSelected.solution === 'D'">
        </div>
    </div>

    <div class="question-stats-dates" ng-show="vm.questions.length > 0">
        <span class="question-stats-dates-date">Fecha de expiración: {{vm.questionSelected.expiration.toLocaleDateString()}} {{\
        <span class="question-stats-dates-expiration">Expirada:
            <span ng-if="vm.questionSelected.expired">Si</span>
            <span ng-if="!vm.questionSelected.expired">No</span>
        </span>
    </div>

    <div class="question-stats-graph" ng-show="vm.questions.length > 0 && vm.answers.length > 0">
        <canvas id="pie" class="chart chart-pie"
            chart-data="vm.rates" chart-labels="vm.labels" chart-legend="true" chart-colours="vm.colors">
        </canvas>

        <div class="question-stats-graph-percents">
            <div class="question-stats-graph-percent">
                <img src="resources/icons/tick32.png">Respuestas Correctas {{vm.rates[0]}}
            </div>
            <div class="question-stats-graph-percent">
                <img src="resources/icons/wrong32.png">Respuestas Incorrectas {{vm.rates[1]}}
            </div>
            <div class="question-stats-graph-percent">
                <img src="resources/icons/sad32.png">No sabe / No contesta {{vm.rates[2]}}
            </div>
            <div class="question-stats-graph-percent">
                <span>Total de Respuestas: </span>{{vm.sum}}/{{vm.students.length}}
            </div>
        </div>
```

Fragmento 19 – Vista del Módulo de Estadísticas por Pregunta

```less
.question-stats-layout {
    .flex-display();
    .flex-direction(column);
    .typ16-400-black;

    .question-stats-select {
        .flex-display();
        .flex-direction(row);
        .justify-content(space-between);
        .align-items(center);
        padding: 25px 20px;

        select, option {
            width: 480px;
        }
    }

    .question-stats-detail {
        .flex-display();
        .flex-direction(column);
        .justify-content(space-between);
        padding: 0 20px;
        word-wrap: break-word;

        .question-stats-detail-question {
            padding-bottom: 15px;
        }

        .question-stats-detail-answer {
            .flex-display();
            .flex-direction(row);
            .align-items(center);
            padding-bottom: 10px;
        }

        .icon-tick {
            margin-left: 5px;
            padding-bottom: 2px;
        }
    }
```

Fragmento 20 – Estilos LESS de la Vista de Estadísticas por Pregunta

```less
// Flexbox display
// flex or inline-flex
.flex-display(@display: flex) {
    display: ~"-webkit-@{display}";
    display: ~"-moz-@{display}";
    display: ~"-ms-@{display}box"; // IE10 uses -ms-flexbox
    display: ~"-ms-@{display}"; // IE11
    display: @display;
}

// The 'flex' shorthand
// - applies to: flex items
// <positive-number>, initial, auto, or none
.flex(@columns: initial) {
  -webkit-flex: @columns;
     -moz-flex: @columns;
      -ms-flex: @columns;
          flex: @columns;
}

// Flex Flow Direction
// - applies to: flex containers
// row | row-reverse | column | column-reverse
.flex-direction(@direction: row) {
  -webkit-flex-direction: @direction;
     -moz-flex-direction: @direction;
      -ms-flex-direction: @direction;
          flex-direction: @direction;
}
```

Fragmento 21 – Definición de Mixins LESS

```less
.typ12-400-grey-light {
    font-size: 12px;
    font-weight: 400;
    color: @grey-light;
}

.typ12-400-grey-dark {
    font-size: 12px;
    font-weight: 400;
    color: @grey-dark;
}

.typ14-400-black {
    font-size: 14px;
    font-weight: 400;
    color: @black;
}

.typ14-400-white {
    font-size: 14px;
    font-weight: 400;
    color: @white;
}
```

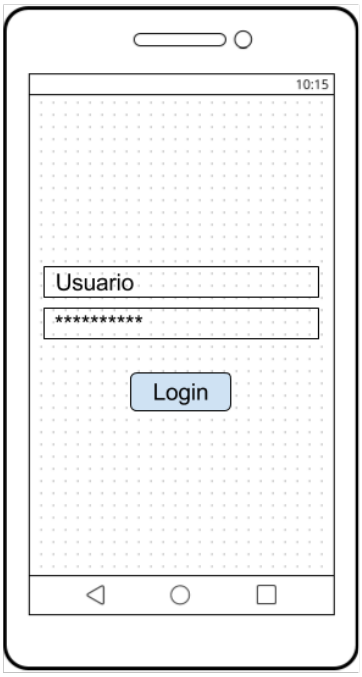
Fragmento 22 – Definición de Tipografías LESS

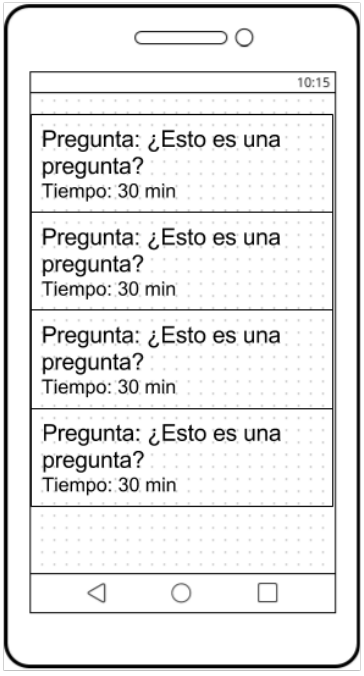
```
@background: #E6F3F7;
@topbar: #143541;
@border: #3997B9;
@black: #000000;
@white: #FFFFFF;
@border-danger: #E12523;
@danger-dark: #D10808;
@danger: #F72F2F;
@blue-dark: #3284A0;
@blue-medium: #41A2C4;
@blue-light: #A1D1E2;
@secondary: #AAD5E5;
@secondary-dark: #6FB8D3;
@grey-dark: #6D6D6D;
@grey-medium: #A9A9A9;
@grey-light: #D4D4D4;
```

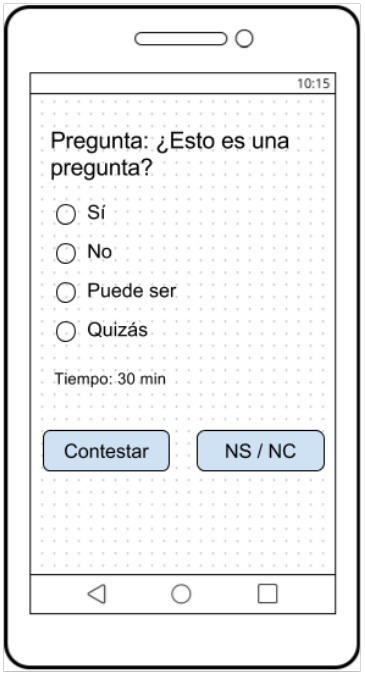
Fragmento 23 – Definición de Constantes de Colores

D. Maquetas Aplicación Android

A continuación mostramos las maquetas que se entregaron para la aplicación Android:

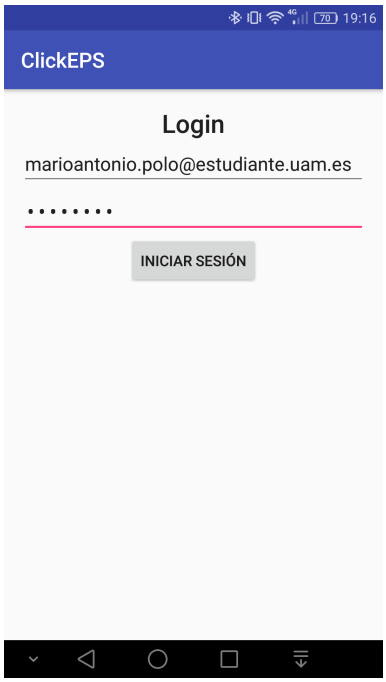


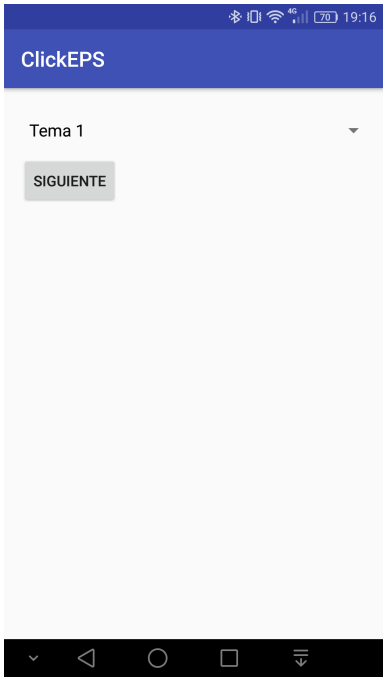


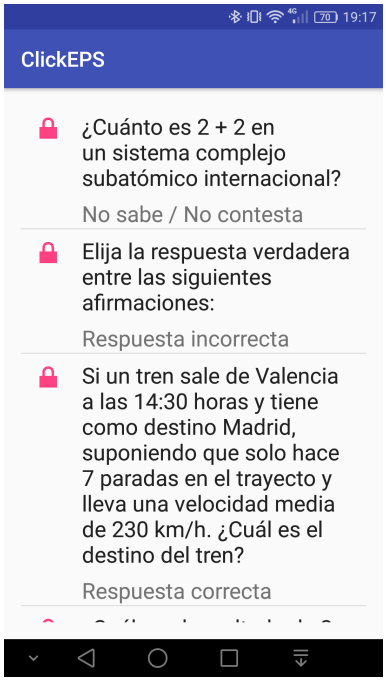


E. Vistas finales de la Aplicación Android

En este apartado vemos el aspecto final de la aplicación Android:

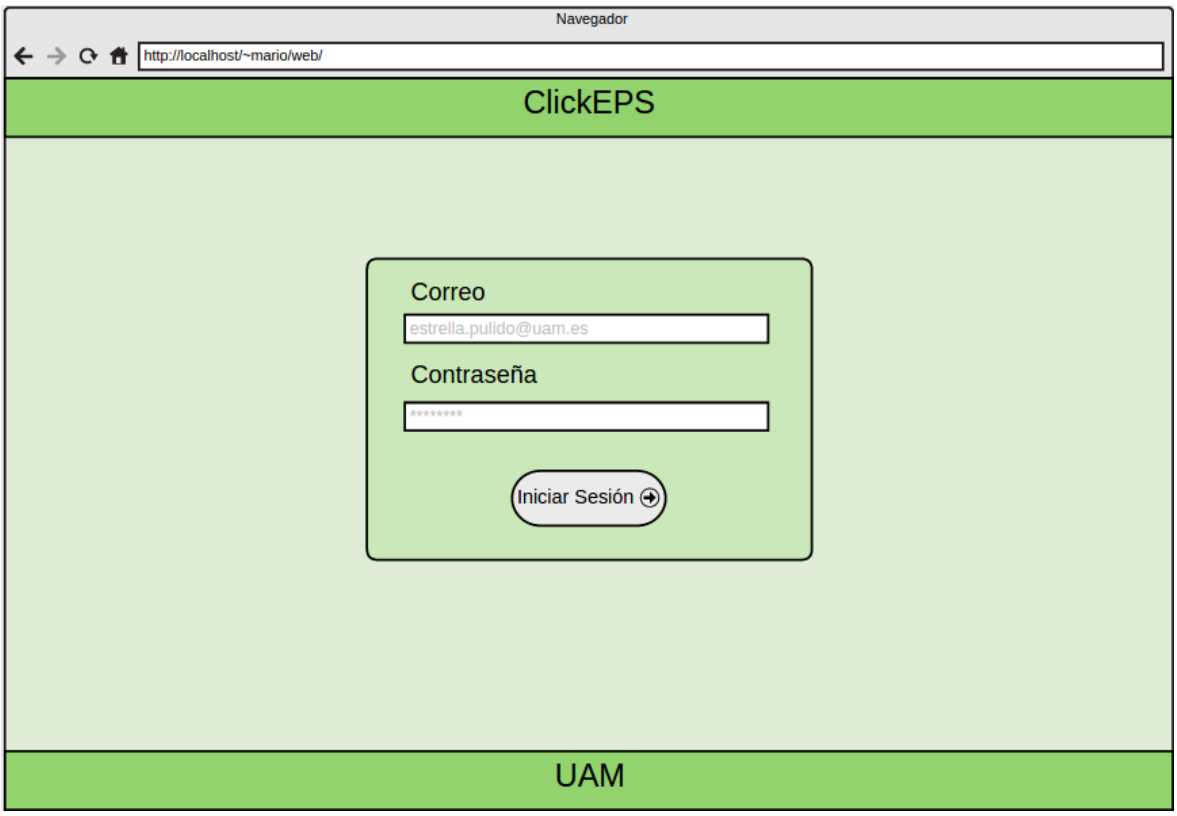


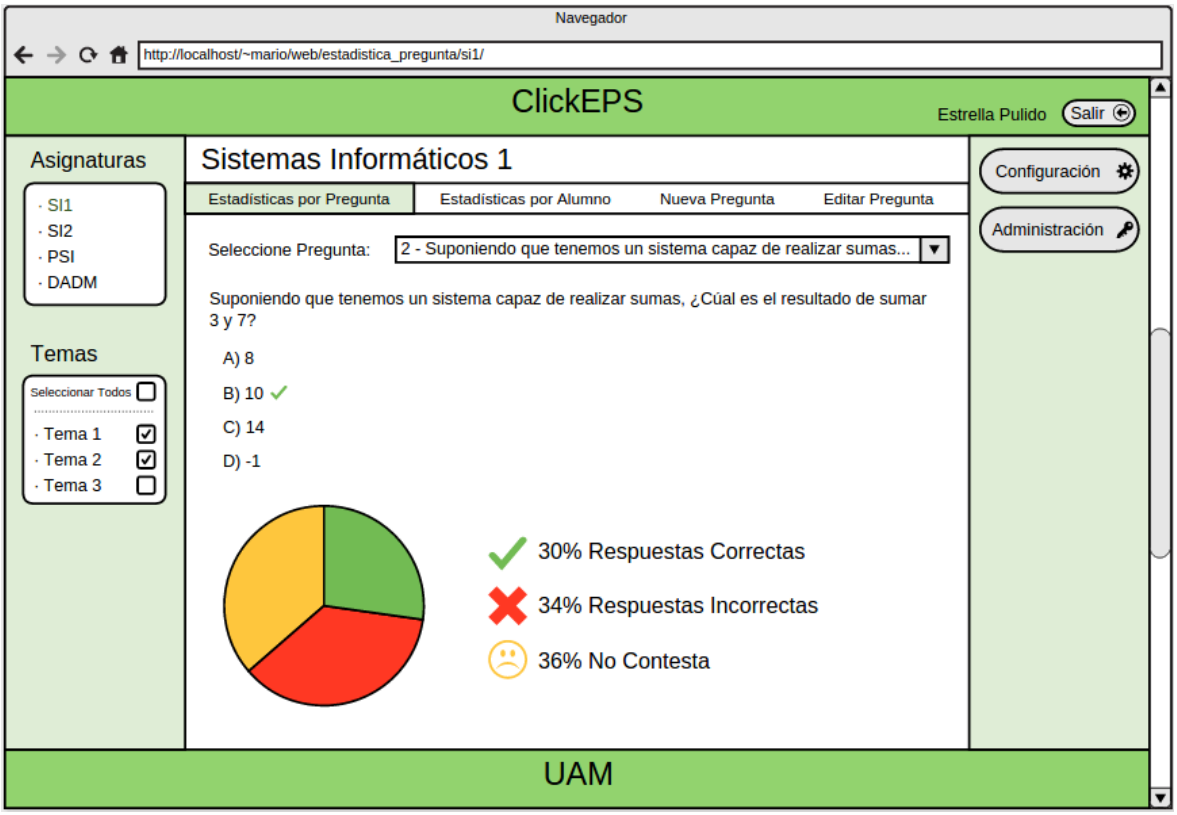


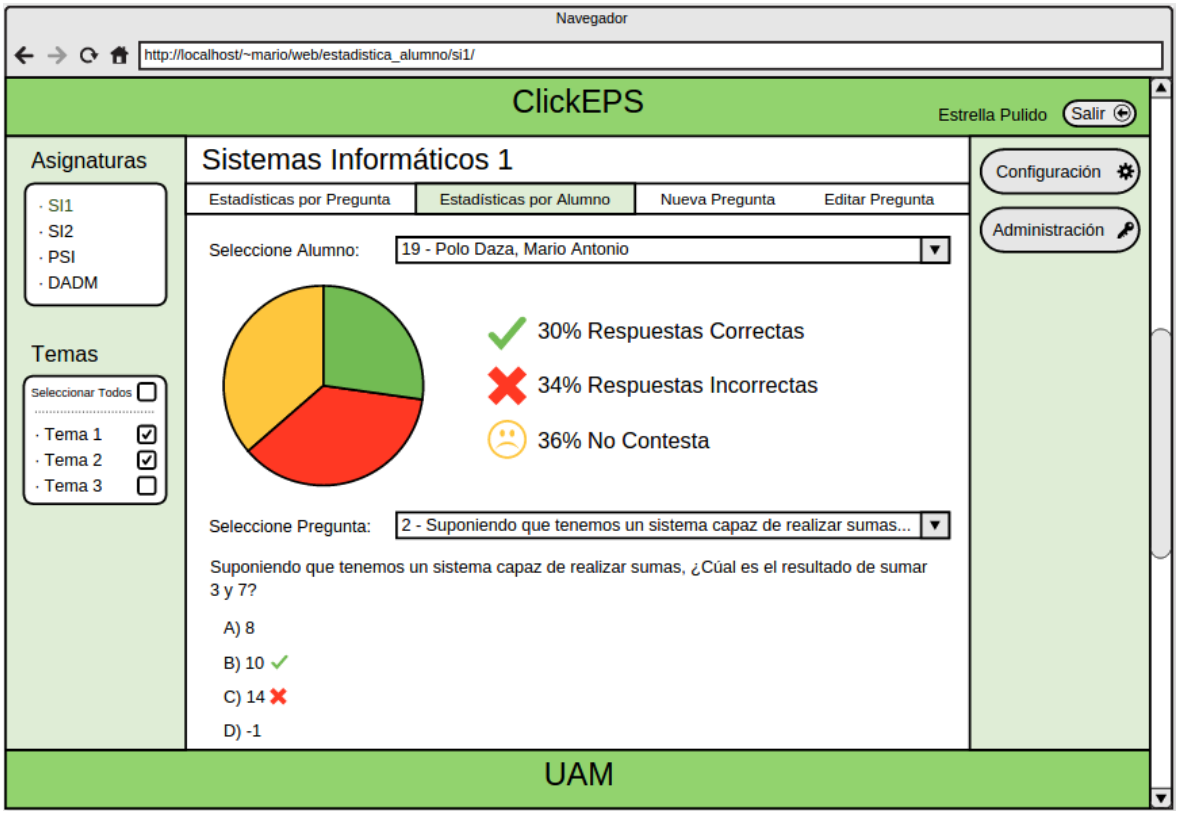


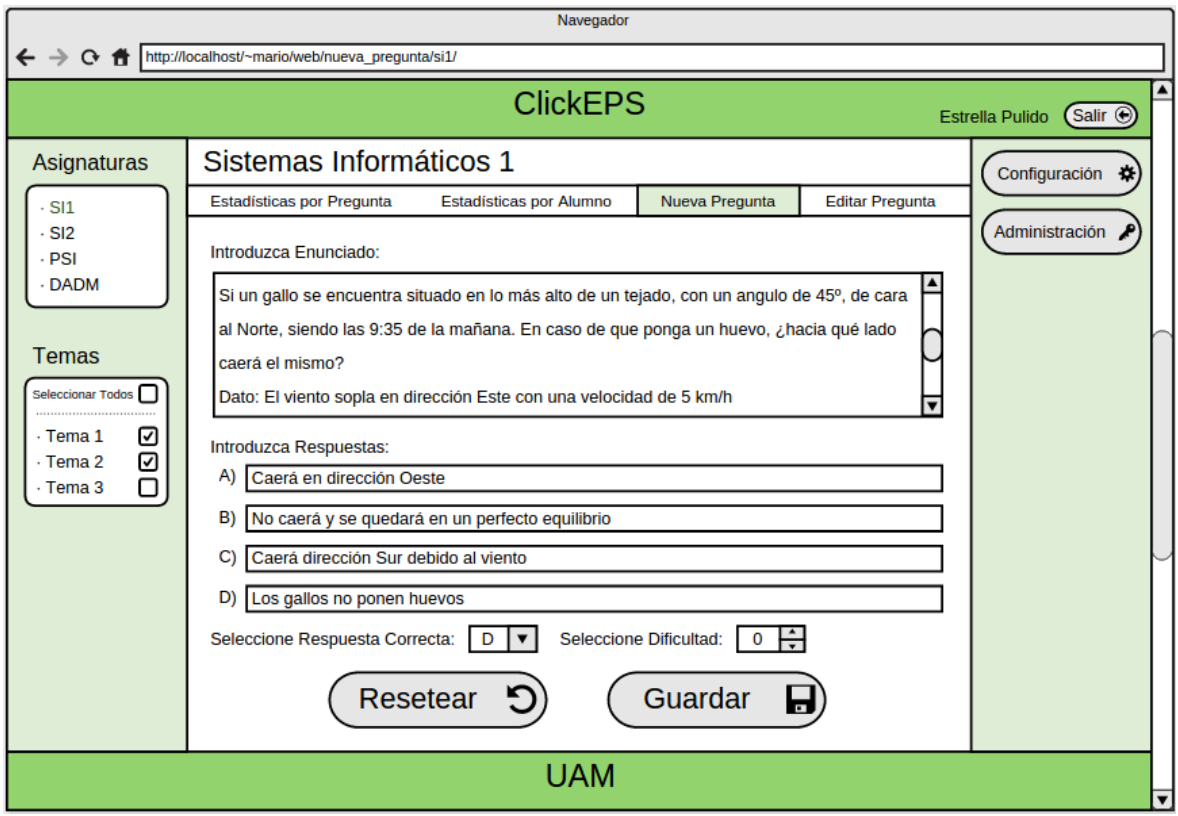
F. Maquetas Aplicación Web

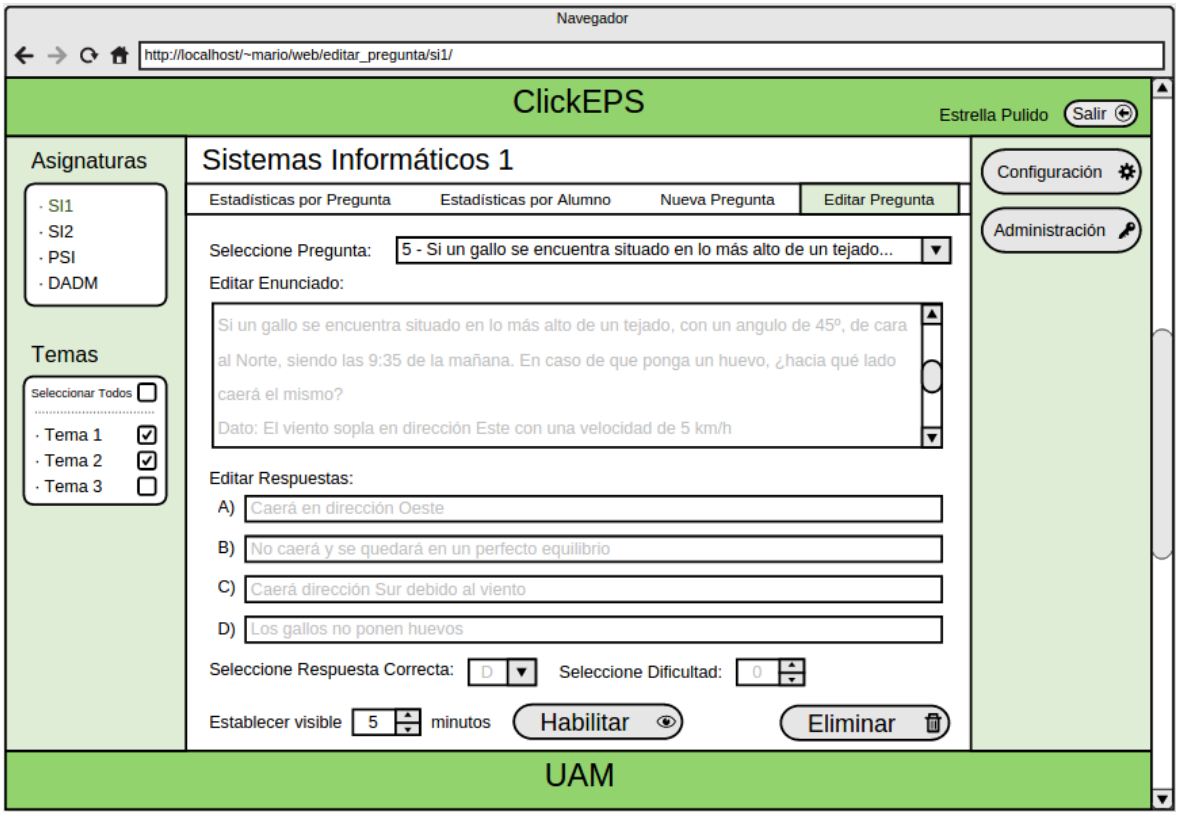
En este apartado veremos las maquetas sobre la aplicación Web:





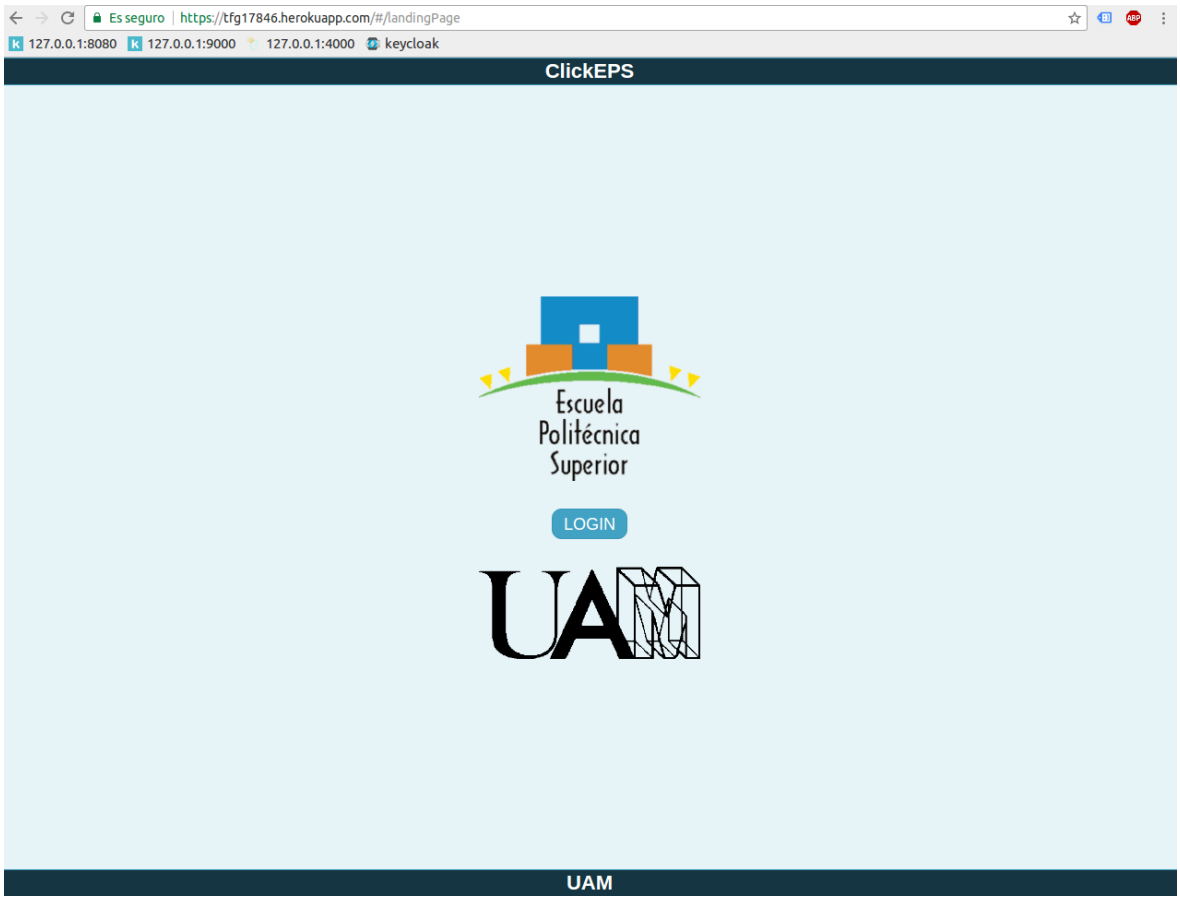

Navegador
http://localhost/~mario/web/estadistica_alumno/si1/
ClickEPS
Estrella Pulido
Salir
Asignaturas
· SI1
· SI2
· PSI
· DADM
Temas
Seleccionar Todos
· Tema 1
· Tema 2
· Tema 3
Sistemas Informáticos 1
Estadísticas por Pregunta
Estadísticas por Alumno
Nueva Pregunta
Editar Pregunta
Seleccione Alumno:
19 - Polo Daza, Mario Antonio
30% Respuestas Correctas
34% Respuestas Incorrectas
36% No Contesta
Seleccione Pregunta:
2 - Suponiendo que tenemos un sistema capaz de realizar sumas...
Suponiendo que tenemos un sistema capaz de realizar sumas, ¿Cúal es el resultado de sumar 3 y 7?
A) 8
B) 10
C) 14
D) -1
Configuración
Administración
UAM

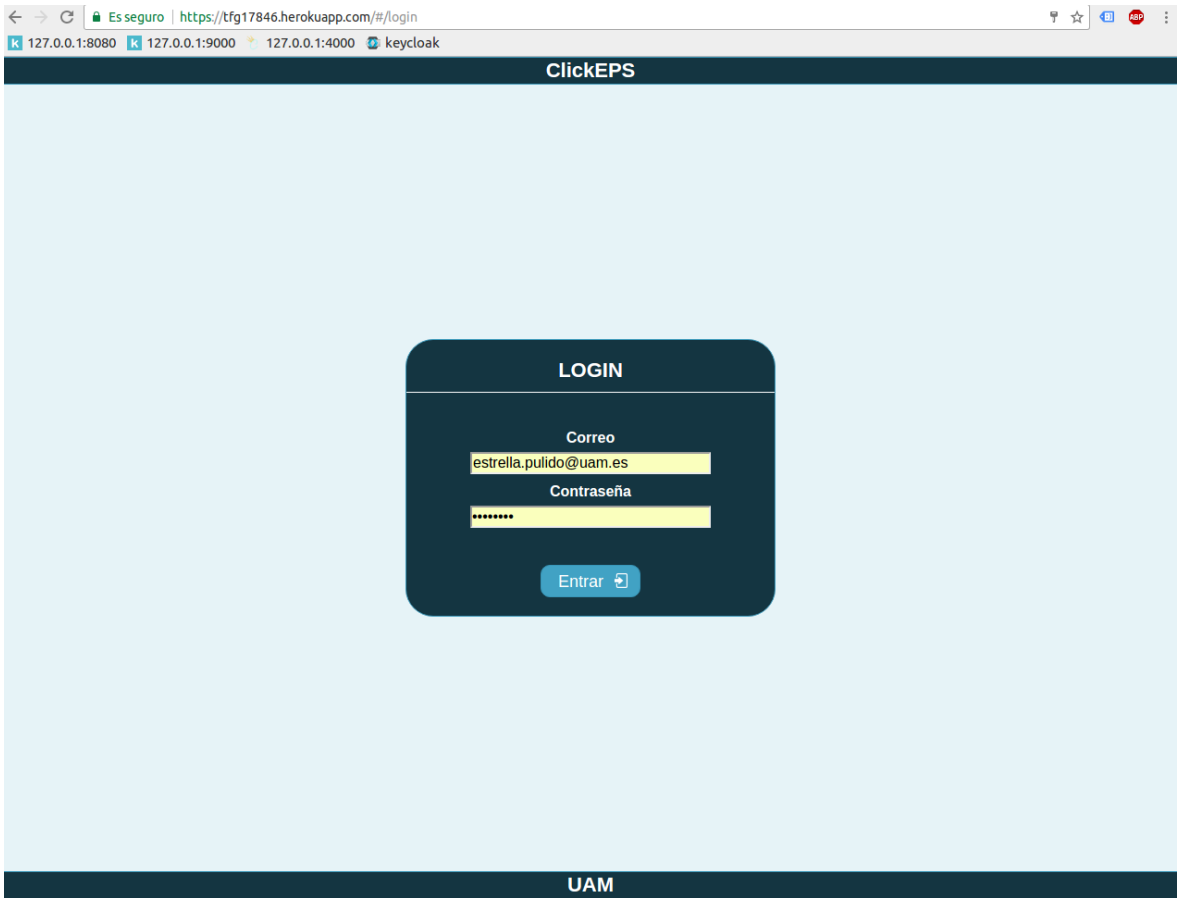

Navegador
http://localhost/~mario/web/nueva_pregunta/si1/
ClickEPS
Estrella Pulido
Salir
Asignaturas
· SI1
· SI2
· PSI
· DADM
Temas
Seleccionar Todos
· Tema 1
· Tema 2
· Tema 3
Sistemas Informáticos 1
Estadísticas por Pregunta
Estadísticas por Alumno
Nueva Pregunta
Editar Pregunta
Introduzca Enunciado:
Si un gallo se encuentra situado en lo más alto de un tejado, con un angulo de 45º, de cara al Norte, siendo las 9:35 de la mañana. En caso de que ponga un huevo, ¿hacia qué lado caerá el mismo?
Dato: El viento sopla en dirección Este con una velocidad de 5 km/h
Introduzca Respuestas:
A) Caerá en dirección Oeste
B) No caerá y se quedará en un perfecto equilibrio
C) Caerá dirección Sur debido al viento
D) Los gallos no ponen huevos
Seleccione Respuesta Correcta: D
Seleccione Dificultad: 0
Resetear
Guardar
Configuración
Administración
UAM

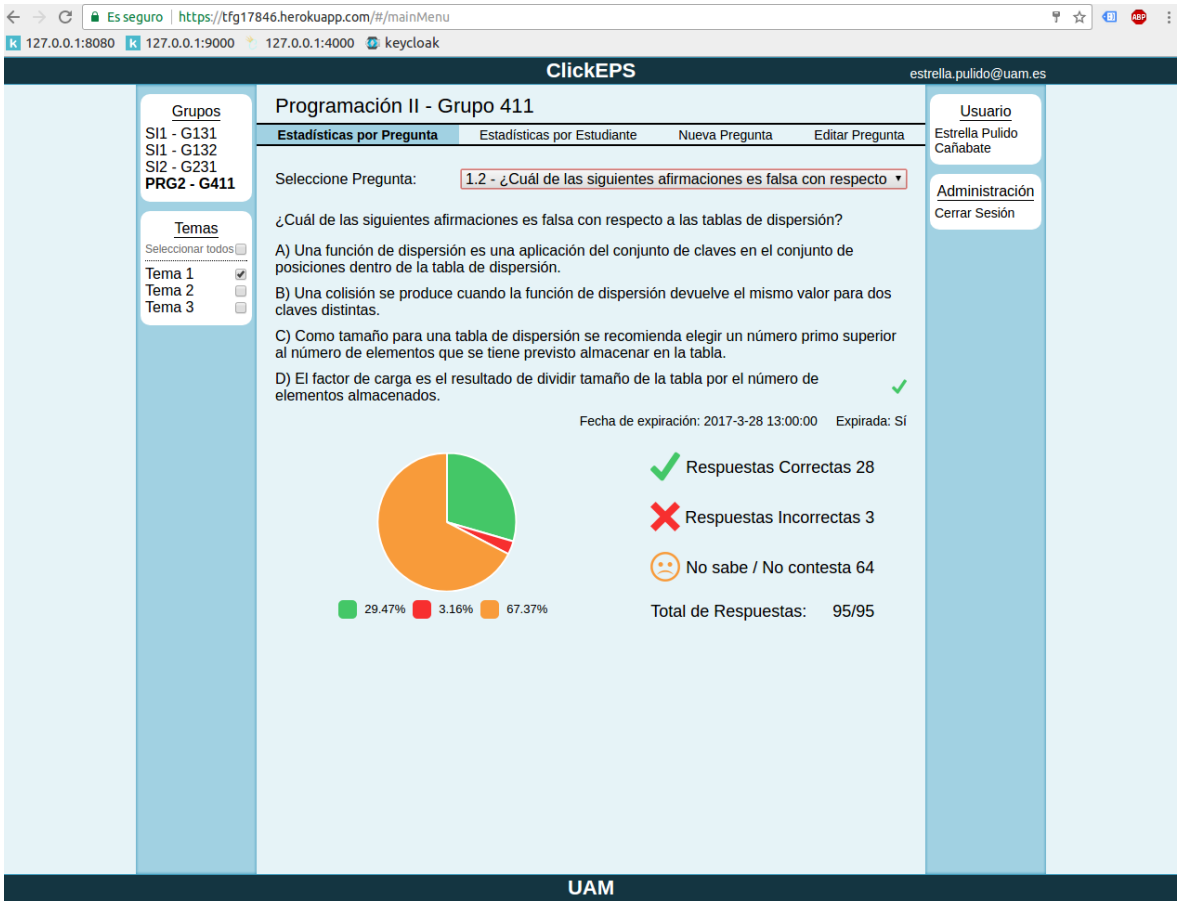

Navegador
http://localhost/~mario/web/editar_pregunta/si1/
ClickEPS
Estrella Pulido
Salir
Asignaturas
· SI1
· SI2
· PSI
· DADM
Temas
Seleccionar Todos
· Tema 1
· Tema 2
· Tema 3
Sistemas Informáticos 1
Estadísticas por Pregunta
Estadísticas por Alumno
Nueva Pregunta
Editar Pregunta
Seleccione Pregunta:
5 - Si un gallo se encuentra situado en lo más alto de un tejado...
Editar Enunciado:
Si un gallo se encuentra situado en lo más alto de un tejado, con un angulo de 45º, de cara al Norte, siendo las 9:35 de la mañana. En caso de que ponga un huevo, ¿hacia qué lado caerá el mismo?
Dato: El viento sopla en dirección Este con una velocidad de 5 km/h
Editar Respuestas:
A) Caerá en dirección Oeste
B) No caerá y se quedará en un perfecto equilibrio
C) Caerá dirección Sur debido al viento
D) Los gallos no ponen huevos
Seleccione Respuesta Correcta:
D
Seleccione Dificultad:
0
Establecer visible
5
minutos
Habilitar
Eliminar
Configuración
Administración
UAM

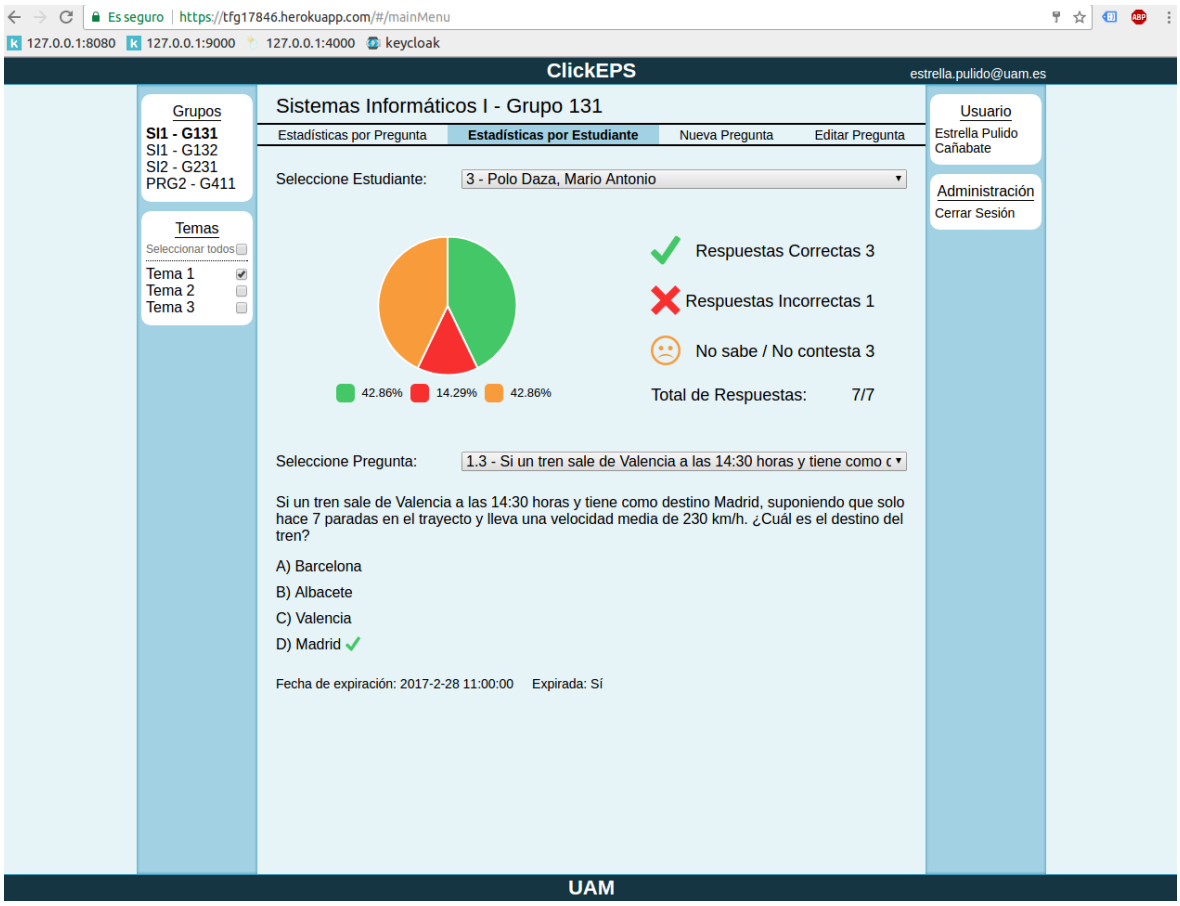
G. Vistas finales de la Aplicación Web

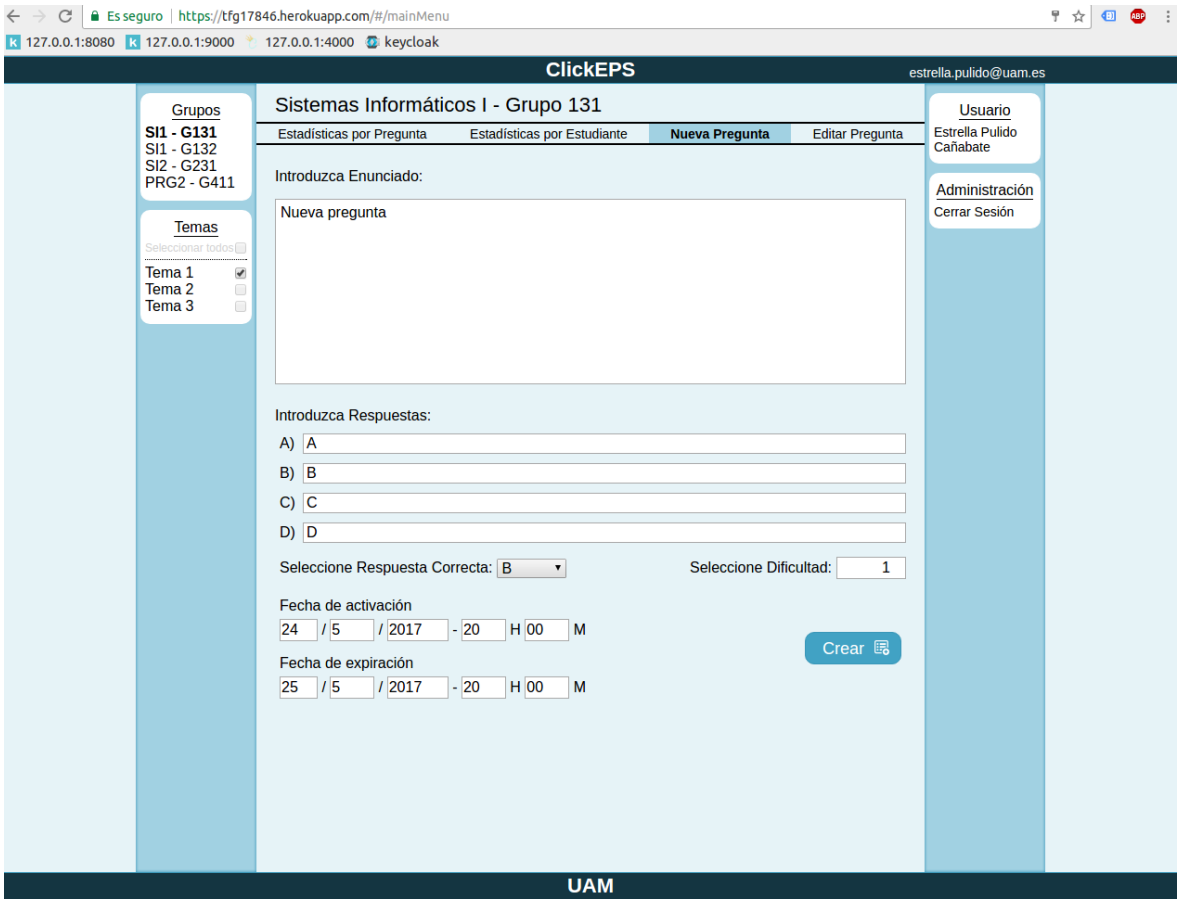
Y finalmente este es el aspecto de la aplicación Web:

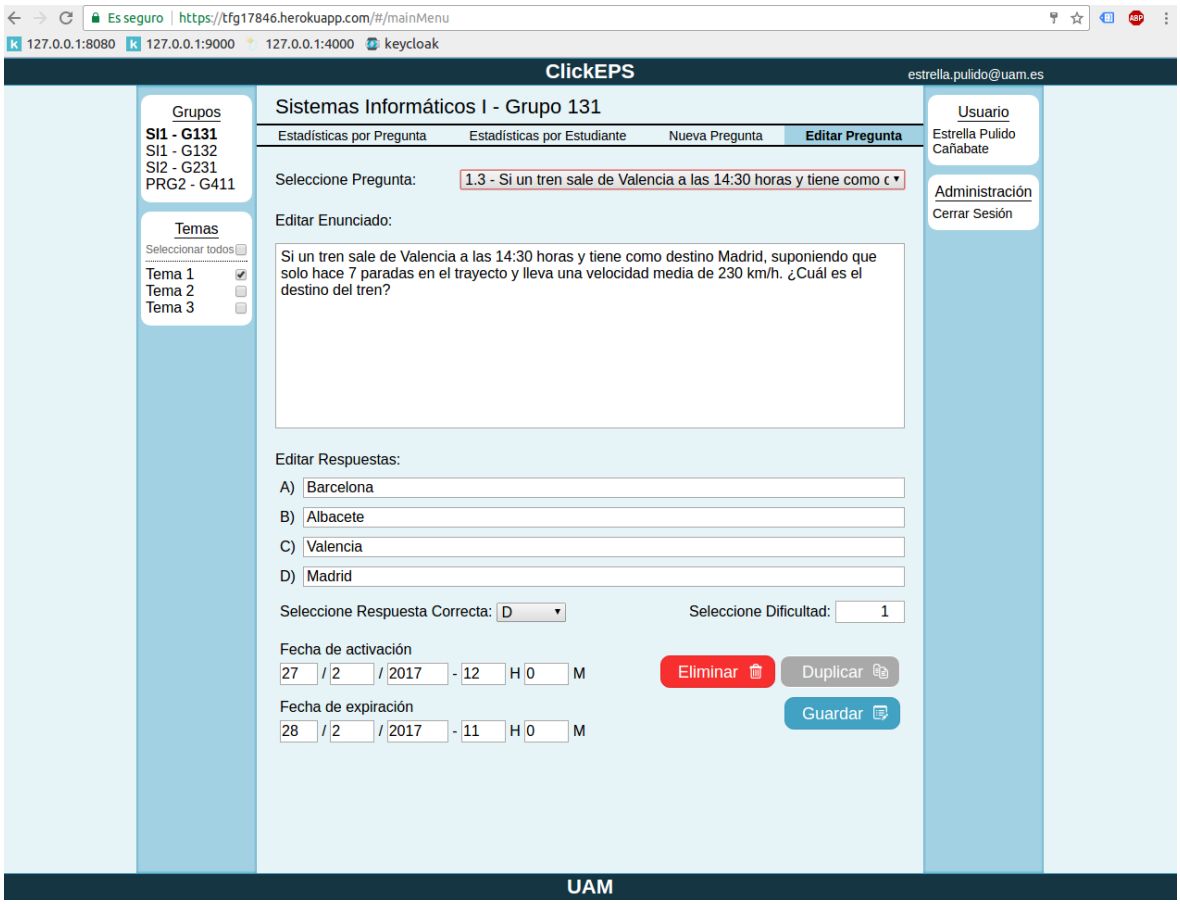



Es seguro | https://tfg17846.herokuapp.com/#/login
127.0.0.1:8080 127.0.0.1:9000 127.0.0.1:4000 keycloak
ClickEPS
LOGIN
Correo
estrella.pulido@uam.es
Contraseña
Entrar
UAM


Es seguro | https://tfg17846.herokuapp.com/#/mainMenu
127.0.0.1:8080 127.0.0.1:9000 127.0.0.1:4000 keycloak
ClickEPS
estrella.pulido@uam.es
Grupos
SI1 - G131
SI1 - G132
SI2 - G231
PRG2 - G411
Temas
Seleccionar todos
Tema 1
Tema 2
Tema 3
Programación II - Grupo 411
Estadísticas por Pregunta
Estadísticas por Estudiante
Nueva Pregunta
Editar Pregunta
Seleccione Pregunta:
1.2 - ¿Cuál de las siguientes afirmaciones es falsa con respecto
¿Cuál de las siguientes afirmaciones es falsa con respecto a las tablas de dispersión?
A) Una función de dispersión es una aplicación del conjunto de claves en el conjunto de posiciones dentro de la tabla de dispersión.
B) Una colisión se produce cuando la función de dispersión devuelve el mismo valor para dos claves distintas.
C) Como tamaño para una tabla de dispersión se recomienda elegir un número primo superior al número de elementos que se tiene previsto almacenar en la tabla.
D) El factor de carga es el resultado de dividir tamaño de la tabla por el número de elementos almacenados.
Fecha de expiración: 2017-3-28 13:00:00 Expirada: Sí
29.47% 3.16% 67.37%
Respuestas Correctas 28
Respuestas Incorrectas 3
No sabe / No contesta 64
Total de Respuestas: 95/95
Usuario
Estrella Pulido Cañabate
Administración
Cerrar Sesión
UAM


Es seguro | https://tfg17846.herokuapp.com/#/mainMenu
127.0.0.1:8080 127.0.0.1:9000 127.0.0.1:4000 keycloak
ClickEPS
estrella.pulido@uam.es
Grupos
SI1 - G131
SI1 - G132
SI2 - G231
PRG2 - G411
Temas
Seleccionar todos
Tema 1
Tema 2
Tema 3
Sistemas Informáticos I - Grupo 131
Estadísticas por Pregunta
Estadísticas por Estudiante
Nueva Pregunta
Editar Pregunta
Seleccione Estudiante:
3 - Polo Daza, Mario Antonio
42.86%
14.29%
42.86%
Respuestas Correctas 3
Respuestas Incorrectas 1
No sabe / No contesta 3
Total de Respuestas: 7/7
Seleccione Pregunta:
1.3 - Si un tren sale de Valencia a las 14:30 horas y tiene como c
Si un tren sale de Valencia a las 14:30 horas y tiene como destino Madrid, suponiendo que solo hace 7 paradas en el trayecto y lleva una velocidad media de 230 km/h. ¿Cuál es el destino del tren?
A) Barcelona
B) Albacete
C) Valencia
D) Madrid
Fecha de expiración: 2017-2-28 11:00:00 Expirada: Sí
Usuario
Estrella Pulido Cañabate
Administración
Cerrar Sesión
UAM


Es seguro | https://tfg17846.herokuapp.com/#/mainMenu
127.0.0.1:8080 127.0.0.1:9000 127.0.0.1:4000 keycloak
ClickEPS
estrella.pulido@uam.es
Grupos
SI1 - G131
SI1 - G132
SI2 - G231
PRG2 - G411
Temas
Seleccionar todos
Tema 1
Tema 2
Tema 3
Sistemas Informáticos I - Grupo 131
Estadísticas por Pregunta
Estadísticas por Estudiante
Nueva Pregunta
Editar Pregunta
Introduzca Enunciado:
Nueva pregunta
Introduzca Respuestas:
A) A
B) B
C) C
D) D
Seleccione Respuesta Correcta: B
Seleccione Dificultad: 1
Fecha de activación
24 / 5 / 2017 - 20 H 00 M
Fecha de expiración
25 / 5 / 2017 - 20 H 00 M
Crear
Usuario
Estrella Pulido Cañabate
Administración
Cerrar Sesión
UAM


Es seguro | https://tfg17846.herokuapp.com/#/mainMenu
127.0.0.1:8080
127.0.0.1:9000
127.0.0.1:4000
keycloak
ClickEPS
estrella.pulido@uam.es
Grupos
SI1 - G131
SI1 - G132
SI2 - G231
PRG2 - G411
Temas
Seleccionar todos
Tema 1
Tema 2
Tema 3
Sistemas Informáticos I - Grupo 131
Estadísticas por Pregunta
Estadísticas por Estudiante
Nueva Pregunta
Editar Pregunta
Seleccione Pregunta:
1.3 - Si un tren sale de Valencia a las 14:30 horas y tiene como c
Editar Enunciado:
Si un tren sale de Valencia a las 14:30 horas y tiene como destino Madrid, suponiendo que solo hace 7 paradas en el trayecto y lleva una velocidad media de 230 km/h. ¿Cuál es el destino del tren?
Editar Respuestas:
A) Barcelona
B) Albacete
C) Valencia
D) Madrid
Seleccione Respuesta Correcta: D
Seleccione Dificultad: 1
Fecha de activación
27 / 2 / 2017 - 12 H 0 M
Fecha de expiración
28 / 2 / 2017 - 11 H 0 M
Eliminar
Duplicar
Guardar
Usuario
Estrella Pulido Cañabate
Administración
Cerrar Sesión
UAM