UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

GEA: Plataforma para la gamificación de sesiones magistrales

Jorge Fernández Peláez

Febrero, 2019

GEA: PLATAFORMA PARA LA GAMIFICACIÓN DE SESIONES MAGISTRALES

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de Información

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

GEA: Plataforma para la gamificación de sesiones magistrales

Autor: Jorge Fernández Peláez

Director: Carlos González Morcillo

Director: Santiago Sánchez Sobrino

Febrero, 2019

Jorge Fernández Peláez

Ciudad Real – Spain

E-mail académico: jorge.fernandez7@alu.uclm.es
E-mail personal: jorge.fernandez24994@gmail.com

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

Desde el origen de las primeras universidades en el siglo XI, las clases magistrales se configuran como uno de los principales métodos tradicionales de enseñanza universitaria. En este modelo docente, el docente realiza la explicación de los conceptos relacionados con la temática de la asignatura y los estudiantes atienden. Las clases magistrales están profundamente arraigadas en todos los niveles educativos y son el exponente de una verdad casi universal: los conocimientos se adquieren a partir de una explicación por parte de quien los tiene. Sin embargo, el papel pasivo del estudiante en este enfoque pedagógico tiene sus consecuencias en el bajo rendimiento de los alumnos.

El presente Trabajo Fin de Grado (GEA) surge como una posible solución a estos problemas. En este proyecto se plantea la construcción de un sistema que permita aplicar la gamificación a las sesiones magistrales de los docentes, que puedan servir como elemento complementario junto con seminarios de problemas, sesiones de laboratorio o trabajos en grupo. Antes que prohibir a los alumnos el uso de las nuevas tecnologías como *smartphones*, se les anima a utilizarlas en las aulas, bajo supervisión del profesor, para así conseguir dinamizar las mismas e incrementar el interés y la participación.

En GEA los docentes contarán con una herramienta para fortalecer los niveles de atención de los alumnos, introduciendo diversos puntos de participación por parte de los alumnos. Las preguntas añadidas a la docencia magistral añade un nivel de interacción básico fundamental en el modelo constructivista, donde el estudiante asimila los conceptos y desarrolla un modelo mental que es perfeccionado según avanza el temario.

Abstract

Since the origins of the first universities in the 11th century, master classes have been one of the main traditional methods of university teaching. In this teaching model, the teacher explains the concepts related to the subject and students pay attention. Master classes are deeply rooted in all levels of education and are the exponent of a quasi-universal truth: knowledge is acquired from an explanation on the part of those who have it. However, the passive role of the student in this pedagogical approach has its consequences on the low performance of the students.

The present dissertation emerges as a possible solution to these problems. In this project the building of a system that allows to apply gamification to master sessions is proposed. It can serve as complementary element together with seminars of problems, laboratory sessions or works in group. Rather than prohibiting students from using new technologies such as smartphones, they are encouraged to use them in classrooms, under the supervision of the teacher, in order to make them more dynamic and increase interest and participation.

In GEA, teachers will have a tool to strengthen student attention levels, introducing various points of participation by students. The questions added to the masterclass add a basic level of interaction fundamental in the constructivist model, where the student assimilates the concepts and develops a mental model that is perfected as the agenda advances.

Agradecimientos

En primer lugar, me gustaría agradecer el apoyo y confianza que he tenido por parte del equipo de Furious Koalas, en especial a Carlos, David y Santiago. Desde el principio hasta al final de mi etapa con ellos, siempre han contando conmigo, pudiendo trabajar además en la maquetación del material del Curso de Videojuegos y conociendo a grandes profesionales del sector. Depositan mucho esfuerzo en todo lo que hacen, a veces no apreciado o visto por los alumnos y si el curso está tan bien reconocido, en gran parte es gracias a ellos.

A mis padres, abuelos, hermanos y hermana, cuñado y cuñada, mostrando su preocupación por mi bienestar mientras hacía este trabajo, pues no todos los momentos han sido buenos y me han proporcionado el apoyo que necesitaba, aunque no me gustara escuhar lo que decían, era necesario y eso te hace crecer, no sólo como profesional del sector, sino como persona, lo cual considero aún más importante.

A mis tios, Pepe y Javi, dos grande influencias para mi y un ejemplo a seguir. Solo tienes una vida y es corta, pero es tuya y hay que aprovecharla al máximo. Gracias por eso.

A mis amigos, más concretamente Dani, Lazmy y Penélope. Son pocas las probabilidades que tienes en la vida de encontrar a personas que te quieran tanto como ellos, que entiendan por lo que estás pasando y te den su apoyo incondicional aunque tropieces dos veces con la misma piedra. Os quiero mucho.

Y a Stack Overflow.

Jorge

A mi querido calvo gruñón

Índice general

Índice de tablas

Índice de figuras

Índice de listados

Listado de acrónimos

CRS	Classrom Response System
GAE	Google App Engine
GEA	Gamify 'Em All
PUD	Proceso Unificado de Desarrollo
MVC	Modelo Vista Controlador

Capítulo 1

Introducción

La evolución de la informática, así como del acceso a Internet en hogares y/o colegios, ha permitido la aparición de nuevos sistemas relacionados con la educación que o promueven nuevas formas de enseñanza o mejoran las ya existentes. Una de estas nuevas tecnologías son los denominados Classrom Response System (CRS) o sistemas de respuesta en clase. Estos sistemas suelen estar destinados a promover la participación por parte de los alumnos, haciendo así las clases más dinámicas e interactivas.

Los CRS hacen uso de los llamados *clickers* como el *iClicker*[1] o el *LCD Clicker*[2]. Estos *clickers* permiten a los alumnos o usuarios, responder a las preguntas propuestas por el profesor, generalmente de manera anónima, enviado la respuesta a un receptor que está conectado al ordenador personal del profesor y así tener una retroalimentación o *feedback* de la clase o presentación que estén impartiendo.

Sin embargo, el uso de estos *clickers* implica realizar una inversión económica, que puede llegar a ser bastante alta, para comprar el hardware necesario, la cual, puede que no sea posible realizar para algunos institutos y/o profesores.

1.1 Contexto

Una de las tareas que más dificil le resulta a los profesores de realizar en la docencia, es la de conseguir motivar y promover la participación de sus alumnos en sus clases magistrales, así como en presentaciones que puedan realizar. Es por esto, que los CRS han comenzado a ganar popularidad entre los profesores en los últimos años.[FM06]

Antes de la aparición de los mismos, era recurrente que los profesores utilizaran otras técnicas para involucrar a sus alumnos como, votaciones a mano alzada, realización de trabajos en grupos, etc. Es en el año 1996 cuando se hacen las primeras referencias a este tipo de sistemas, aunque llamados Sistemas de Comunicación en Clase (*Classroom Communication Systems*). Dufresne[DGL+96], desprestigia la enseñanza tradicional, basada en que los alumnos escuchen pasivamente la presentación del profesor sin participar en ella y defiende una enseñanza en la que el alumno tenga un papel mucho más activo.

[1] https://www.iclicker.com/

[2] https://www.turningtechnologies.com/lcd/

Tras esto, fueron varios los CRS que fueron apareciendo a lo largo de la literatura pero con distintos nombres: *audience response systems*[MAG03], *voting machines*[RBL+05] o *electronic response systems* [JS02]. A pesar de que el nombre de estos fuera distinto, una reciente investigación de los mismos, concluyó que los productos comerciales que proporcionaban eran bastante similares[BL03].

La aparición de los *smartphones* fue un hito importante dentro del campo de la informática, y como tal, también afecto a los CRS. Como ya se ha explicado, para realizar este tipo de actividades, era necesario tener hardware extra en las aulas o lugares donde se fueran a utilizar: los emisores de respuesta o *clickers* utilizados por los alumnos, el receptor por parte del profesor y un proyector o similar para poder mostrar la presentación. Ahora, utilizando los *smartphone*, el ordenador personal del profesor y un programa extra como podría ser una plataforma web, se puede transformar fácilmente un aula convencional en un CRS.

Aprovechando esta facilidad, son varias las plataformas web que han sido desarrolladas emulando los CRS, como por ejemplo *Kahoot!*, *Pear Deack* o *Quizziz*.

1.2 Entorno

Gamify 'Em All (GEA) resuelve la problemática de un CRS accesible a la mayoría de colegios, institutos o universidades, independientemente de los recursos económicos que tengan. Aprovechando el auge de los *smarthphones* y teniendo en cuenta que la mayoría de los alumnos tienen uno a edades tempranas (a partir de los 12 años, 3 de cada 4 niños poseen uno), GEA supone una ventaja frente a los CRS tradicionales ya que no requiere de dispositivos adicionales como *clickers* o receptores para las respuestas.

El entorno de ejecución está compuesto de un proyector, utilizado como soporte para la visualización general de la sesión el aula, la plataforma web donde se almacenarán las distintas sesiones y se le dará acceso a los alumnos a las mismas y los propios *smarthphones* de los alumnos. En este entorno, se distinguen dos roles, el **rol del profesor**, encargado de preparar las sesiones, dar paso a los alumnos y dinamizar las mismas utilizando la plataforma web y el **rol del alumno**, que simplemente responderá a las preguntas planteadas por el profesor utilizando su *smartphone* (ver figura 1.1)

Figura 1.1: Diagrama de componentes y roles que intervienen en GEA

Las tareas que realiza cada uno de los distintos elementos del entorno son las siguientes:

- **Plataforma Web**: será el núcleo principal del sistema. Encargada de almacenar los distintos profesores, con sus sesiones asociadas en una base de datos, y de permitir el acceso a los alumnos a las sesiones creadas por los profesores mediante el uso de un PIN. Además, al finalizar cada sesión, se proporcionará un informe a modo de *feedback* al profesor con las respuestas de cada pregunta hecha por cada alumno.
- **Rol profesor**: aunque no sea un componente como tal del sistema, será el profesor el encargado de preparar las sesiones en plataforma web para posteriormente utilizarlas en sus clases. Ademas, con el *feedback* obtenido, puede derivar el flujo de la sesión si lo tenía previamente preparado.
- **Rol alumno**: al igual que con el rol del profesor, no es un componente del sitema, pero será el alumno el encargado de contestar a las preguntas utilizando su *smarthpone*. La pregunta y las distintas respuestas se mostrarán en el dispositivo del alumno, aunque se le dará la opción al profesor de mostrarlas por un proyector también.

1.3 Estructura del documento

El presente documento se ha estructurado según las indicaciones de la normativa de trabajos fin de grado de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha, empleando los siguientes capítulos

Capítulo 2: Objetivos

En este capítulo se describen y desglosa la lista de objetivos y subobjetivos planteados para este TFG.

Capítulo 3: Antecedentes

En este capítulo se hace un repaso a los conocimientos y campos que han sido necesarios estudiar para el desarrollo de GEA. Es decir, se presenta un *estado del arte* de las tecnologías ya existentes y que abarcan las distintas áreas de este proyecto.

Capítulo 4: Metodología

En este capítulo se explica y se justifica la metodología exogida para el desarrollo

de este sistema. Además, se describen los recursos, tanto hardware como software, utilizados para la realización de este sistema.

Capítulo 5: Arquitectura

En este capítulo se destallan los pasos y las decisiones que se han tomado y que han sido necesarias para poder desarrollar el sistema.

Capítulo 6: Resultados

En este capítulo se enumeran y explican los resultados que se han obtenido del desarrollo obtenido. Se exponen también los costes económicos del sistema.

Capítulo 7: Conclusiones

En este capítulo se resume a modo de conclusión el desarrollo del sistema y los objetivos alcanzados. También se plantean una sería de líneas de trabajo futuro para continuar con el desarrollo del proyecto.

Capítulo 2

Objetivos

UNA vez presentadas las ideas y desarrollado el contexto en la introducción, en este capítulo se concreta qué se pretenda llevar acabo, sintetizando primero el objetivo principal y pasando a detallar el conjunto de objetivos específicos derivados.

2.1 Objetivo general

Como se ha explicado en la Sección 1, la aparición de nuevas tecnologías como los *smartphones* ha provocado que sea más difícil mantener la atención continuada por parte de los alumnos. La cantidad de estímulos y notificaciones que reciben en tiempo real continuamente compite con la atención del profesor. El objetivo de este proyecto es dotar a los profesores de una herramienta web para poder aplicar técnicas de gamificación en sus sesiones con el fin de dinamizar las clases y aumentar la participación por parte de los alumnos, utilizando sus *smartphones* u ordenadores.

Por lo tanto, el objetivo principal del proyecto sería el **desarrollo de una plataforma para la gamificación de clases magistrales**, de modo que se permita incluir en el flujo de una presentación diferentes tipos de encuestas (preguntas, sondeos y cuestionarios).

2.2 Objetivos específicos

A partir del objetivo principal descrito en la sección anterior, se presentan los distintos subobjetivos que se deben cumplir en los siguientes apartados.

2.2.1 Implementación básica del sistema web

La plataforma que se desarrollará para la aplicación de técnicas de gamificación será un sistema web, parecido a un CRS, por lo que se deben acometer las siguientes tareas:

- Agregar un servicio de terceros que permita el registro y/o autenticación de usuarios.
- Gestionar correctamente las sesiones de cada usuario (creación, asignación, recuperación y cierre).
- Sistema de roles de manera que cada uno de ellos tenga acceso a zonas determinadas del sistema. Los profesores a la creación y administración de las sesiones, y los

alumnos a la realización de las mismas.

- Realizar un diseño de la interfaz *responsive* de manera que se asegure la correcta visualización en distintos dispositivos.

2.2.2 Mejora de la interacción entre profesor y alumno

El sistema deberá proporcionar herramientas para poder mejorar la comunicación e interacción entre alumno y profesor, por lo que se tendra que:

- Desarrollar un formulario amigable para la inclusión de nuevas sesiones.
- Mostrar las sesiones a los alumnos de manera llamativa y con dinamismo.
- Proporcionar *feedback* en tiempo real al profesor de las respuestas de los alumnos.

2.2.3 Tiempo variable entre interacciones

La herramenta le facilitará al profesor mecanismos para poder tener el control de la sesión:

- Elegir en qué momento se pasa a la siguiente pregunta.
- Dar la opción al profesor de elegir en cualquier momento qué preguntas serán las siguientes a mostrar.
- Recibir en tiempo real las respuestas de los alumnos.

2.2.4 Generación de informes

Al finalizar cada sesión, la plataforma web proporcionará informes al profesor, de manera que deberá de ser capaz de:

- Recoger la respuesta de cada alumno en cada pregunta, de una sesión determinada, así como la puntuación total.
- Mostrar los resultados en tiempo real de todos los alumnos unidos a una sesión en concreto.
- Dar la opción de descargar un fichero con los resultados de los alumnos.

Capítulo 3

Antecedentes

Los principales campos en los que se basa GEA son la **gamificación**, las **tecnologías web** y los **sistemas de respuesta en aulas** o CRS. Para poder cumplir con el objetivo principal del sistema propuesto, se han estudiado distintos aspectos de la gamificación, así como varias tecnologías web que nos permitan implementar de la mejor manera posible las distintas funcionalidades necesarias para el correcto funcionamiento del sistema y algunas técnicas de gamificación destinadas a la educación, más concretamente para ser aplicadas en sesiones magistrales, basándose en los CRS.

3.1 Gamificación

En los últimos años, el término gamificación (o ludificación) ha tenido una gran repercusión debido a la gran cantidad de entornos existentes, como el *marketing* o la educación. En ellos, se han utilizado mecánicas típicas de los juegos o videojuegos para conseguir una mayor participación por parte de los usuarios, haciendo que las actividades propuestas sean más entretenidas y dinámicas.

Aunque el término gamificación sea nuevo (fue en el año 2011 cuando se dio una de las primeras definiciones [DKND11]), el razonamiento principal que sigue se lleva aplicando desde hace muchos años atrás. Un ejemplo de ello sería en la educación de los más pequeños, donde se les premiaba dándoles recompensas cuando conseguían resolver un problema.

Podemos definir la *gamificación* como la **aplicación de elementos o técnicas típicas de los juegos y videojuegos en otro tipo de entornos**, con el objetivo de promover la involucración de los usuarios en dichos entornos.

Recientemente, la gamificación está siendo aplicada en el ámbito empresarial, sobre todo en los sectores con un alto componente social [Maa13] [VMK+70]. Por ejemplo, dar el título de *empleado del mes* es una técnica para el aumento de la productividad basada en gamificación. Debido a esto, se han distinguido tres tipos de gamificación [Rau13] [Ném15]:

- **Gamificación interna**: para mejorar la productividad y compromiso de los empleados.
- **Gamificación externa**: relacionado con el *marketing* y las ventas, para fidelizar y atraer nuevos clientes.

- **Gamificación para modificar comportamientos**: enfocado al cambio de comportamiento de un grupo social con una causa. Suele estar más presente en ámbitos como la educación.

Al igual que se han distinguido distintos tipos de gamificación, Yu-Kai Chou en su libro *Octalysis: Complete Gamification Framework* [Cho13] hace una clasificación de lo que él considera 8 núcleos principales de la gamificación. Esto núcleos o *core drivers* los organiza en 4 grupos, atendiendo a dos criterios distintos:

- **Núcleos del lado izquierdo y derecho del cerebro**: los núcleos agrupados en el lado izquierdo, son técnicas de gamificación que utilizan como motivación en el usuario las recompensas, unos objetivos a cumplir, un reconocimiento... mientras que los núcleos del lado derecho son técnicas centradas en la creatividad o la expresión social.
- **Núcleos de sombrero negro o blanco**: los núcleos pertenecientes al sombrero blanco son aquellos que hacen sentir al usuario *poderoso* o que tiene un objetivo que cumplir, mientras que los incluidos en el sombrero negro están destinados a generar obsesión y adicción al usuario. Un ejemplo de técnicas del núcleo del sombrero negro serían las *loot boxes*, recompensas aleatorias que han sido ya reguladas en algunos países de Europa como Bélgica debido a la ludopatía que generaban.

En la Figura 3.1 podemos ver el resultado final de dichas agrupaciones propuestas por Yu-Kai Chou.

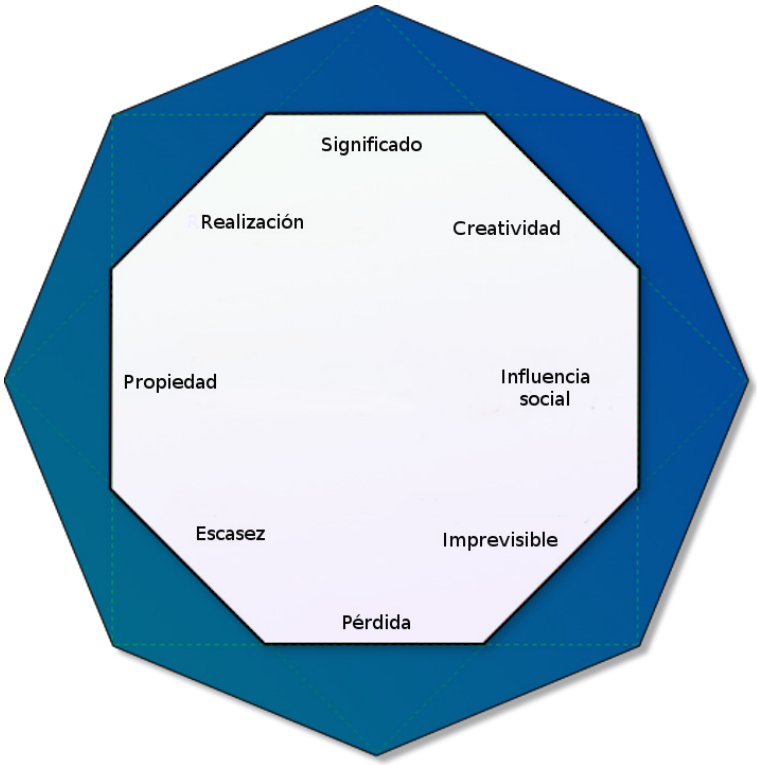

Figura 3.1: Octalysis de Yu-Kai Chou [Cho13]

Uno de los campos en los que la gamificación está teniendo una gran repercusión es en la **educación** [LH11] [Arn14]. No se puede negar que la aparición de nuevas tecnologías como pueden ser los *smartphones* y las redes sociales han creado mayores distracciones y cierto abandono por parte de los alumnos, por lo que la combinación de la gamificación con las nuevas tecnologías se ha vuelto crítica a la hora de promover el aprendizaje y participación por parte de los alumnos [Bru09][KL09a].

3.2 Tecnologías web

Los métodos por los que los ordenadores se comunican con otros ordenadores mediante el uso de lenguajes de marcas y paquetes multimedia son conocidos como **tecnologías web**.

Los modelos y tecnologías de desarrollo web han evolucionado mucho en la última década, existen multitud de aplicaciones, *frameworks*, librerías, arquitecturas y sistemas de publicación en diferentes versiones que a su vez reciben cambios o mejoran con el tiempo.

El progreso también ha tenido lugar en lo relacionado con la administración de sistemas, servicios de alojamiento, técnicas de escalabilidad, monitorización y gestión de centros de procesos de datos.

Esta evolución ha dado lugar a la convergencia de una gran cantidad de tecnologías, herramientas y estilos arquitectónicos para desarrollar sitios web y aplicaciones.

La arquitectura que siguen la mayoría de tecnologáis web es una arquitectura cliente-servidor, donde el **servidor** es una aplicación en ejecución capaz de atender las peticiones de un cliente y devolverle una respueta en concordia, y el **cliente** consume unos recursos o servicios proporcionados por el servidor.

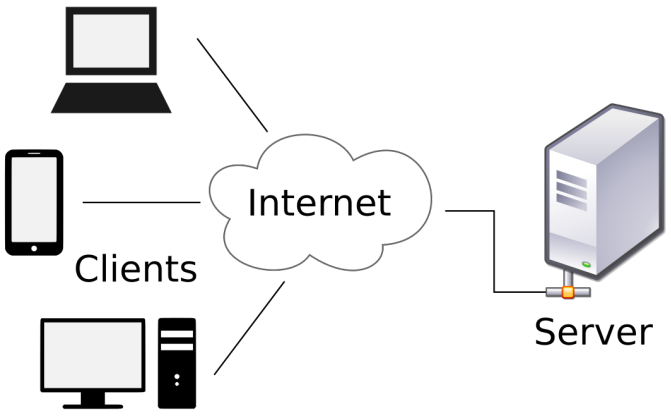

Figura 3.2: Diagrama de la arquitectura Cliente - Servidor

Siguiendo esta arquitectura, se puede hacer una abstracción de las tecnologías web existentes y separar en 2 ramas, la parte del servidor (*back-end*) y la parte del cliente (*front-end*).

3.2.1 Back-end

El *back-end* de un sitio web consta de un servidor, una aplicación y una base de datos. Un desarrollador *back-end* se encarga de construir y mantener la tecnología que impulsa aquellos componentes que, juntos, permiten que el *front-end* exista.

Para que el servidor, la aplicación y la base de datos se comuniquen entre sí, los dispositivos internos utilizan lenguajes del lado del servidor como PHP, Ruby, Python, Java y .Net para crear una aplicación, y herramientas como MySQL, Oracle y SQL Server para encontrar, guardar o cambiar datos y devolverlos al usuario en el *front-end*. Antes de comenzar a escribir código, necesitan colaborar con las partes interesadas para entender sus necesidades particulares, para poder traducirlas en requisitos técnicos y proponer la solución más efectiva y eficiente para la arquitectura de la tecnología.

Las tecnologías clásicas que se utilizaban en el desarrollo del *back-end*, eran conocidas como LAMP, el cual es un acrónimo de Linux (sistema operativo), Apache (servidor), MySQL (base de datos) y PHP (lenguaje del lado del servidor). Actualmente, las tecnologías LAMP están siendo sustituidas por las nuevas tecnologías, que suelen estar escritas en Python y JavaScript, debido a la potencia y simplicidad de estas.

Entre las nuevas tecnologías escritas con Python podemos destacar:

- **Flask**: *micro web framework* desarrollado en Python. Es un *micro framework* debido a que no trae módulos extra para abordar todas las tareas del desarrollo web, se enfoca en proporcionar lo mínimo necesario para que puedar hacer funcionar una aplicación básica en poco tiempo. Es utilizado por algunas aplicaciones como *Pinterest* y *LinkedIn*.
- **Django**: *web framework* de alto nivel, gratuito y de código abierto que fomenta el desarrollo rápido y el diseño limpio y pragmático.[1] Está fuertemente inspirado en el patrón de diseño Modelo-Vista-Controlador (MVC), pero para *Django*, el controlador es llamado vista, la cual describe qué datos serán presentados y no cómo se verán, y la vista es denominada *template*, que describen cómo los datos son presentados

Por otro lado, **Node.js** es un entorno de ejecución para JavaScript contruido con el motor de JavaScript V8 de Chrome[2]. Orientado a eventos asíncronos, está diseñado para construir aplicaciones en red escalables.

Una de las mayores ventajas que tiene Node.js con respecto al resto, es que sabiendo que JavaScript es también utilizado en el desarrollo web *front-end*, hace que solo tengamos que

[1] https://www.djangoproject.com/
[2] https://developers.google.com/v8/

utilizar un solo lenguaje para toda la aplicación web que se quiera contruir.

Google App Engine

Google App Engine (GAE) es un *framework* web y una plataforma de computación en la nube para el desarrollo y alojamiento de aplicaciones web en centros de datos gestionados por Google. GAE fue desarrollado con la intención de alojar aplicaciones en las que el número de usuarios simultáneos fuese significativo. Por esta razón, la escalabilidad en GAE es un aspecto fundamental y ofrece un servicio de escalado automático, los recursos se incrementan a la vez que el número de peticiones aumentan para una aplicación de manera invisible para el desarrollador.

GAE es gratuito hasta un cierto nivel de recursos consumidos. El desarrollador tiene la capacidad de establecer una cuota y así controlar el coste de la factura que Google emitirá a final de mes. Al contrario que ocurre con otras soluciones de hosting web, con GAE solo se paga por los recursos consumidos, entre los que se incluyen uso de CPU, almacenamiento, ancho de banda de entrada/salida, y otros servicios específicos de GAE, como por ejemplo *memcache*. Es posible incluso desplegar aplicaciones de prueba, con un uso limitado de recursos, de manera gratuita.

Desde el punto de vista del entorno de ejecución, una aplicación desplegada en GAE responde a peticiones web. Una petición web empieza en el momento en el que un cliente, comúnmente a través de un navegador web, contacta con la aplicación a través de una petición HTTP, como por ejemplo la solicitud de una página web vinculada a una determinada URL. Cuando GAE recibe la petición, identifica la aplicación gracias al nombre de dominio de la dirección, que consistirá en un subdominio .appspot.com. (proporcionado de manera gratuita por Google para cualquier aplicación) o en un subdominio registrado de manera explícita por el desarrollador. En este punto, GAE selecciona el servidor, entre los posibles, que atenderá la petición, de forma que la respuesta sea lo más rápida posible. Después, invoca a la aplicación con el contenido de la petición web, recibe los datos de respuesta de la aplicación y envía la respuesta al cliente.

Desde la perspectiva de la aplicación en GAE, el núcleo o entorno de ejecución existe cuando el manejador de peticiones o request handler arranca, mientras que desaparece cuando el manejador finaliza. En este contexto, GAE proporciona diversas opciones para mantener un estado que persista entre peticiones, pero estos mecanismos viven fuera del núcleo de ejecución de GAE. Como consecuencia de este enfoque en el que no se mantiene un estado explícito, GAE puede distribuir el tráfico entre tantos servidores como sean necesarios, que serán desplegados en función de la carga de la aplicación. A nivel global, GAE permitirá la existencia de entornos de ejecución más allá del ciclo de vida de los manejadores de peticiones, para minimizar las operaciones de inicialización y carga. Cada instancia de la aplicación mantiene memoria local para cachear código e inicializar estructuras de datos. Así, GAE

creará y destruirá instancias en función del tráfico de la aplicación. Es posible habilitar el multi-threading para manejar peticiones de manera concurrente en una misma instancia.

3.2.2 Front-end

El desarollo web del *front-end*, también conocido como desarrollo del lado del cliente, consiste en producir código HTML, CSS Y JavaScript con la finalidad de que el usuario sea capaz de interaccionar con él [Mas17]. La idea general, es que sea el *front-end* el encargado de recolectar los datos de entrada del usuario, para ajustarlos a la forma en la que el *back-end* demanda esos datos, devolviendo generalmente una respuesta que el *front-end* recibe.

El objetivo principal del diseño de un sitio web es asegurarse que cuando los usuarios entran, pueden ver la información en un formato que es fácilmente legible y relevante, lo caul resulta bastante complicado debido a la cantidad de dispositivos existentes con los cuales los usuarios acceden a los sitios web: ordenadores, *smartphones*, *tablets*... Los desarrolladores necesitan asegurarse de que el sitio se muestra bien en cualquier tipo de dispositivo, además de que funciones en distintos buscadores (*cross-browser*), sistemas operativos (*cross-platform)* y dispositivos (*cross-device*).

Existen diversas herramientas para desarrollar el *front-end* de un sitio web y entender cuáes son las mejores para cada tarea marcan la diferencia entre desarrollar una página bien diseñada y escalable. El mayor problema que supone el desarrollo del *front-end* es que las tecnologías utilizadas en el mismo están cambiando constantemente, por lo que los desarrolladores necesitan estar al día.

Lo más básico para desarrollar el *front-end* del sitio web consiste en uno o varios ficheros **HTML**, **CSS** y la programación del sitio, normalmente utilizando **JavaScript**.

- **HTML** (*Hyper Text Markup Language*): es el componente esencial de una página web, sin el cual esta no existiría. Ofrece una visión general de cómo se mostrará el sitio web. Fue desarrollado por Tim Berners-Lee en 1989 y tras esto son varias las versiones que han aparecido en el mercado de la *World Wide Web*, la última de ellas, HTML5 en Octubre de 2014.
- **CSS** (*Cascading Style Sheets*): funcionalidad básica del desarrollo *front-end*. Define el diseño de la página y le da un estilo visual único. Normalmente, un buen diseño implica que es fácil de usar, bien estructurado y una buena estética.
- **Programación**: Aunque lo más común es utilizar JavaScript, existen otros lenguajes como ActionScript o PHP. JavaScript ha evolucionado muchísimo desde comandos introducidos directamente en HTML hasta aplicaciones asíncronas ejecutadas en el buscador con grandes y variadas funcionalidades. A partir de JavaScript han surgido varias librerías como pueden ser *React*, *Angular* o *Vue.js*, de las cuales hablaremos más en profundidad a continuación.

Angular

Es un *framework* para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. La primera versión de *Angular* fue publicada en 2010, aunque en el año 2014 fue reescrita y nombrada como *Angular 2*, siendo publicada esta en el año 2016. Esto supuso un gran cambio en la dirección del desarrollo con *Angular*, ya que introdujo grandes cambios conceptuales del *framework*.

Para poder empezar a desarrollar con *Angular*, se requiere un esfuerzo inicial, lo que hace que tenga una de las configuraciones iniciales más complejas de los *frameworks* actuales. En su página oficial, ofrece varios tutoriales, el *quick start* por ejemplo, muestra una aplicación sencilla pero no muestra los aspectos principales del núcleo de *Angular*.

Para estudiar sus principales características, vamos a dividirlas en tres apartados[3]:

- **Velocidad y rendimiento**
  - Generación de código: convierte tus plantillas en código altamente optimizado para las máquinas virtuales de JavaScript de hoy en día, ofreciéndote todas las ventajas del código escrito a mano con la productividad de un framework.
  - Universal: ejecuta la primera vista de tu aplicación en Node.js, .NET, PHP, y otros servidores para renderizado de forma casi instantánea obteniendo solo HTML y CSS
  - División del código: las aplicaciones de Angular se cargan rápidamente gracias al nuevo enrutador de componentes. Éste ofrece una división automática de códigos para que los usuarios sólo carguen el código necesario para procesar la vista que solicitan.
- **Productividad**
  - Plantillas: permite crear rápidamente vistas de interfaz de usuario con una sintaxis de plantilla simple y potente.
  - Angular CLI(*Command Line Interface*): las herramientas de línea de comandos permiten empezar a desarrollar rápidamente, añadir componentes y realizar test, así como previsualizar de forma instantánea la aplicación.
  - IDEs: la mayoría de editores e IDE tienen soporte para Angular
- **Historia completa del desarrollo**
  - Testing: utiliza Karma para realizar pruebas unitarias, y Protractor para realizar pruebas end-to-end de forma rápida y estable.
  - Animación: permite crear animaciones complejas y de alto rendimiento con muy poco código a través de la intuitiva API de Angular.

[3] https://angular.io/features

- Accesibilidad: posee características para crear aplicaciones accesibles con los componentes disponibles para ARIA.

React

También conocida como *React.js* o *ReactJS*, fue publicada en el año 2013. Al igual que *Angular*, el núcleo de la librería fue reescrito hace poco, siendo notables los primeros cambios en su versión 16.0, aunque esto se consideró más como una evolución del *framework* que como un cambio de dirección.

Entre los *frameworks* más utilizados del momento, es el más popular de todos. Es utilizado por sitios web como *Facebook* o *Netflix* y además ha surgido una gran comunidad de desarrolladores en torno a React. Tanto en la tabla 3.1 como en la figura 3.3 podemos ver la comparación de popularidad entre los 3 *frameworks* de JavaScript más utilizados actualmente.

Framework	Estrellas en GitHub	Descargas por npm	StackOverflow Tags
Angular 2+	31.000	1.93 Millones	87.000
React	83.000	5.85 Millones	67.000
Vue.js	76.000	0.78 Millones	12.000

Tabla 3.1: Comparación de popularidad de distintos *frameworks* de JavaScript ([Sat18])

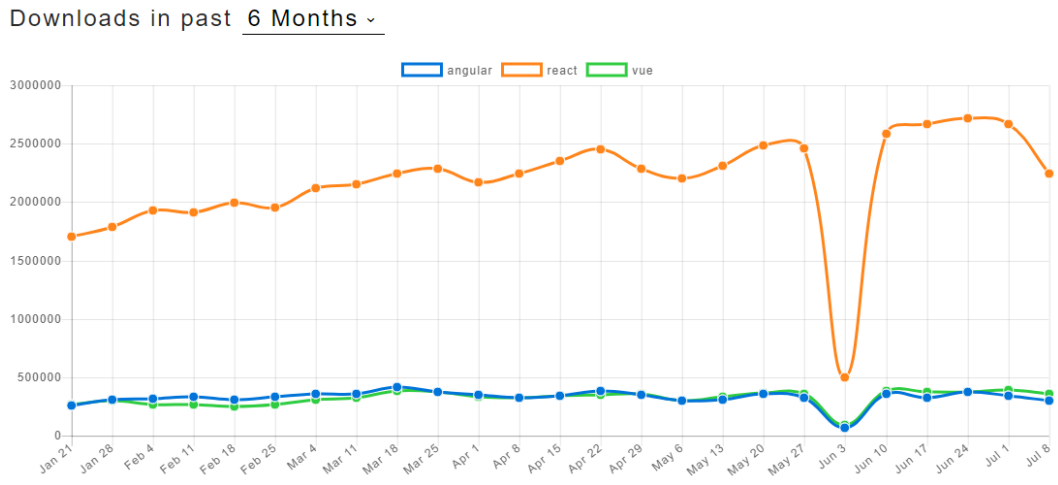

Figura 3.3: Gráfico de comparación de descargas [4]

Para la creación de un nuevo proyecto, el equipo de *React* ha creado una herramienta llamada `create-react-app`, una utilidad de *npm* que nos da una *build* ya configurada de un proyecto sobre el cual podemos empezar a trabajar.

Es bastante aconsejable tener conocimientos de ES6 si se pretende desarrollar una aplicación web con *React*, ya que utiliza muchos conceptos propios de ES6 como la interpolación de cadenas o las llamadas *arrow funtcions*.

En cuanto al funcionamiento de *React*, nos permite definir **componentes** como si de clases o funciones se trataran. Para definir nuevos componentes, deberemos heredarlos de la clase , como vemos en la línea 3 del listado 3.1. Todos los componentes en *React* deberán tener el método `render()`, que especifíca los elementos HTML de dicho componentes.

En la línea 5, devolvemos los elementos HTML y hacemos uso de `this.props.name`, que corresponde con el argumento `name` que deberemos proporcionar al componente `Welcome` al crearlo. Para poder utilizar el componente `Welcome` en otros componentes de *React*, deberemos exportarlo como se muestra en la línea 9.

```
1  var React = require('react');
2
3  class Welcome extends React.Component {
4      render(){
5          return <h1>Hello, {this.props.name}</h1>
6      ;}
7  }
8
9  export default Welcome;
```

Listado 3.1: Componente básico utilizando React

Vue.js

Es un *framework* de desarrollo progresivo de JavaScript, lo que significa que si utilizas un *framework* de aplicaciones web, puedes incluir *Vue.js* a solo una parte de la aplicación.

Es uno de los *frameworks* más recientes, apareció en Febrero de 2014 de la mano de Evan You, después de haber trabajado en AngularJS. Actualmente, *Vue.js* es matenido económicamente gracias a *Patreon*[5]. Está diseñado desde cero para ser adaptado gradualmente. La biblioteca principal se centra únicamente en la capa de la Vista y es fácil de integrar con otras bibliotecas o proyectos existentes.

Para comenzar a desarrollar con *Vue.js* simplemente necesitaremos una etiqueta de script en el HTML. En comparación a *React* y *Angular* es el framework que más fácil lo pone para empezar a desarrollar nuestra aplicación web. De los *frameworks* mencionados anteriormente, *Vue.js* es el menos perturbador en cuanto a la necesidad del uso de nuevos lenguajes o tecnologías. Funciona con ES5 aunque ES6 también está sorportado, además, se basa en HTML para la composición de la interfaz de usuario y CSS para el estilo.

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

[4] `http://www.npmtrends.com/angular-vs-react-vs-vue`
[5] `https://www.patreon.com/`

A diferencia de *React*, *Vue* puede trabajar directamente con plantillas en HTML, lo cual implica una seria de ventajas con respecto a los otros *frameworks*:

- Mayor facilidad de manejo para aquellos desarrolladores que estén acostumbrados a trabajar con HTML directamente.
- Las plantillas basadas en HTML hacen que sea mucho más fácil ir migrando progresivamente aplicaciones ya existentes.

3.3 Classroom Response System

Un CRS es cualquier sistema utilizado en un entorno *cara a cara* para poder realizar encuestas o tests a los alumnos de manera que se obtengan respuestas inmediatas de esas encuestas. Dentro de los CRS, podemos distinguir aquellos que hacen uso de las nuevas tecnologías y los que no, por ejemplo, un CRS no tecnológico puede ser cuando el profesor le pide a los alumnos que levanten la mano para estar a favor o en contra de alguna pregunta propuesta.

Los CRS tecnológicos utilizan transmisores, como *clickers* que permiten a los alumnos responder a las preguntas formuladas por el profesor, normalmente proyectadas, que han sido previamente preparadas.

Entre las principales ventajas de un CRS tecnológico frente a uno no tecnológico, es la posibilidad de que las respuestas sean anónimas, de manera que el alumno pierda esa vergüenza o presión al responder. Además, existe la posibilidad de generar gráficos sobre las respuestas de los alumnos para que puedan ser vistas por todos, o incluso almacenar los resultados para analizarlos posteriormente y preparar las sesiones magistrales en función de estos datos.

Las actividades que se pueden realizar con CRS pueden dividirse en tres etapas:

- **Presentaciones y preguntas**: en la sesión magistral, el profesor presenta una serie de conceptos y materiales, intercalando preguntas para obtener *feedback* por parte de lo estudiantes. Normalmente, en un CRS tecnológico se tiende a utilizar software para preparar la presentación (como puede ser Microsoft PowerPoint), de este modo, se pueden añadir preguntas en las diapositivas, en las que se da un corto periodo de tiempo para que pueda responden el alumno.
- **Respuestas del alumno**: en un CRS tecnológico, los alumnos disponen de pequeños transmisores para poder contestar a las preguntas planteadas por el profesor. Estos transmisores, envían la respuesta al ordenador del profesor, que son tratadas por un software específico para mostrar en tiempo real las respuestas de los alumnos y que estos vean las respuestas de todos y puedan así comparar más fácilmente. Esto facilita que se desarrollen debates en el aula.
- **Análisis de los datos**: una vez terminada la sesión, se pueden almacenar los datos de las respuestas, lo que permite al docente tener un mayor seguimiento de sus alumnos.

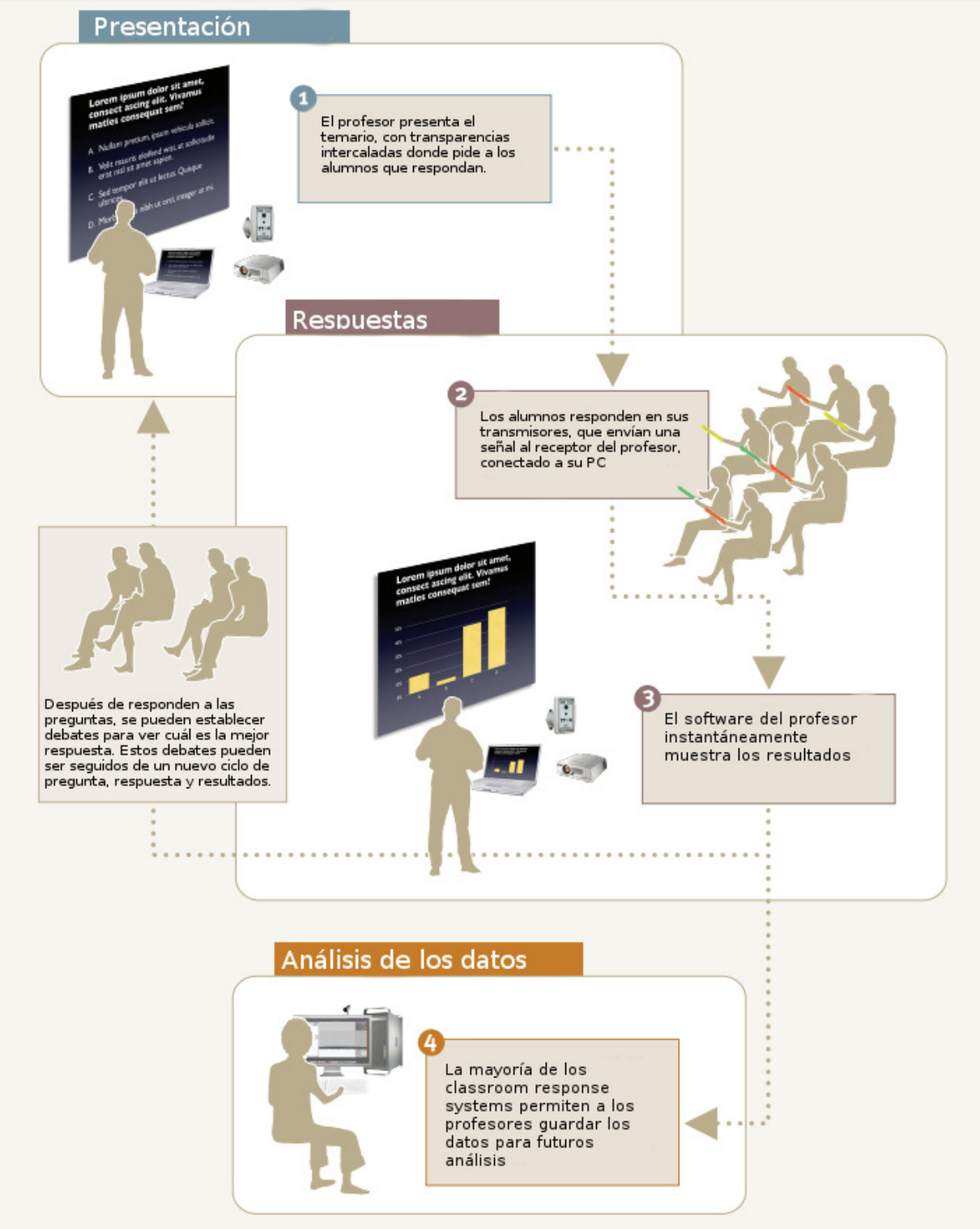

Figura 3.4: Diagrama de flujo de trabajo en un CRS [Bru09]

Existen además tres niveles de implementación de los CRS, cada cual hace un cambio mayor en cuanto a la forma de enseñanza y, según indica la literatura actual, en los resultados del aprendizaje:

- **Básico**: en este nivel de implementación, el profesor utiliza el CRS únicamente para monitorizar la clase, es decir, para conseguir la atención por parte de los alumnos

aunque no con una gran participación por parte de los mismos.

- **Medio**: el profesor utiliza el nivel medio para conseguir información en tiempo real. Dependiendo de las respuestas recibidas por parte de los alumnos, puede determinar en qué conceptos o temas profundizar más en la propia sesión.
- **Avanzado**: este nivel implica un cambio en la filosofía y estrategias de enseñanza del profesor. Para ello, las sesiones se basaran en ciclos de presentación de conceptos, preguntas, respuestas y pequeñas sesiones de debate.

Utilizando los CRS tecnológicos, sobretodo aquellos que hacen uso de los *clickers*, lo más común es realizar preguntas para escoger 1 respuesta entre varias, normalmente 4, pero existen más tipos de preguntas que pueden formularse dentro de un CRS tecnológico:

- **Preguntas de recuerdo**: se utilizan para recalcar conceptos claves de la sesión y comprobar si los estudiantes lo han comprendido, o recuerdan el temario dado días anteriores. No son muy propensas a generar debate.
- **Comprensión de conceptos**: van un paso más allá de las preguntas de recuerdo. Las respuestas ofrecidas a estas preguntas suelen estar basadas en los errores generales que comenten los alumnos o responder con sinónimos a lo explicado por el profesor.
- **Preguntas de aplicación**: destinadas a que los alumnos apliquen lo aprendido. Suelen proponer un escenario, a ser posible del "mundo real", donde los alumnos deben dar una solución.
- **Pensamiento crítico**: los alumnos deben analizar la relación existente entre varios conceptos aprendidos anteriormente o hacer una evaluación dependiendo de un criterio dado. Las respuestas dadas a este tipo de preguntas suelen ser todas correctas, aunque siempre habrá una que sea la mejor respuesta.
- **Perspectiva del estudiante**: son preguntas que fomentan la participación por parte del estudiante, ya que se les pide que den una opinión, dando lugar a debates entre toda la clase.
- **Monitorización**: estan relacionadas con el seguimiento por parte del profesor de los alumnos. Proporcionan *feedback* al profesor de los conceptos que han sido comprendidos, de los que no y cómo enfocar sus futuras clases.
- **Experimentos en clase**: los CRS tecnológicos pueden ser utilizados también para recoger datos de los experimentos realizados por los alumnos en clase.

Una vez vistos las posibles implementaciones que pueden tener esos sistemas en las aulas, así como los tipos de preguntas y flujos de trabajo utilizándolos, vamos a estudiar los beneficios y complicaciones o retos que suponen estos sistemas [KL09b].

Entre los **beneficios** o **ventajas** más relevantes encontramos:

- **Asistencia a clase**: para determinar si el uso de un CRS incrementa la asistencia a clase, se introdujeron en varios instituos y universidades para poder estudiarlos. En varios casos, los alumnos se veían forzados a ir a clase, porque los profesores puntuaban a través de estos sistemas, pero, por norma general, se incrementó la asistencia en un 15 %. [BL01]
- **Atención**: los estudiantes necesitan estar concentrados y prestando atención cuando se imparten las clases. La atención suele disminuir a los 20 minutos de la clase, si a esto le sumamos que algunas clases pueden llegar a durar 3 horas, es inevitable que algunos conceptos explicados no sean entendidos, o simplemente se ignoren u olviden. Una de las prácticas utilizadas con los CRS es la de plantear una pregunta cada aproximadamente 20 minutos de manera que los alumnos tengan que prestar atención en ese momento para responder.[BL01] [Cal07]
- **Anonimato y participación**: se les puede dar la posibilidad de a los alumnos de responder de manera anónima, lo que hace que no se sientan juzgados por sus profesores o compañeros, incrementando así la participación. [Cal07][SBD04]
- **Continuidad**: datos recogidos por parte de los estudiantes, nos dan a entender que el uso de estos sitemas incrementa el seguimiento de la asignatura. Las principales razones por las que existe esta continuidad es debido a que se sienten más inmersos en el proceso de aprendizaje, además de la "diversión"de usar los *clickers*. [Ber06]

En cuanto a los desafíos que ofrecen los CRS tecnológicos podemos distinguir tres:

- **Tecnológicos**: los principales problemas que presentan estos sistemas, en cuanto al hardware se refiere, es la pérdida de los mismos por parte de los alumnos, o que algún dispositivo, normalmente un *clicker*, esté defectuoso y la señal de la respuesta no llegue al receptor del profesor.
- **Profesor**: en teoría, una de las ventajas de estos sistemas en que los profesores reciben un *feedback* por parte de los alumnos para poder dirigir mejor sus sesiones. Sin embargo, en la práctica, puede que aquellos profesores más novatos se vean frustrados debido a que los alumnos no entienden los conceptos de la manera que el profesor los ha explicado. Además, según estudios, se cree que la cantidad de material impartida es menor por el hecho de utilizar estos sistemas. Podemos incluir también la dificultad que supone plantear preguntas que se adapten correctamente a un sistema CRS.
- **Alumnos**: puede que algunos alumnos se sientan frustrados al utilizar un CRS tecnológico debido a que supone un cambio en la enseñanza y no se adapten a tal cambio. Otro problema que puede surgir entre los alumnos son los debates que aparecen utilizando estos sitemas, ya que piensan que los distraen del tema principal de la clase.

Es por esto, que son varias las aplicaciones que han surgido para poder dinamizar las clases y mejorar la atención por parte de los alumnos, como pueden ser *Kahoot!*[6], *Quizizz*[7], y *Pear Deck*[8] entre otros. Puesto que están muy relacionados con lo que se pretende que sea el sistema final, vamos a estudiarlos más a fondo.

3.3.1 Kahoot!

Es una plataforma gratuita que permite la creación de cuestionarios de evaluación a través de su página web o con su App. El profesor crea concursos en el aula donde los alumnos son los participantes y pueden responder a través de sus *smartphones*.

Esta aplicación se engloba dentro del aprendizaje móvil electrónico (M-learning) y de la gamificación.

Para poder crear nuevos *kahoots* deberemos registrarnos en su página web como profesor, y dentro de esta, nos dará la opción de crear 4 tipos distintos. (ver figura 3.5)

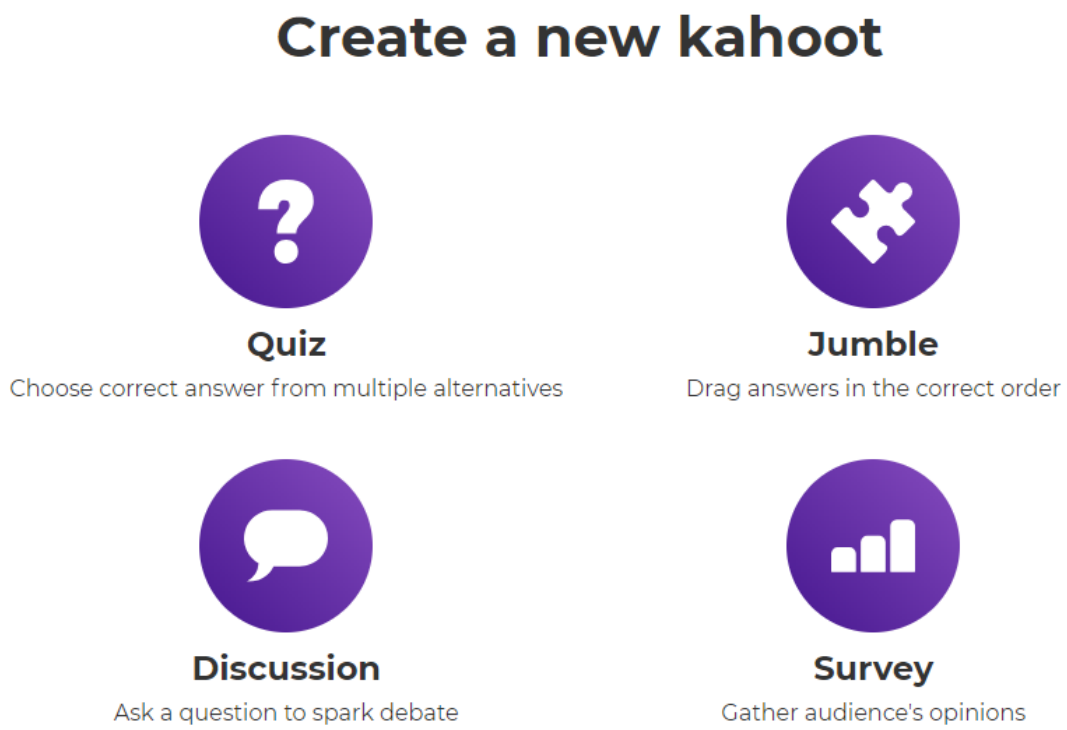

Figura 3.5: Tipos de *Kahoots* disponibles

[6] https://kahoot.it/
[7] https://quizizz.com/
[8] https://www.peardeck.com/

- **Quiz**: permite crear una sesión donde se les plantea a los alumnos diferentes preguntas con solo una respuesta correcta.
- **Jumble**: siguiendo el formato de puzzle, se deberá arrastrar las respuestas en el orden correcto.
- **Discussion**: utilizados para generar debate en el aula.
- **Survey**: recoger opiniones de los alumnos y poder mostrar gráficas y diferentes resultados.

Para la creación de *Kahoots* del tipo *Quiz*, el formulario que ofrecen es bastante amigable y sencillo. (ver Figura 3.6). Una vez establecido el nombre y las distintas opciones que nos ofrecen, podemos ir añadiendo las preguntas que queramos en un formulario muy similar. Al terminar, automáticamente desplegará nuestro Quiz en su sistema y podemos acceder a él utilizando un pin.

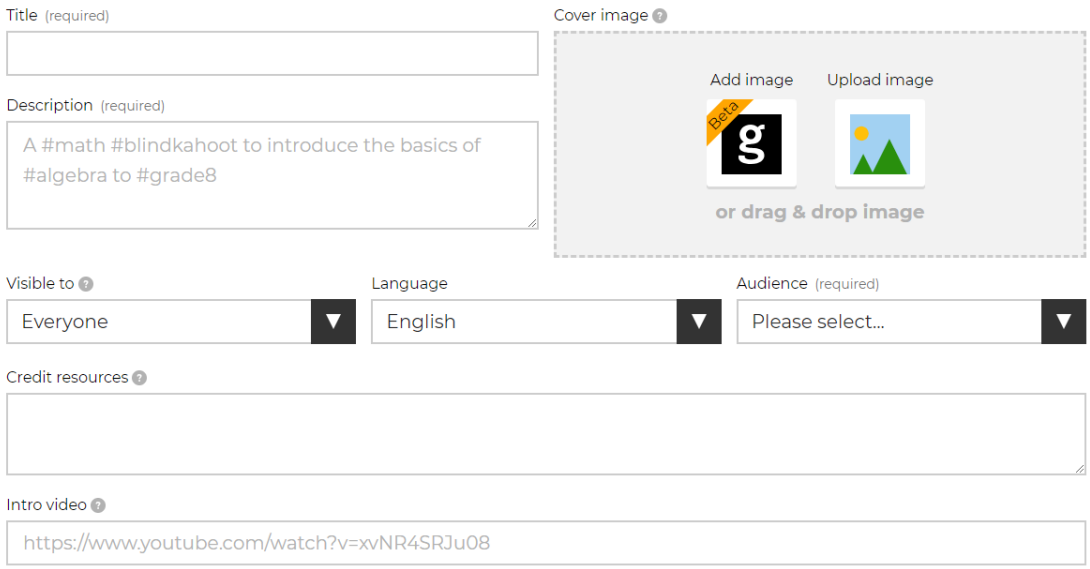

Figura 3.6: Formulario para la creación de *Kahoots* del tipo *Quiz*

La dinámica que sigue *Kahoot!* es sencilla, el profesor prepara la sesión, del tipo que más se adecúe a lo que él pretenda. A través de un proyector, el profesor muestra a los alumnos desde su portatil la pregunta y respuestas disponibles, mientras que en los *smartphones* de los alumnos, aparecen 4 símbolos, asociados a las 4 respuestas posibles y deben seleccionar uno de ellos (figura 3.7) .

El principal inconveniente de esta dinámica es que siempre se requerrirá de algun proyector para que la sesión pueda seguir por todos los alumnos, ya que la pregunta y posibles respuestas no aparecen en sus *smartphones*, aunque si lo que se quiere es que los alumnos estén atentos a la pantalla del profesor (lo cual no es recomendable ya que se podría utilizar para dar *feedback* al profesor de las respuestas de los alumnos), es mejor opción.

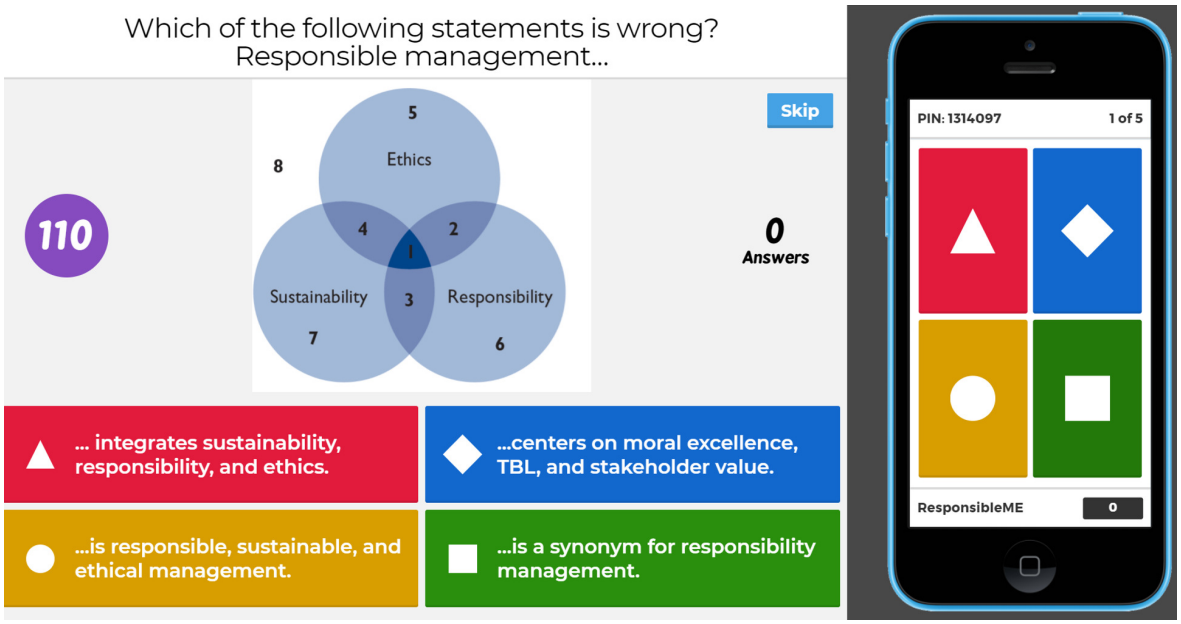

Figura 3.7: Interfaz de usuario de *Kahoot!*. A la izquierda la del profesor y a la derecha la del alumno.

3.3.2 Quizizz

Es una aplicación web que permite la creación de cuestionarios en linea. A diferencia de *Kahoot!*, *Quizizz* sólo ofrece la posibilidad de crear cuestionarios de un solo tipo, los cuales consisten en responder una respuesta entre varias.

Para crear un Quiz, nos mostrará primero un formulario donde escribiremos el título del *Quiz* y podremos seleccionar una foto para añadirla como fondo al mismo (véase figura 3.8). Después, deberemos ir añadiendo las preguntas de una en una (ver figura 3.9) o podremos ir seleccionando preguntas que ya estén almacenadas en la página de Quiz.

Aunque se podría decir que es una *copia* de *Kahoot!*, proporciona algunas funcionalidades muy interesantes:

- Los **enunciados** de las preguntas **aparecen en los *smartphones*** de los alumnos.
- Se pueden utilizar las **sesiones elaboradas como deberes** para realizarlos en clase.
- Permite compartir las actividades con Google Classroom.
- Como curiosidad, permite la adición de *memes* que se muestran al responder correcta o incorrectamente las preguntas.

En las figuras 3.8 y 3.9 se pueden ver los distintos formularios que *Quizziz* ofrece para la creación de *quizs* y las preguntas correspondientes.

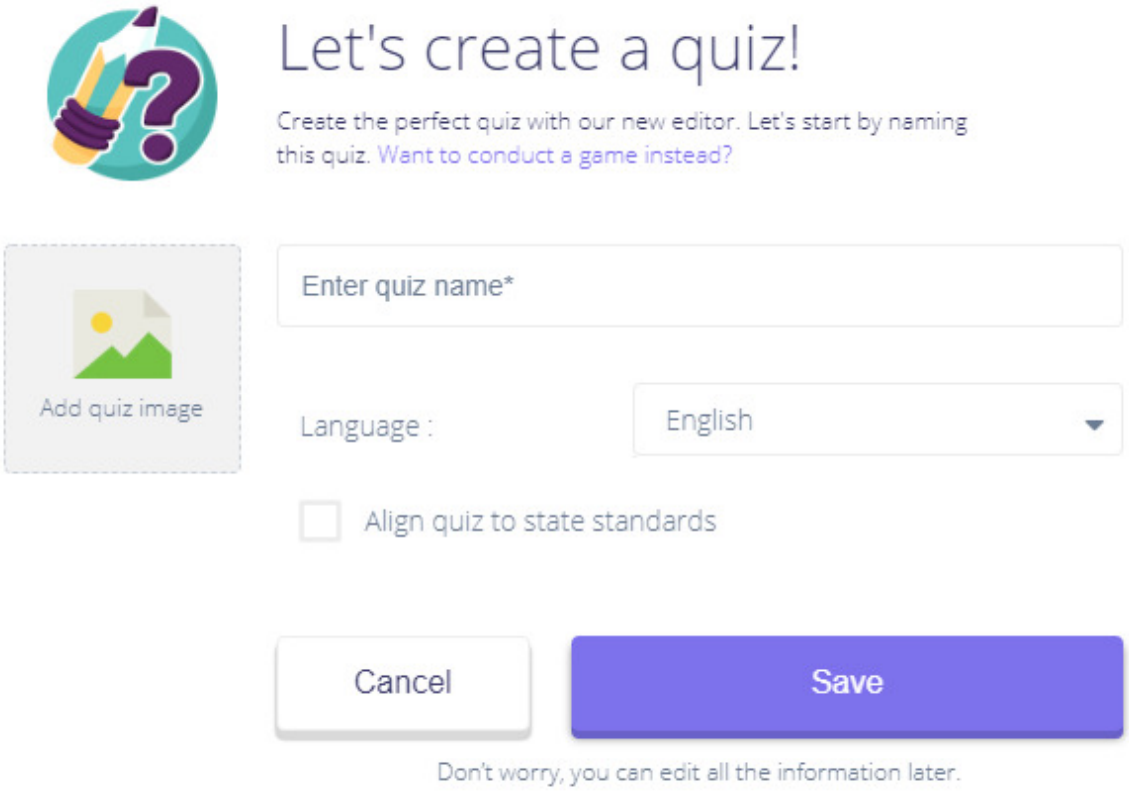

Figura 3.8: Formulario para la creación de *Quizs*

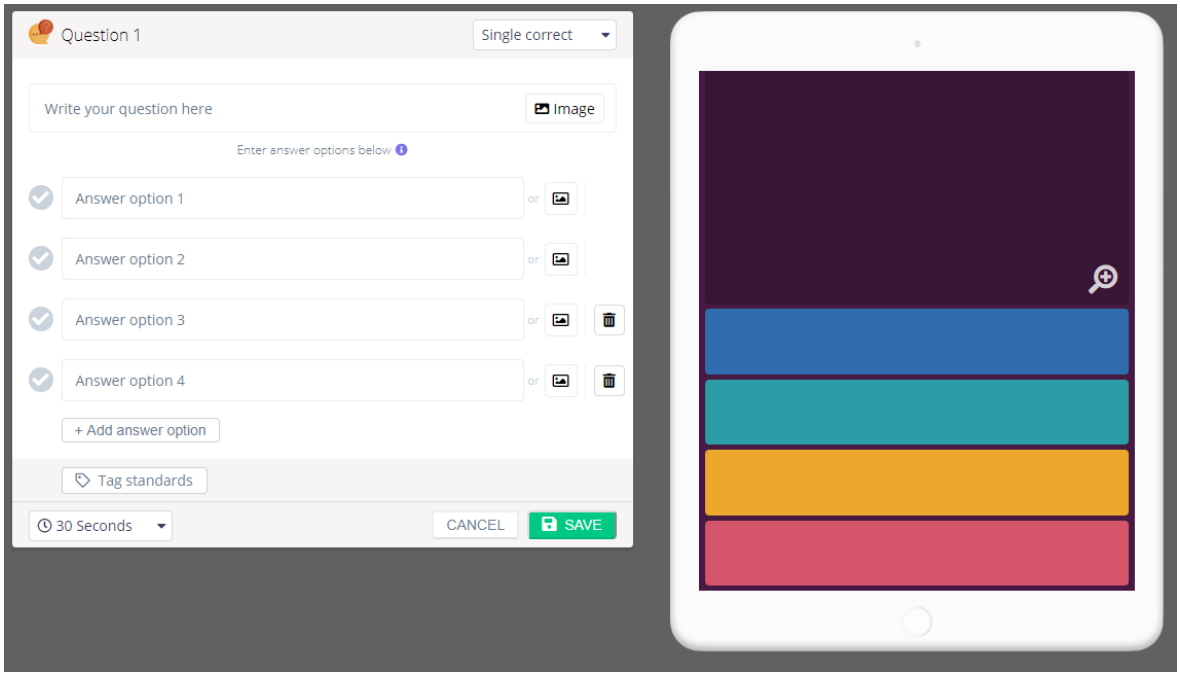

Figura 3.9: Formulario para la inclusión de preguntas en un *Quiz*

3.3.3 Pear Deck

Es una herramienta para realizar presentaciones interactivas, destinada a aumentar el aprendizaje y/o atención de los alumnos durante las clases. Idealmente, cada alumno dispondrá de su propio dispositivo donde podrá seguir la sesión.

Existen 5 tipos de preguntas en *Pear Deck*:

- **Dibujar**: Los estudiantes responden dibujando en un canvas o en una imagen de fondo, por ejemplo, dada la función $f(x) = x^2$, dibujar la función en los ejes x e y (ver figura 3.10)
- **Arrastrar**: Los alumnos responden arrastrando icónos o imágenes que el profesor haya elegido, se da la opción de pdoer subir hasta 5 iconos/imágenes. Un ejemplo de este tipo de pregunta sería colocar varios símbolos en un mapamundi.
- **Texto**: El alumno responde utilizando texto libremente.
- **Números**: Los estudiantes responden a ecuaciones y otro tipo de operaciones matemáticas utilizando números.
- **Varias respuestas**: Es la más común en estos tipos de herramientas o plataformas, el alumno escoge una respuesta entre varias que les propone el docente.

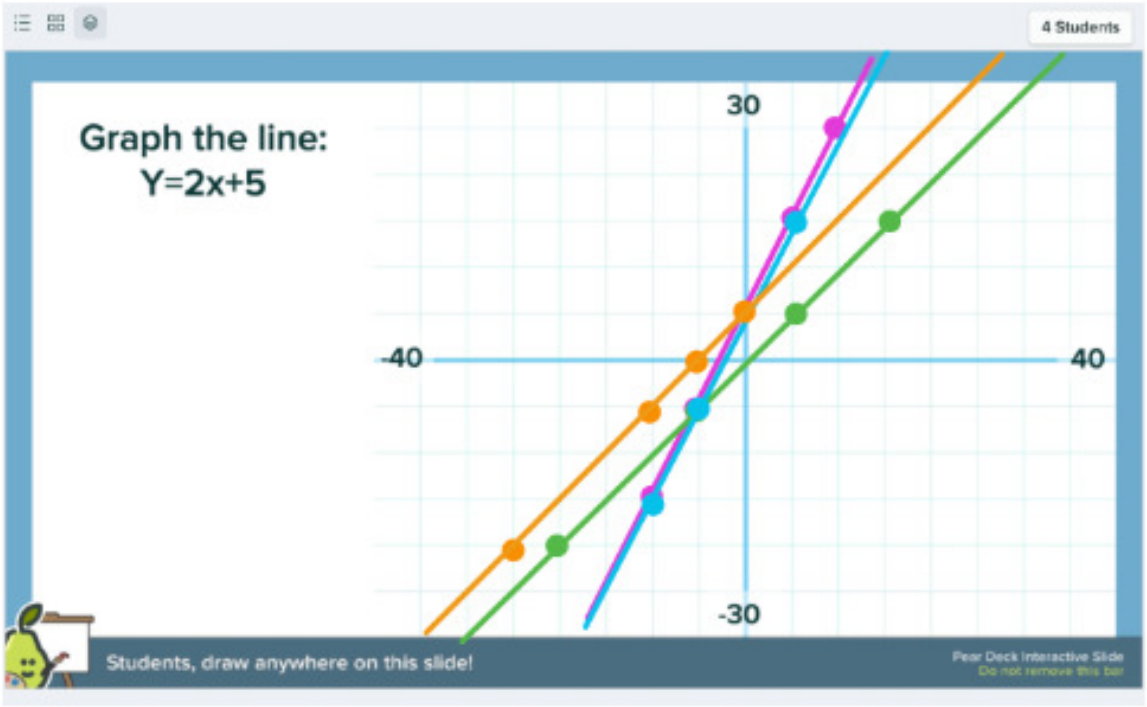

Figura 3.10: Respuesta a las sesiones del tipo dibujar

El profesor puede visualizar en cada momento la respuesta de cada alumno, y si lo desea, mostrar por un proyector las respuesta al resto de los alumnos.

En lo que respecta a la realización de las presentaciones por parte del profesor, necesitará añadir elementos interactivos para conseguir que los alumnos estén atentos a la presentación. Para esto, *Pear Deck*, incluye una herramienta de búsqueda utilizando Google para poder añadir imágenes y vídeos en las presentaciones.

Además, lo que diferencia *Pear Deck* de las demás paltaformas, es que es una aplicación de Google Drive, es decir, que se encuentra integrada con todo el ecosistema de Google, lo cual le da las siguientes ventajas:

- Utilizar las transparencias o presentaciones creadas en Google Slides dentro de *Pear Deck*
- Los estudiantes simplemente utilizarán su cuenta de Google sin tener que registrarse en sitios extra.
- Cada vez que creas una sesión o archivo de *Pear Deck*, este se queda automáticamente guardado en tu Google Drive.
- Se pueden organizar a los alumnos por clases, de manera que al crear una sesión, invites a la clase y no a los alumnos uno a uno.
- Al finalizar la sesión, se puede enviar a los alumnos de manera inmediata un fichero detallado con los resultados obtenidos en esa sesión. Dicho fichero puede ser editado desde Google Docs.

Con esto, finaliza el capítulo correspondiente al estado del arte del sistema. En el siguiente capítulo se discutirán las metodologías de trabajo que se han llevado a cabo para realizar este proyecto y las distintas herramientas utilizadas en el desarrollo del mismo.

Capítulo 4

Metodología

En este capitulo se profundizará en la metodología utilizada para el desarrollo de este proyecto, el por qué se escogio dicha metodología y cómo se ha llevado a cabo. Además, se discutirán los medios utilizados para realizar el proyecto, tanto medios hardware como software.

4.1 Metodología de trabajo

La metodología seleccionada para este proyecto ha sido el desarrollo iterativo e incremental, siguiendo el Proceso Unificado de Desarrollo (PUD)[JBR00]. El PUD es un marco de desarrollo software que se caracteriza principalmente por estar:

- **Dirigido por casos de uso**: utilizados para capturar los requisitos funcionales y para definir los contenidos de la iteraciones. Cada iteración toma un conjunto de casos de uso y desarrolla todo el camino a través de las distintas disciplinas: diseño, implementación, pruebas...
- **Centrado en la Arquitectura**: Se ha desarrollado una arquitectura Modelo Vista Controlador (MVC) con ciertas modificaciones para poder adaptarla al proyecto, la cual se detallará más en profundidad en el capítulo 5.
- **Desarrollo e iterativo incremental**: el proyecto se ha dividido en varias fases o iteraciones. Cada iteración cubre una serie de casos de uso que se han desarrollado conforme a la arquitectura propuesta. Al terminar cada iteración, los resultados son expuestos al director del proyecto de manera que la involucración es mayor y se define mejor el resultado final.

4.1.1 Iteraciones

A continuación, se describen las distintas iteraciones que se han realizado a lo largo del desarrollo del proyecto GEA:

4.1.2 Iteración 1

Al iniciar el proyecto, unas de las primeras que se realizó fue la **recogida de requisitos** que se tenían que cumplir. Para detallar estos requisitos, se tuvieron varías reuniones con el

director del proyecto. A continuación, se enumeran los requisitos extraídos:

1. La plataforma web permitirá la creación de sesiones magistrales por parte del profesor y garantizará la correcta unión de los alumnos a las mismas.
2. La diseño de la web debe adaptarse al tamaño de pantalla de distintos dispositivos donde vaya a ser utilizado.
3. La plataforma web proporcionará herramientas de autenticación con Google.
4. El sistema dispondrá de herramientas para proporcionar *feedback* en tiempo real sobre los resultados de los alumnos.
5. El sistema web dará la opción de generar un informe con los resultados finales obtenidos por los alumnos.

4.1.3 Iteración 2

Una vez conocidos los requisitos y objetivos del proyecto, se llevó a cabo una **investigación de sistemas similares** para definir el diseño web del mismo. En esta iteración se estudiaron los sistemas web basados en CRS que no requieren de hardware extra para su funcionamiento, como los mencionados en la sección 3 y se fueron extrayendo ideas de aquellos sistemas estudiados, que se piensan encajarían bien en el proyecto que se pretende realizar.

4.1.4 Iteración 3

Después de haber hecho los primeros bocetos del diseño y haber fijado algunos ya como finales, se comenzaron a estudiar los actuales ***frameworks* de desarrollo web**, tanto para el ***front-end*** como el ***back-end*** para poder desarrollar el proyecto. Una vez recogida la suficiente información, se escogió *React* para la implementación del *front-end* y *Node.js* para el desarrollo del *back-end*.

Tras la elección de la tecnología, se creó una interfaz simple y un servidor, y se hicieron las pruebas pertinentes para poder comunicar *React* con *Node.js* a través de un *proxy*. Además de esto, para poder desplegar la plataforma, y utilizar otros servicios, se integró la solución en GAE.

4.1.5 Iteración 4

En esta fase, se comenzaron a implementar los diseños de la interfaz que se habían establecido anteriormente y sus respectivas funcionalidades. También, se estudiaron métodos de autenticación para ir almacenando los distintos usuarios del sistema y poder asociar así las distintas sesiones que vayan creando. Se escogió el correo de gmail como modo de autenticación debido a su popularidad y a componentes de *React* que facilitan el proceso.

4.1.6 Iteración 5

En anteriores fases, se integraró la solución en GAE, lo que ha facilitado el uso de varios servicios que ofrece, como **Google Coud Datastore** y **Datastorage**. Se **definieron los modelos** que se iban a utilizar para almacenar los datos de usuarios, sesiones y preguntas. Tras la inclusión de estos en Datastore, se implementaron las funciones necesarias para acceder, eliminar y editar los registros de la misma en el *back-end* del proyecto.

4.1.7 Iteración 6

Una vez definidos los modelos que van a ser utilizados para almacenar las distintas sesiones en la plataforma web, se debe dar la posibilidad a los usuarios de **crear** dichas **sesiones**, además de poder acceder a las mismas, usando un pin en caso de ser alumno, o desde un enlace en el caso de ser un profesor con el rol de administrador. Se hicieron sesiones de prueba con dos preguntas, dando la opción al profesor de escoger la pregunta que fueran a ver los alumnos y de reiniciar las sesiones mediante el uso de sockets.

4.1.8 Iteración 7

Al finalizar las sesiones, se requiere de un informe que refleje los resultados obtenidos por cada alumno, en esta fase se llevó a cabo la implementación de dicha funcionalidad, además de algunas funcionalidades extra como la posibilidad de incluir un **temporizador** en cada pregunta de la sesión de manera que al llegar al 0, el alumno no pudiera responder a la pregunta actual.

4.1.9 Iteración 8

Como última tarea a realizar, se realizaron las pruebas necesarias en el sistema para comprobar que todo funcionaba correctamente, se generó una versión final para su despliegue en GAE.

4.2 Características hardware y software del desarollo

En esta sección, se explican los elementos software y hardware que han sido requeridos a lo largo de todo el desarrollo del proyecto.

4.2.1 Medios hardware

Los componentes hardware que han sido requeridos para realizar este proyecto son los siguientes:

- **Equipo de trabajo** : ordenador de sobremesa donde se llevará a cabo el desarrollo del sistema con las siguientes características:
  - Procesador: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
  - Memoria RAM: 16.00 GB

- Disco Duro: 1 TB & 128 GB SSD
- GPU: GTX 1060 6GB
- Sistema Operativo: Windows 10 & Ubuntu 16.04

- **Motorola Moto G5**: en él se llevarán a cabo las pruebas de conexión a la herramienta web desde móvil y cuenta con las siguientes características:
  - Procesador: Snapdragon 430 octa-core a 1,4 GHz
  - GPU: Adreno 505 a 450 MHz
  - Memoria RAM: 2.00 GB
  - Memoria Interna: 8.00 GB
  - Sistema Operativo: Android 7.0

4.2.2 Medios software

A continuación se enumeran las diversas herramientas software empleadas y divididas en categorias:

Sistemas Operativos

- **Ubuntu 16.04 LTS**: es el sistema operativo que se ha utilizado principalmente para el desarrollo del proyecto. Se escogió este sistema por ser de software libre y por las facilidades que ofrece para el desarrollo web con respecto a otros sistemas operativos.
- **Android 7.0**: es el sistema operativo que posee el móvil donde se han realizado la mayoría de las pruebas para comprobar el correcto *repsonsive design*.

Servicios en la nube

- **Google App Engine**: utilizado para el alojamiento del proyecto. Destacar del mismo que se han utilizado distintos servicios integrados del mismo como **Datastore**[1], que nos permite mantener una base de datos NoSQL, y **Storage**[2] utilizado para almacenar distintas imágenes utilizadas a lo largo del proyecto y dar a los usuarios la opción de subir las suyas propias. Todas estas funcionalidades son accesibles a través del SDK de Google Cloud que ha sido utilizado en su versión 194.0.0.
- **Google Compute Engine**: ha sido utilizado para realizar un servidor dedicado únicamente al uso de *websockets* y permitir así la comunicación entre los dispositivos del profesor y el alumno.

[1] `https://cloud.google.com/datastore/docs/concepts/overview`
[2] `https://cloud.google.com/storage/`

Herramientas de desarrollo

- **Atom**[3]: ha sido el editor de texto seleccionado para elaborar todo el código del proyecto en su versión 1.22.1. A parte de los lenguajes de programación ya soportados, incluye temas para JavaScript o incluso para *frameworks* específicos como *React* o *Node.js*.
- **Make**[4]: para facilitar la inicialización del proyecto y el despliegue en Google App Engine se han utilizado *Makefiles* en su versión 4.1.
- **Npm**[5]: ha sido el manejador de paquetes para *Node.js* utilizado en este proyecto en su versión 5.6.0. A parte de facilitar la descarga de nuevos componentes tanto para *Node.js* como para *React*, permite la ejecución de pequeños *scripts* y de la inicialización del sistema web en local.
- **Nodemon**[6]: se ha utilizado para agilizar el proceso de desarrollo, ya que reinicia el servidor cada vez que detecta cambios en el código del mismo, por lo que no nos obliga a cerrarlo y abrirlo cada vez que se incluyen nuevas funcionalidades o se corrigen fallos. Ha sido utilizado en su versión 1.17.2.
- **Mercurial**[7]: se ha empleado este sistema de control de versiones para mantener un repositorio del proyecto. El servicio de repositorios ha sido proporcionado por *Bitbucket*.

Lenguajes

- **React**: *framework* de desarollo basado en JavaScript que ha sido utilizado para implementar el *front-end* de este proyecto. Como se explica en el capítulo 3, es el *framework* más popular actualmente, con una gran comunidad detrás y que ofrece muchas facilidades y componentes ya realizados para el desarrollo del *front-end*. La versión utilizada de *React* ha sido la 16.4.1.
- **Node.js**: *framework* de desarollo basado en JavaScript que ha sido utilizado para implementar el *back-end* del proyecto. Versión utilizada: 4.6.2.
- **HTML**: empleado junto con *React* para el desarrollo del *front-end*. En su versión HTML5, nos permite incluir varios elementos que facilitan la interacción del sistema con el usuario.
- **CSS**: utilizado para el estilo de HTML y adaptar los distintos elementos que componen el interfaz web a varios dispositivos donde es posible acceder al proyecto realizado.

[3] `https://atom.io/`
[4] `https://www.gnu.org/software/make/`
[5] `https://www.npmjs.com/`
[6] `https://nodemon.io/`
[7] `https://www.mercurial-scm.org/`

Bibliotecas

- **socket.io/socket.io client**[8]: librería de JavaScript que permite, junto con Google Compute Engine, la comunicación bidireccional entre los roles del profesor y del alumno. Versiones utilizadas: 2.1.1 para ambas.
- **react-google-login**[9]: componente de react que permite la autenticación de usuarios utilizando su cuenta de correo de *gmail*. Versión utilizada: 3.2.1.
- **react-csv**[10]: componente de react que ha sido utilizado para habilitar la descarga de ficheros .csv al finalizar las sesiones. Versión utilizada: 1.0.19.

Documentación

- **Sharelatex**[11]: editor en linea de LaTeX utilizado para la elaboración de este documento. Permite una rápida compilación del proyecto y la sencilla importación de plantillas.
- **Visual Paradigm Online**[12]: editor en linea de diagramas UML. Proporciona herramientas para realizar los diagramas de mánera rápida y sencilla.
- **Inkscape**: Programa utilizado para la creación de gráficos vectoriales. Se ha utilizado para generar varias imágenes de este documento.

[8] https://socket.io/
[9] https://www.npmjs.com/package/react-google-login
[10] https://www.npmjs.com/package/react-csv
[11] https://es.sharelatex.com/
[12] https://online.visual-paradigm.com/es/

Capítulo 5

Arquitectura

Este capítulo está dedicado a la prsentación de la arquitectura del proyecto GEA. Para ello, se diferenciarán los distintos componentes del mismo, dividiéndolo según la arquitectura **cliente - servidor** y dentro de la parte del cliente, atendiendo a los dos roles de los que dispone el sistema, **profesor** y **alumno**. Para cada componente desarrollado, se indicará el problema que resuelve, las decisiones que fueron llevados a cabo para solucionarlos y cuando sea necesario, se incidirá en cuestiones de diseño.

Finalmente, también se hablará sobre los recursos que han sido necesarios para el desarrollo del sistema, indicando además el coste de todos ellos.

5.1 Visión general de la arquitectura

En el capítulo 1 se presentó un esquema resumido de cómo sería el funcionamiento general de GEA. Para introducir la arquitectura y desarrollar los componentes posteriormente, se va a seguir un enfoque *top-down*, en el que se presentan los componentes del sistema con un alto nivel de abstracción, para poder ofrecer una imagen general del sistema.

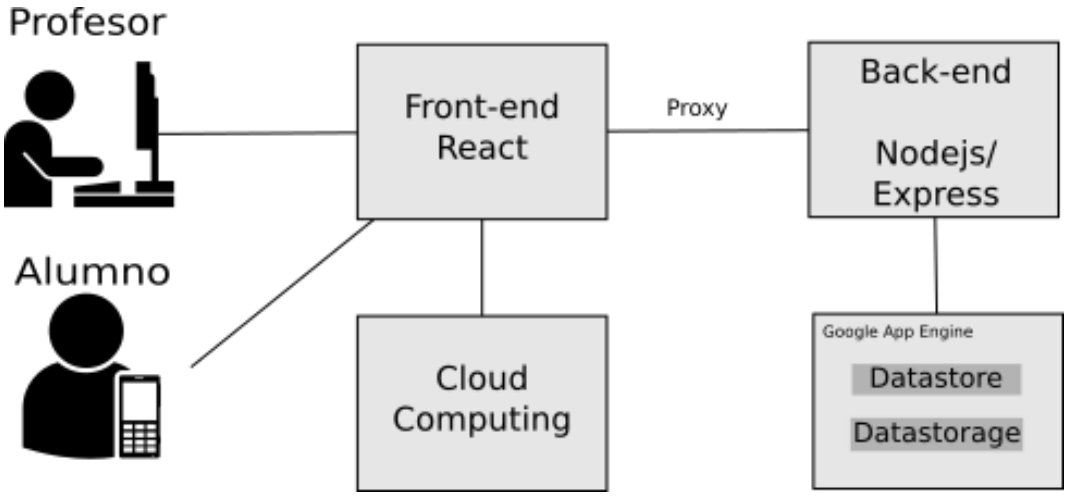

Figura 5.1: Diagrama de componentes técnicos de GEA

Como podemos ver en la figura 5.1 el sistema no está dirigido a un solo usuario, si no, que tendremos dos roles dentro del sistema. Por un lado, tendremos el rol de profesor, encargados de crear las presentaciones en sus sesiones magistrales y de controlar las mismas, siendo

ellos capaces de ver en todo momento la pregunta actual, resultados, etc. Por otra parte encontramos el rol del alumno, a los que se les presentarán estas sesiones en sus *smartphones* y serán los encargados de responder a las preguntas planteadas por el profesor.

Front-end/React

En capítulos anteriores se ha discutido qué tecnologías usar para implementar el sistema. En este caso, el *front-end* será implementado con *React*, que nos permite centrarnos en la interfaz del sistema, lo que será visto por el profesor y alumnos. A los alumnos se les presentará la opción de unirse a las sesiones previamente indicadas por los profesores y a los profesores se les darán los mecanimos necesarios para poder crear nuevas sesiones y acceder a las que quieran realizar en sus sesiones magistrales, permitiendo realizar funciones de administrador de la misma como avanzar a la siguiente pregunta, controlar las puntuaciones en tiempo real de manera ordenada y acceder a la vista actual de los alumnos.

Back-end/Nodejs-Express

Esta parte del sistema será la encargada de atender las peticiones por parte del *front-end*. Proporcionará la información necesaria cuando los alumnos deseen acceder a una sesión y a los profesores les informará sobre sus sesiones ya creadas y enviará los datos de las sesiones a las que quieran acceder, así como los mecanimos necesarios para poder guardar nuevas sesiones en el sistema. Se ha decidido implementarla con *Express*, una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

Google App Engine

Es necesario mantener de manera ordenada y consistente toda la información que se maneja en el sistema, es por eso que se requiere del uso de GAE para dicha tarea. No solo será el encargado de que se pueda acceder al sistema desde un navegador web, además, nos proporciona dos servicios para el manejo de las bases de datos o **modelos** requeridas por el sistema (***Datastore***) y el almacenamiento de ficheros necesarios para las sesiones, en este caso imágenes (***Datastorage***).

Google Compute Engine

Se requiere de mecanismos que nos permitan la comunicación en tiempo real de los profesores con los alumnos para poder enviar las respuestas de los mismos. Debido a ciertas restricciones impuestas por GAE por seguridad, que se discutirán más adelante, ha sido necesario la implementación de un servidor de *websockets* en Google Compute Engine. A parte de enviar las respuestas de los alumnos, envía la información necesaria para saber cuándo el cliente de los alumnos debe cambiar de pregunta y dar por finalizada la sesión y así poder dar un control total al profesor.

5.2 Patrones de diseño

Para la implementación de GEA se ha utilizado el patrón de diseño **Modelo -Vista - Controlador** (MVC), muy utilizado en este tipo de aplicaciones o sistemas. Está caracterizado por la separación de la lógica de la aplicación de la lógica de la presentación y es utilizado en la mayoría de los *frameworks* modernos.

- **Modelo:** contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. Responsable además de acceder a la capa de almacenamiento de datos.
- **Vista:** o interfaz de usuario, compone la información que se envía al cliente y los mecanismos de interacción con el mismo. Recibe los datos proporcionados por el modelo y puede a su vez actualizar la interfaz actual.
- **Controlador:** actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

En el capítulo 3 se hizo un estudio de los *frameworks* actuales más utilizados, y se escogió *React* para el desarrollo de este proyecto debido a su crecimiento en los últimos años, la comunidad que había detrás y la gran cantidad de paquetes desarrollados que podemos incluir en nuestros proyectos[1]. Sin embargo, lo normal cuando se desarrolla una aplicación con *React* no es seguir el patrón MVC, ya que está orientado únicamente en la interfaz o vista, a diferencia de otros *frameworks* modernos como podría ser Angular que también incluye la lógica.

Es por esto que se ha adaptado el proyecto para seguir el patrón MVC. React al no implementar la lógica y centrarse sólo en la vista, nos da la oportunidad de utilizar otros *frameworks*. En este caso se ha escogido *Node.js* y *Express*.

Para poder establecer la comunicación entre la parte del *front-end* y *back-end*, entre la vista y el modelo, es necesario implementar el controlador. *React*, al estar enfocado únicamente en la Vista, necesita obligatoriamente dicho controlador para poder hacer llamadas al *back-end* y recibir la información necesaria. La clave en este proyecto para poder establecer dicha comunicación recae en un *proxy*, el cual nos indica dónde se está ejecutando el servidor *Express*.

El correcto funcionamiento del *proxy* se consigue añadiendo la siguiente línea en el *package.json* en la parte del cliente del proyecto:

```
"proxy": "http://localhost:8080/"
```

[1] https://www.npmjs.com/

Posteriormente, en nuestros componentes creados con React, deberemos realizar las peticiones al *back-end* ya sea para enviar datos, recibirlos o ambas. En el listado 5.1, se muestra el código correspondiente al perfil del usuario, donde pide al *back-end* los *quizs* (línea 2) que tiene asociados utilizando su e-mail(línea 9). En la línea 13 se recibe la respuesta y se transforma a JSON para poder manejar los datos de manera más sencilla.

```
1  callApi = async () => {
2      const response = await fetch('/listQuizs', {
3        method:'post',
4        headers: {
5           'Accept': 'application/json',
6           'Content-Type': 'application/json',
7         },
8        body: JSON.stringify({
9          email: this.props.location.state.profile.email
10       })
11     });

13     const body = await response.json();

15     if (response.status !== 200) throw Error(body.message);
16     return body;
17 }
```

Listado 5.1: Funcionamiento del proxy

Con esto conseguimos adaptar el patrón MVC al proyecto, dejando a *React* encargándose únicamente de la Vista y delegando la lógica a *Express*.

5.3 Autenticación de usuarios

Para mantener la información de las sesiones creadas por los disitintos profesores, es necesario que se autentiquen en el sistema. En vez de crear un mecanismo propio para la autenticación, se ha optado por dar la posibilidad al usuario de acceder al sistema mediante una cuenta de correo en Google. Debido a que el correo de Google, *gmail*, es de los más utilizados, se le da al usuario una manera rápida y sencilla de autenticarse en el sistema sin la necesidad de crear nuevos credenciales.

5.3.1 OAuth2

OAuth2 es un protocolo de autorización con el que el usuario (en nuestro caso los profesores) permite a terceros (GEA) el acceso a determinados contenidos protegidos bajo credenciales a un servidor confiable (Google), evitando así que el usuario deba proporcionar sus credenciales a terceros para acceder a las funcionalidades y/o servicios del sistema creado.

Lo primero que debemos hacer para poder hacer uso del OAuth2 es registrar el dominio de GEA en los proveedores de recursos a los que se desea acceder. Con esto conseguimos darnos de alta como cliente en el proveedor obteniendo un identificador y clave secreta para poder identificar GEA como un cliente válido. Una vez registrado el dominio y configurado en nuestro sistema el *token* o identificador y la clave secreta, ya podremos hacer uso de OAuth2, disponiendo de la información necesaria que nos proporciona el proveedor de credenciales seguro.

Si bien es cierto que OAuth2 nos facilita la autenticación de los usuarios, se debe tratar la información que este nos proporciona de manera correcta, para saber qué usuarios se han registrado ya en el sistema y poder así devolver la información necesaria o en caso contrario crear un usuario nuevo y añadirlo al sistema.

En la figura 5.2 se muestra el diagrama de secuencia correspondiente al inicio de sesión por parte de los usuarios:

1. Obtención de credenciales de usuario a través del proveedor
    1 El sistema envía el *client_id* y *secret_key* que se le han sido proporcionados al registrar el dominio en el proveedor.
    2 Una vez comprobados que el *client_id* y la *secret_key* son correctas, se le piden los credenciales correspondientes al usuario en el proveedor utilizado.
    3 El usuario introduce sus credenciales del proveedor y los envía para ser validados.
    4 Una vez comprobados los credenciales del usuario, el proveedor envía el token de la sesión, conteniendo los datos correspondientes al usuario que serán utilizados por el sistema.
2. Inicio de sesión en el sistema
    1 Con el token de la sesión ya obtenida, el usuario hace una petición para iniciar sesión en el sistema.
    2 El sistema al recibir el token, pide al proveedor que valide los mismos.
    3 En caso de que el token sea correcto, el proveedor lo valida y envía la confirmación al sistema.
    4 Confirmados los datos del usuario, el sistema realiza las operaciones pertinentes para iniciar la sesión del usuario.
    5 El sistema confirma al usuario que su sesión ha sido iniciada.

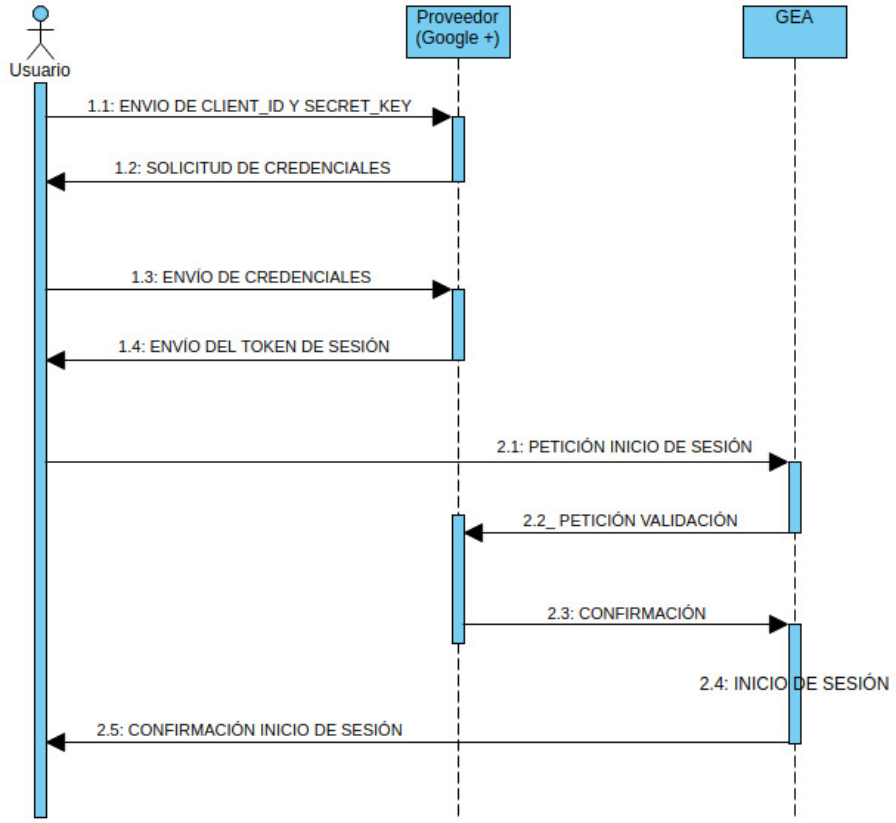

Figura 5.2: Diagrama de secuencia de inicio de sesión utilizando OAuth2

react-google-login

Este componente[2] de React nos facilita mucho la implementación del *login* de usuarios. Basado en OAuth2 nos proporciona los datos de los usuarios devueltos por el proveedor una vez introducidos los credenciales de manera muy sencilla. En el listado 5.2 vemos un ejemplo de implementación del componente, en el que simplemente para que funcione debemos introducir el *client_id* de nuestro sistema una vez registrado el dominio en el proveedor.

```
<GoogleLogin
    clientId="658977310896-knrl3gka66fldh83dao2rhgbblmd4un9.apps.googleusercontent.com"
    buttonText="Login"
    onSuccess={successGoogle}
    onFailure={failureGoogle}
  />
```

Listado 5.2: Componente ejemplo react-google-login

[2] https://www.npmjs.com/package/react-google-login

Nos proporciona además dos llamadas a métodos (`OnFailure()` y `OnSuccess()`) en función de si el *login* ha sido completado o no. Una vez añadido el compenente al sistema, simplemente debemos sobreescribir los métodos proporcionados para gestionar los datos de la manera que a nosotros nos interese.

5.4 Creación de usuarios

Debido al uso de OAuth2, no es necesario que los usuarios creen nuevos credenciales para acceder al sistema, pero de cara a la persistencia del sistema, es necesario crear un **modelo** nuevo para mantener la información relativa a los usuarios. Es por ello que en *Google Datastore* se definirá una nueva entidad ***User*** con las siguientes propiedades:

- **ID**: Clave generada automáticamente por *Datastore* que identifica de manera única a cada usuario.
- **email**: Corresponde a la cuenta de correo electrónico de *gmail* que el usuario ha utilizado para autenticarse en el proveedor y a su vez en el sistema. Será utilizado como clave para poder acceder a la información del usuario, debido a que no se puede conocer de antemano el ID generado por *Datastore* de cada usuario.
- **lastName**: Apellido del usuario.
- **name**: Nombre del usuario. Tanto el nombre como el apellido se utilizarán para mostrar los datos de usuario al inicio de su sesión.
- **quizs**: propiedad de tipo *matriz* en *datastore* donde se guardan los identificadores o *keys* de los *quizs* que ha creado el usuario de manera que se le muestre al usuario aquellos que ha creado hasta la fecha.

Es importante destacar la facilidad que se le proporciona a los usuarios para poder acceder al sistema. En caso de que sea su primer inicio, quedará registrado de manera automática sin necesidad de que el usuario deba aportar nuevos credenciales, simplemente con los que posee en el proveedor es suficiente.

De cara a la **implementación**, se debe comprobar si el usuario está ya registrado en *datastore*. Esto se consigue realizando una consulta a la tabla de entidades de ***User***, devolviéndonos la lista de jugadores con el *e-mail* asociado al proveedor. Si la lista está vacía, significa que es la primera vez que ese usuario accede y se le añade a la tabla de usuarios. En el listado 5.3 se ve reflejada dicha funcionalidad implementada en la parte del *back-end* del sistema, en el archivo *server.js*.

```
app.post('/users', (req, res) => {
    user.getUser(req.body.email, (err, results) => {
    if (results.length === 0) {
        usersDataStore.addUser(req.body.name, req.body.lastName, req.body.email, (key));
    }
  });
});
```

Listado 5.3: Comprobación de usuario existente

En cuanto a la inclusión del usuario en caso de no existir, el listado 5.4 muestra cómo añadirlo a *datastore*, creando anteriormente la entidad con los datos correspondientes al usuario y devolviendo la clave proporcionada por *datastore*.

```
function addUser(name, lastName, email, callback){
    var entity = {
      key: datastore.key(model),
      data: {
        name: name,
        lastName: lastName,
        email: email
      }
    };
    datastore.save(entity, callback(datastore.KEY));
}
```

Listado 5.4: Adición de usuarios en *datastore*

5.5 Creación de sesiones

Una de las funcionalidades más importantes de GEA es la de crear sesiones por parte de los profesores, con la opción de que los alumnos puedan realizarlas posteriormente con supervisión del profesor. La creación de dichas sesiones solo está disponible para los usuarios que se hayan registrado como profesor, por lo que cuando estos inicien sesión, se les deberá mostrar claramente que tienen la opción de crear nuevas sesiones, además de ver las que ya se hayan creado.

Las sesiones creadas por los profesores están compuestas por un título y una serie de preguntas, tipo test, donde se les da la opción a los alumnos de elegir una respuesta entre cuatro de las cuales solo una es correcta. Además, al profesor se le da la oportunidad de personalizar sus presentaciones, añadiendo una imagen de fondo para la sesión y pudiendo añadir imágenes en las preguntas, por lo que son varios aspectos a tener en cuenta cuando se cree la sesión.

Teniendo en cuenta lo anteriormente dicho, se deben diseñar unos formularios adecuados

que recojan toda la información necesaria para poder crear las sesiones correctamente. En este caso se han considerado dos tipos de formulario, el primero para indicar el título de la sesión y dar la oportunidad de incluir una imagen de fondo y el segundo para ir añadiendo las preguntas de dicha sesión.

5.5.1 Título de las sesiones

La información que recoge el formulario en esta parte de la creación de la sesión es la siguiente:

- **Título**: En un campo para introducir texto se le pedirá al profesor de manera obligatoria que introduzca el nombre de la sesión con el cual se almacenará en *Datastore*, para facilitar el acceso en posteriores sesiones.
- **Imagen de fondo**: Se le proporcionará al profesor un mecanismo para poder elegir una imagen de fondo para sus sesiones. Dicha imagen, se almacenará en *Datastorage* para poder recuperar fácilmente su url y mostrarla como imagen de fondo de la sesión.

Debido a que se le permite al usuario añadir imágenes de fondo en sus sesiones, se deben establecer ciertos mecanismos para guardar estas imágenes y posteriormente acceder a ellas. Aquí es donde entra Google Datastorage, que nos permite almacenar ficheros, devolviéndonos la url de los mismos.

En el formulario del título, se incluye un *input* de tipo *file* donde el usuario selecciona la imagen que desea establecer, mostrándole una vista previa de la misma en el propio formulario. Una vez el usuario continua con la creación de la sesión, pasando a las preguntas, la imagen queda almacenada internamente como una cadena.

5.5.2 Preguntas de las sesiones

Continuando con la creación de sesiones, el profesor ahora debe indicar las preguntas que desea incluir en la sesión, incluyendo las cuatro respuestas necesarias y una correcta de manera obligatoria.

El formulario que se ha creado para esta parte, recoge la siguiente información:

- **Enunciado**: el profesor introduce en forma de cadena de texto el enunciado de cada pregunta de la sesión.
- **Respuesta**: se le proporciona al usuario 4 *inputs* de tipo texto en el que escribirá las posibles respuestas a la pregunta anteriormente formulada.
- **Respuesta correcta**: al lado de cada respuesta, el profesor debe seleccionar una única respuesta de manera obligatoria utilizando una *check-box*.
- **Imagen**: de manera opcional, al igual que en el título de la sesión, se le da al profesor la opción de añadir una imagen a la pregunta que esté realizando en ese momento.

Cuando el profesor haya indicado que ha terminado la sesión, esta se tiene que almacenar correctamente en nuestro sistema, manteniendo la información necesaria para luego poder mostrar la sesión al profesor al autenticarse y proporcionar los mecanismos necesarios para que los alumnos se puedan unir posteriormente.

Para mantener la persistencia del sistema, se añaden dos nuevas tablas en *datastore* a la anteriormente creada, ***Quiz*** y ***Questions***. Las propiedades de la tabla *Quiz* son las siguientes:

- **ID**: Al igual que en la tabla *Users*, *Datastore* nos genera un identificador para cada entidad de la tabla.
- **BackgroundImg**: propiedad de tipo cadena que corresponde a la url de la imagen de fondo almacenada en *Datastorage*.
- **Pin**: indica el número de la sesión que los usuarios deberán utilizar para poder acceder a ella. Está almacenada como una propiedad de tipo entero.
- **Questions**: *array* de *keys* correspondientes a las preguntas por las que está compuesta dicha sesión. Se almacenan con el ID que nos proporciona *Datastore* para poder recuperarlas más fácilmente.
- **Title**: cadena de texto donde se guarda el título de la sesión.

Por otro lado, las propiedades de la tabla *Questions*:

- **ID**: como ocurre con las dos tablas anteriormente mencionadas, se nos proporciona un identificador único para cada entidad.
- **Answers**: propiedad de la entidad de tipo *array* que contiene en forma de cadena de texto las respuestas a la pregunta del profesor.
- **Correct Answer**: se almacena la respuesta correcta en forma de entero que corresponde con la posición del *array* de la propiedad *Answers*.
- **ImgUrl**: al igual que ocurre con *BackgroundImg* en la tabla *Quiz*, se mantiene la url de la imagen en *Datastorage* de cada pregunta (en caso de que haya sido añadida por el usuario).
- **Statement**: propiedad de la entidad de tipo cadena de texto que contiene el enunciado de la pregunta.

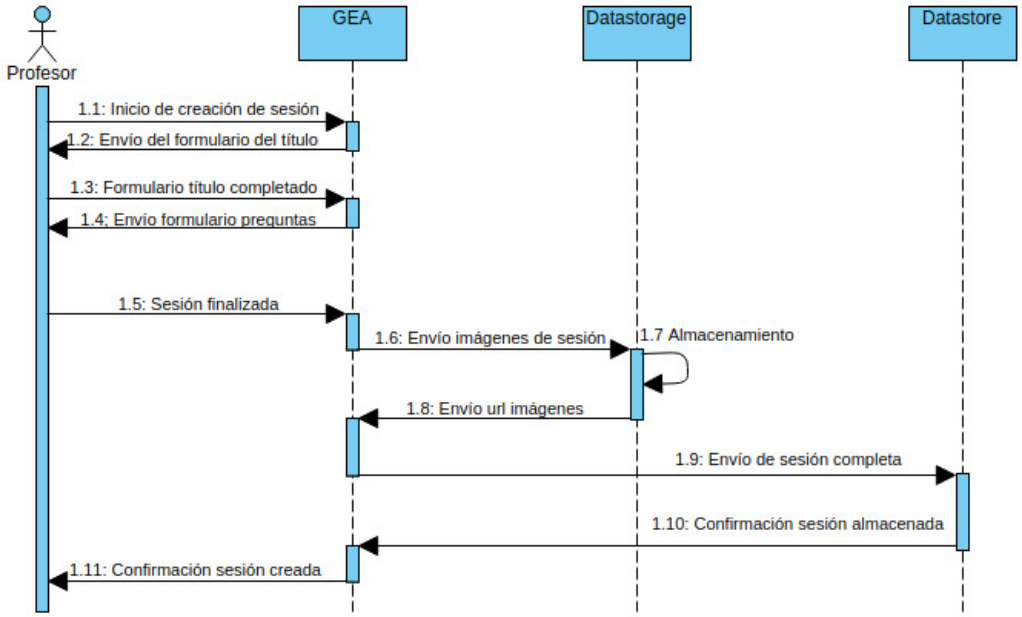

Figura 5.3: Diagrama de secuencia de la creación y almacenamiento de una sesión

En la figura 5.3 se muestra el diagrama de secuencia correspondiente a la adición de sesiones o *quizs* donde intervienen tanto el sistema GEA como Googla *Datastore* y *Datastorage*:

- Inicio de creación de sesión: el profesor tras autenticarse en el sistema, accede a la funcionalidad para poder crear sesiones.
- Envío del formulario del título: tras recibir la petición por parte del profesor, el sistema le envía al profesor el formulario correspondiente para poder introducir los datos necesarios correspondientes al título de la sesión.
- Formulario título completado: una vez rellenados todos los campos del formulario, el profesor envía los mismos al sistema.
- Envío formulario preguntas: al recibir los datos del formulario del título, el sistema envía el formulario de las preguntas para que el profesor añada las que desee.
- Sesión finalizada: cuando el profesor termina con la sesión y ha añadido todas las preguntas, se avisa al sistema.
- Envío imágenes de sesión: con toda la información ya necesaria, lo primero que hace el sistema es enviar la información relativa a las imagenes a *datastorage* para que estas puedan ser subidas.
- Almacenamiento: una vez recibidas las imágenes, *datastorage* las almacena, creando una url específica para cada una.

- Envío url imágenes: al completar la subida de todas las imágenes, *datastorage* devuelve al sistema las url asociadas a dichas imágenes de manera ordenada.
- Envío de sesión completa: con la url de las imagenes, el sistema ya tiene todos los datos necesarios para poder guardar la sesión correctamente y los envía a *datastore*. Dentro de esta información, se le proporciona también el e-mail del profesor para poder recuperar las sesiones asociadas al mismo cuando se autentique en el sistema.
- Confirmación de sesión almacenada: cuando *datastore* termina de almacenar la sesión de manera completa, envía la confirmación al sistema.
- Confirmación sesión creada: una vez el sistema recibe la confirmación de que la sesión ha sido creada, se le confirma al usuairo y se le devuelve a la pantalla de inicio cuando se autentica.

En lo que respecta a la **implementación** de esta funcionalidad del sistema, hay varios aspectos a mencionar:

Subida de imágenes

Tanto las imágenes de fondo como las imágenes de cada pregunta siguen el mismo proceso para ser almacenadas en *datastorage*. Debido al uso que se hará de estas, no se hará distintición en la manera de tratar a dichas imágenes, solo se diferenceriará a la hora de recuperarlas, puesto que una se utiliza como personalización de fondo y el resto se usa en cada una de las preguntas, en el caso de que exista.

En en listado 5.5 se muestra el código correspondiente a la subida de imágenes en *datastorage*. El argumento `imageData` se corresponde con la imágen previamente computada, de manera que pueda tratarse sin ningún problema.

Lo primeros que debemos de comprobar es si el parámetro mencionado anteriormente está vacio o no, debido a que se da la posibilidad de no añadir imágenes en las preguntas. En caso de que la imagen exista, se crea un nuevo nombre para almacenarla en *datastorage* utilizando la hora actual y un número aleatorio para evitar conflictos de nombres, y añadiendo la url de datastorage, con la configuración previa, como se ve en la línea 7.

En la línea 14 del listado, vemos cómo se devuelve la url de la imagen una vez ya almacenada en *datastorage*.

Este proceso se repite tantas veces como imágenes haya en la sesión del profesor, incluyendo la imagen de fondo. Una vez almacenadas todas las fotos y obtenidas sus urls correspondientes en el sistema, se pasa a guardar la sesión en *datastore*.

```
1  function uploadQuestionImage(imageData, callback) {
2      if(imageData === ''){
3        callback(null, '');
4      }else{
5        var filename = '' + new Date().getTime() + "-" + Math.random();
6        var file = bucket.file(filename);
7        var imageUrl = 'https:///'+config.bucketName+'.storage.googleapis.com/'+filename;
8        var stream = file.createWriteStream();
9        stream.on('error', callback);

11       stream.on('finish', () => {
12         file.makePublic( (err) => {
13           if (err) return callback(err);
14           callback(null, imageUrl);
15         });
16       });

18       stream.end(imageDataURI.decode(imageData).dataBuffer);
19     }
20 }
```

Listado 5.5: Subida de imágenes en *datastorage*

Almacenamiento de la sesión en *datastore*

Con toda la información necesaria y recibida la confirmación de que todas las imágenes han sido subidas, el siguiente paso es guardar la sesión al completo en *datastore*. El listado 5.6 muestra el código implementado para realizar dicha función.

En la línea número 4 se consulta el número de sesiones ya creadas para poder asignarle el próximo pin a la sesión que se quiere crear en ese momento. En la línea 5 se crea la entidad que será almacenada en la tabla *Quiz* y finalmente en la línea 15 se le da la instrucción para guardar dicha entidad, devolviendo el **ID** que proporciona *Datastore*.

```
function uploadQuiz(quiz, questions, backgroundImgUrl, callback){
    const quizKey = datastore.key(modelQuiz);

    numberOfQuizs((number) => {
      var entity = {
        key: quizKey,
        data: {
          Pin: number + 1,
          Questions: questions,
          Title: quiz,
          BackgroundImg: backgroundImgUrl
        }
      };

      datastore.save(entity)
      .then(() => {
        callback(quizKey)
      });
    });
}
```

Listado 5.6: Adición de *quizs* o sesiones en *datastore*

5.6 Eliminación de sesiones

Al igual que el profesor puede crear nuevas sesiones, se le deben proporcionan los mecanimos necesarios para poder eliminarlas en el caso de que ya no vaya a usarlas más. Para esta funcionalidad, es importante tener en cuenta que debe ser eliminada la referencia a esa sesión en la entidad del profesor correspondiente en *datastore*, para que al acceder en el sistema, no aparezcan sesiones que hayan sido eliminadas, produciendo errorres en la plataforma.

En la figura 5.4 se muestra el diagrama de secuencia que representa la eliminación de un quiz en el sistema. A continuación, se explican más detalladamente cada una de las iteraciones del diagrama:

- Eliminar Quiz: el profesor selecciona el quiz que desea eliminar y se lo comunica al sistema.
- Envío e-mail profesor: el sistema hace una petición a *datastore* enviándole el e-mail del profesor para que le devuelva los *quizs* que ha creado.
- Envío *quizs* asociados: al recibir la petición, se realiza la consulta en *datastore* y le devuelve en forma de lista los *quizs* del profesor.
- Envío pin del *quiz* a eliminar: al recibir los quizs, almacena las *keys* de los mismos para su posterior uso y envía el pin del *quiz* que quiere eliminar.
- Eliminar *quiz*: al recibir los datos necesarios, *datastore* elimina la entidad correspondiente a ese *quiz*.

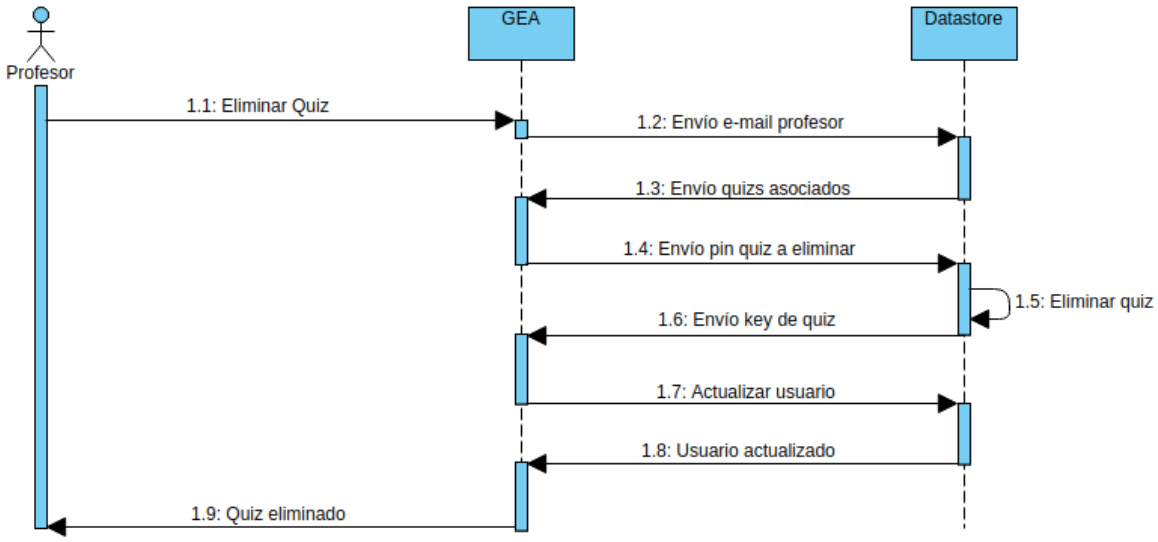

Figura 5.4: Diagrama de secuencia correspondiente a la eliminación de una sesión

- Envío *key* del *quiz*: tras eliminar la entidad, envía la *key* de esa entidad para que la lista de las sesiones creada por el usuario pueda ser actualizada.
- Actualizar usuario: al recibir la *key*, el sistema comprueba cual es la *key* que coincide con las que ese profesor tiene asociadas, y la elimina de la lista en el sistema, es decir, actualiza los valores de la misma para que se muestren de manera correcta. Hecho esto, envía los datos necesarios a *datastore* para que quede registrado en la entidad.
- Usuario actualizado: *datastore* envía la confirmación de que el usuario ha sido actualizado.
- Quiz eliminado: el sistema informa al usuario de que el quiz ha sido eliminado.

El listado 5.7 muestra el código asociado a esta función. En la línea 2, se crea la consulta para obtener las sesiones asociadas al profesor actual. Después, se realiza otra consulta para obtener la *key* que genera automáticamente *datastore* del *quiz* que se quiere eliminar. A continuación, utilizando esa *key*, se elimina de la lista de referencias el *quiz* y se actualizan los datos del profesor en datastore en la línea 8. Para finalizar, en la línea 14, se elimina el *quiz* deseado.

```
function deleteQuiz(quizPin, userEmail){
    var query = datastore.createQuery(modelUser).filter('email', '=', userEmail);
    datastore.runQuery(query, (err, user) =>{
      getQuiz(quizPin, (err, quiz, key) => {
        for(let i=0; i< user[0].quizs.length; i++){
          if(user[0].quizs[i].id === quiz[0][datastore.KEY].id)
            user[0].quizs.splice(i);
            datastore.update(user[0]);
        }
      });
    });

    getQuiz(quizPin, (err, quiz, key) => {
      datastore.delete(quiz[0][datastore.KEY]);
    });
  }
```

Listado 5.7: Eliminación de *quizs*

5.7 Mecánicas de las sesiones

A continuación, se pasa a presentar los mecanismos utilizados para la realización de las sesiones en tiempo real. Es importante destacar que el profesor deberá estar presente en dichas sesiones para poder seguirlas y administrarlas, ya que el control de estas sesiones reside en él.

5.7.1 Unión a las sesiones

Todas las sesiones creadas tienen como propiedad un *pin* con el que los alumnos pueden acceder a ellas. Este pin es generado automáticamente cada vez que se crea una sesión y se le muestran al profesor los pins de las sesiones creadas al autenticarse en el sistema.

Una vez que el profesor le proporciona a los alumnos el pin, en la página de inicio de GEA se le pide a los alumnos que introduzcan el pin para poder acceder a las sesiones, además de un nombre para que el profesor pueda identificar a los alumnos. Si no se introduce el nombre, la sesión será anónima, Las contestaciones de los alumnos no quedarás registradas asociadas a su identificador.

Al introducir el pin, el sistema realiza una consulta a *datastore*, pidiendo que le devuelva la sesión o *quiz* asociado al pin introducido. Cuando la información de la sesión es recibida, se le muestra al usuario y ya puede contestar a la pregunta.

El listado 5.8 nos muestra la **implementación** de la consulta que se realiza para obtener la sesión. En la línea 2 se crea la consulta, indicando que devuelva los *quizs* cuyo pin sea igual al pasado como argumento de la función. En la siguiente línea, se ejecuta la consulta y se devuelve el *quiz* y el ID asociado al mismo.

```
1 function getQuiz(pin, callback) {
2     var query = datastore.createQuery(modelQuiz).filter('Pin', '=', Number(pin));
3     datastore.runQuery(query, (err, quiz) => {
4         callback(err, quiz, datastore.KEY);
5     });
6   }
```

Listado 5.8: Consulta realizada en *datastore* para obtener el quiz dado un pin

5.7.2 Realización de sesiones

Debido a que la realización de las sesiones es en tiempo real, se necesitan mecanismos que nos permitan una comunicación en tiempo real para poder enviar las respuestas de los alumnos al profesor. Aquí es donde intervienen los ***sockets***.

Un *socket* es un método de comunicación entre dos programas, un cliente y un servidor, en una red. Para que ambos programas se comuniquen es necesario cumplir dos requisitos:

- Que un programa tenga los mecanimos para localizar al otro.
- Que ambos programas sean capaces de intercambiar información.

Por esto, se requieren de dos recursos:

- Un par de direcciones de protocolo (en este caso IP), que identifiquen el origen y el destino.
- Un par de números de puertos que identifiquen al programa dentro de la máquina donde se ejecute.

Por esto necesitamos crear un nuevo servidor que nos permita la comunicación por *sockets* entre el alumno y el profesor. Como la plataforma estará funcionando sobre un protocolo HTTP, se utilizarán *websockets*, que ofrecen mejor funcionalidad para este tipo de protocolos ya que no funcionan a tan bajo nivel. Además, se mantendrá una lista de los alumnos que se han unido a la sesión en el caso de que se necesitara enviar un mensaje *broadcast* (es decir, a todos los incluidos en dicha lista) para poder por ejemplo, pasar a la siguiente pregunta.

Para conseguir esta funcionalidad, se utilizará *socket.io* en la parte del servidor y *socket.io-client*, bibliotecas de *Node.js* que nos facilitan muchísimo el trabajo para conseguir la comunicación entre ambas partes.

Existe un problema debido a la seguridad de GAE ya que no permite la implementación de *websockets* en el propio sistema que se ha desplegado, es por esto que se necesita implementar el servidor para *websockets* de manera independiente. Se podría realizar otra aplicación en GAE que únicamente se encargara de los *websockets*, pero debido a los costes de la plataforma, el servidor se implementará en Google Compute Engine, que nos permite tener el servidor ejecutándose en la nube de manera continua sin preocuparnos de la escabilidad, ya que incrementará los recursos según se vayan necesitando.

Como se ha mencionado anteriormente, se necesitan de dos direcciones IP de los dispositivos donde se ejecute y unos puertos para asociar el programa en el dispositivo. En el caso de las IP, se necesita saber la dirección del servidor para poder establecer la comunicación. Accediendo a la página de Google Compute Engine, seleccionaremos nuestro servidor y este nos proporcionará la IP de la máquina virtual en la que se está ejecutando, la cual deberemos facilitar a la parte del cliente para poder establecer la comunicación. En el caso de los puertos, se asignará un puerto que no sea utilizado por otros programas.

En la figura 5.3 se muestra el diagrama de secuencia para poder entender mejor cómo funciona el sistema cuando un alumno entra en la sesión y responde a una pregunta.

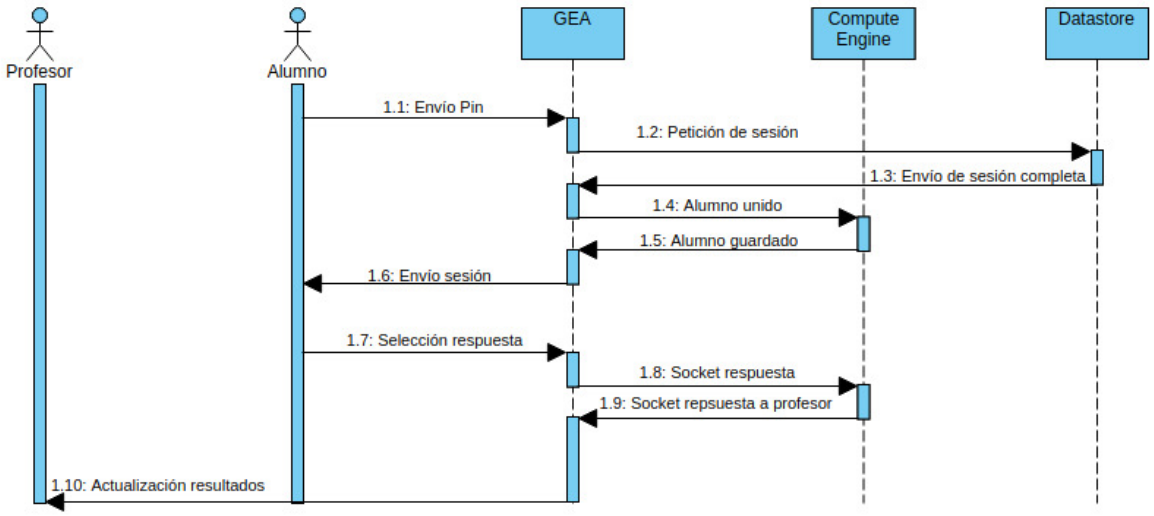

Figura 5.5: Diagrama de secuencia correspondiente a la realización de una pregunta en una sesión

A continuación, se detalla en profundidad cada una de las fases de esta funcionalidad:

- Envío Pin: Cuando el alumno ya conoce el pin de la sesión que se pretende realizar, lo introduce, junto con su nombre y lo envía al sistema.
- Petición de sesión: GEA, al recibir el pin, realiza una consulta a *datastore* para que este le devuelva toda la información relativa al *quiz* con el pin introducido por el alumno.
- Envío de sesión completa: localizada la sesión correcta, *datastore* la envía al sistema para que este se la muestre a los usuarios posteriormente.
- Alumno unido: el sistema GEA envía mediante un *websocket* al servidor que se está ejecutando en *Compute Engine* el identificador del alumno, indicando así que se ha unido a la sesión.
- Alumno guardado: una vez recibido el *websocket* por parte de GEA, se envía otro *websocket* hacia GEA indicando que el alumno ha sido registrado en esa sesión.
- Envío sesión al alumno: Al comprobar que el alumno ya ha sido registrado en la sesión, el sistema le muestra la primera pregunta de la sesión para que pueda responder.
- Selección respuesta: El sistema le muestra la pregunta y las posibles respuestas actuales al alumno. Este selecciona la respuesta que él desee pulsando en el botón correspondiente.
- Socket respuesta: Seleccionada la respuesta y enviada a GEA, se envía un *socket* hacia el servidor para que dicha respuesta quede registrada y poder comunicarselo al profesor.
- Socket respuesta a profesor: Al recibir la respuesta, el servidor de *websockets* envía un *websocket* al profesor con la respuesta que ha sido seleccionada por un alumno.
- Actualización resultados: cuando el profesor recibe la respuesta enviada por el alumno, la tabla de resultados que tiene se actualiza de manera automática, colocando el primera posición a los alumnos con mayor puntuación.

En cuanto a los detalles de la **implementación**, lo más destacable es la comunicación mediante *websockets*, la recuperación de los *quizs* es similar a cuando un usuario se autentica y se le muestran las sesiones que ha creado.

En el listado 5.9 se muestra una versión simplificada del código correspondiente al servidor destinado al manejo de *websockets*. Como se ha mencionado anteriormente, es necesario que una de las partes sea capaz de localizar a otra para poder establecer la comunicación, en este caso, es el cliente quien conoce la dirección IP del servidor, por lo que no hace falta implementar mecanismos para que el servidor conozca las direcciones de los distintos clientes.

En las líneas 1-4 del código, se inicializa lo necesario para el correcto funcionamiento del servidor, a destacar en la línea 4, el uso de la librería `socket.io` que nos facilita la implementación del servidor. En la línea 6 se inicia el servidor en el puerto anteriormente declarado y se mantiene a la espera de nuevas peticiones.

En las siguientes líneas se implementan las distintas acciones que debe hacer el servidor al recibir los *websockets* con los identificadores que aparecen. Por ejemplo, en el caso de recibir un *websocket* con `session`, se unirá el alumno que haya enviado ese socket a la lista de alumnos asociados a esa sesión.

```
1  const PORT = 65080;
2  const expres = require('express');
3  var server_socket = require('http').Server(server_socket);
4  var io = require('socket.io')(server_socket);

6  server_socket.listen(PORT, () => {
7    console.log(`Listening web socket server on port ${PORT}`);
8  });

10 io.on('connection', (socket) => {
11     socket.on('disconnect', () => {
12     });

14     socket.on('session', (data) => {
15         console.log('student joined');
16         socket.join(data);
17     });

19     socket.on('result', (data) => {
20         socket.broadcast.to('admin'+data.pin).emit('recieve-result', data);
21     });
22 }
```

Listado 5.9: Fragmento del servidor de *sockets*

5.7.3 Administración de sesiones

En todo momento el profesor debe ser consciente de lo que se le está mostrando a los alumnos, poder ver los resultados de los mismos y controlar las preguntas que se le van mostrando. Es por esto que se le deben proporcionar los mecanimos necesarios para poder realizar las funciones anteriormente enumeradas.

Para todo esto se debe diseñar una vista que permita acceder a las distintas funcionalidades de manera fácil e intuitiva, a ser posible utilizando botones para cada una de ellas. En el caso de poder ver la sesión actual, es tan sencillo como abrir una ventana nueva donde aparezca la sesión. Se ha utilizado un componente de *React* que nos permite realizar dicha tarea de

manera simplificada, simplemente indicando como argumento la url del sitio a dónde se desea ir.

De manera similiar al envío de respuestas por parte de los alumnos, el sistema GEA envía mediante *websockets*, desde el rol del profesor, las instrucciones para avanzar o retroceder de pregunta. Utilizando un mensaje de tipo *broadcast*, el sistema indica a los dispositivos de todos los alumnos unidos a esa sesión que debe cambiar la pregunta, en ese momento, el estado del *front-end* cambia y muestra la pregunta correspondiente, que ya había sido almacenada previamente debido a que se envía el quiz de manera completa.

La funcionalidad más compleja e interesante y que permite ver al profesor cómo se está desarrollando su sesión, consiste en una tabla de resultados en tiempo real que muestra los resultados de todos los alumnos, es decir, contestaciones dadas a las preguntas, puntuación total obtenida hasta ese momento y además ordenados según su puntuación. Las respuestas se deben ir almacenando en el sistema y estar asociadas a un usuario, para poder ir ordenándolas y mostrarlas en la tabla, es por lo que se ha utilizado una lista para almacenar estos resultados. Cada elemento de la lista, será un elemento de tipo *result* que está compuesto por dos cadenas, que corresponden con el nombre del usuario y la puntuación que lleva hasta el momento, y otra lista con las respuestas del usuario.

Como se ha mencionado, los resultados se muestran en orden de mayor a menor puntuación en cada momento. Aprovechando el método de ordenación `sort` que nos ofrece *JavaScript* para sus elementos de tipo *Array*, sobreescribimos el criterio de ordenación utilizando como tal, la puntuación de cada uno de los usuarios.

Así, cada vez que se recibe un resultado, el sistema comprueba si ese resultado es de un usuario que ya ha sido almacenado en la lista de resultados, en caso de que no, lo añade. Si el usuario ya existe, se hacen los ajutes necesarios en la lista de resultados (ya que en un principio, la lista correspondiente a las respuesta del usuario, está compuesta por '-' para que quede estéticamente bien la tabla) y se añade la respuesta dada. Tras esto, se ejecuta el método `sort` sobre la lista y se actualiza la tabla con la lista *results*.

En cualquier momento, se le da la posibilidad al profesor de poder descargar en formato *.csv* los resultados de la sesión. Para que esto sea posible, se debe mantener una estructura de datos adaptada al formato CSV que nos permita extraer los resultados de la sesión.

5.8 Organización y componentes del proyecto

Se muestra a continuación la jerarquía de directorios en las tablas 5.1 y 5.2 del sistema correspondientes al *front-end* y *back-end*. Como se ha mencionado anteriormente, la parte del código correspondiente a la parte visual e interacción con el usuario corresponde al *front-end* y la parte de la lógica y procesamiento de los datos necesarios corresponde con el *back-end*.

Front-end	build/	Carpeta generada automáticamente que nos permite desplegar el sistema en Google App Engine	
	node_modules/	Carpeta creada automáticamente donde se almacenan los archivos necesarios para node.js	
	public/	Carpeta que contiene archivos del sistema como el icono mostrado en el navegador	
	src/	components/	Componentes creados con React necesarios para implementar la vista del sistema
		css/	Carpeta que contiene los archivos correspondientes al estilo
		fonts/	Archivos correspondientes a las tipografías utilizadas en GEA
		image/	Archivos que contienen las distintas imágenes utilizadas en el sistema

Tabla 5.1: Jerarquía utilizada en el *front-end*

La parte más importante del *front-end* corresponde con los componentes desarrollados en *React* que constituyen toda la parte de la vista del sistema. La manera de implementar estos componentes es sencilla. Como se explica en el capítulo 3, los componentes de *React* poseen un método `Render` que renderizan los elementos HTML implementados en el sistema.

Similiar a un diagrama de clases, en la figura 5.6 se muestran los componentes implementados en *React*, de manera que la relación existente entre estos sea más fácil de entender.

A continuación, se enumeran los distintos componentes implementados en GEA y se hace una breve descripción de los mismos:

- **AddQuiz**: proporciona al usuario el formulario para poder añadir nuevas sesiones. Dentro del estado del mismo se mantienen algunas propiedades como el e-mail del profesor, las distintas preguntas y nombre de la sesión, etc. Además, proporciona funciones para poder comunicar al *back-end* las distintas preguntas que se desean almacenar y la sesión al completo.
- **App**: componente raíz del sistema. Es el componente que se renderiza en la página del inicio del sistema, donde se incluyen los componentes *SessionPick*, *HorizontalMenu*, y *BakcgroundImage*, dando así la opción a los profesores de poder autenticarse en el sistema o a los alumnos de introducir el pin de la sesión desde la página de inicio.
- **BackgroundImage**: componente utilizado para poder establecer una imagen de fondo en las sesiones creadas por los profesores.
- **HorizontalMenu**: permite colocar un menú horizontal desde el cual el usuario puede autenticarse en el sistema o volver a la página de inicio.
- **Popup**: como se ha mencionado anteriormente, al profesor se le da la opción de ver el estado actual de la sesión. Este componente da la posibilidad de que aparezca en una nueva ventana la sesión actual.
- **Profile**: al iniciar sesión en el sistema, este es el componente que se renderiza para mostrar la vista correspondiente al usuario. Este componente, está compuesto a su vez por el componente *QuizList*, y *UserProfile*.
- **Question**: componente que muestra la pregunta actual de la sesión. Además de esto, permite la interacción con el usuario para que pueda responder a las mismas preguntas.

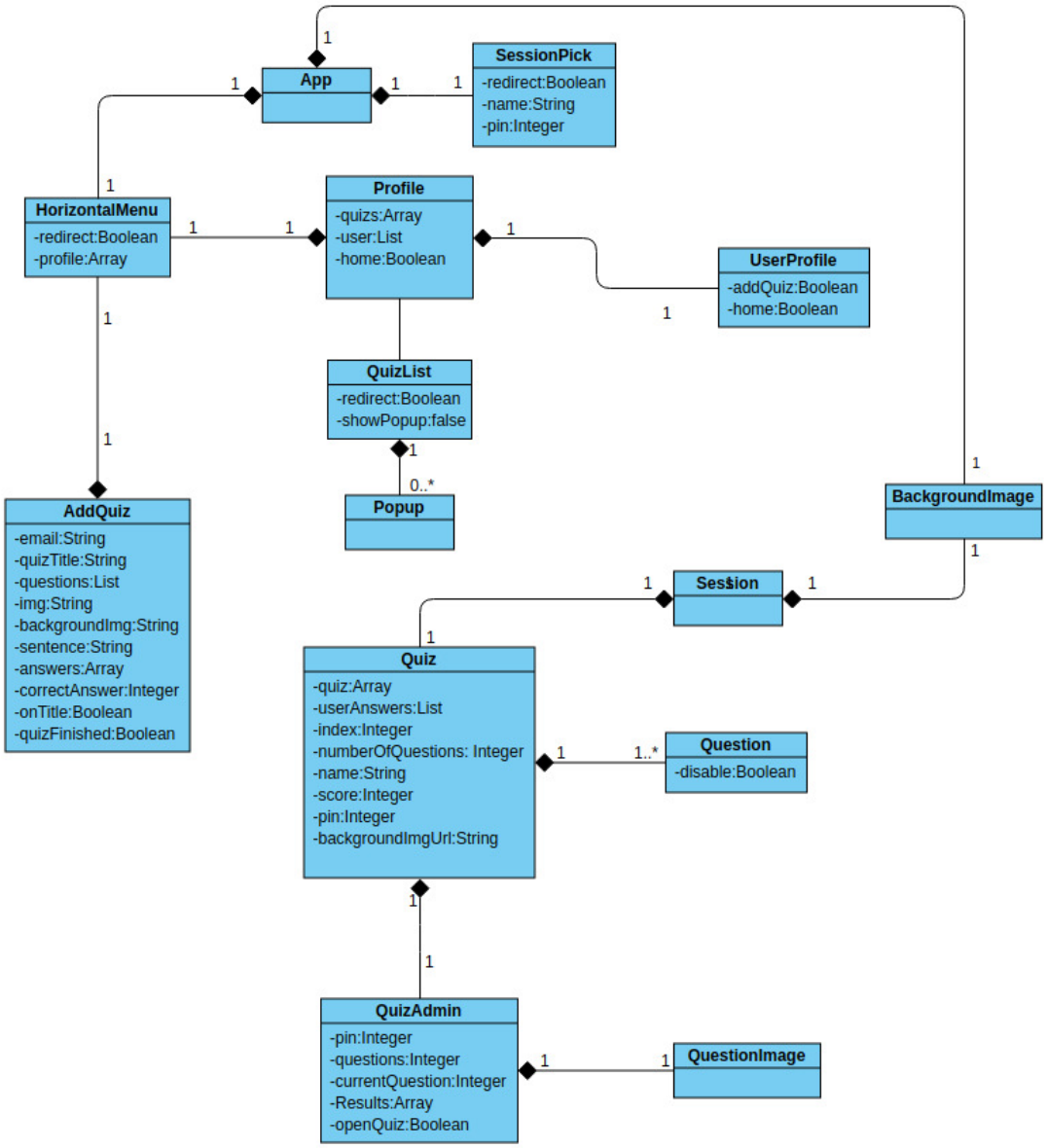

Figura 5.6: Diagrama de clase correspondiente a los componentes implementados en *React*

- **QuestionImage**: da la posibilidad al profesor de poder añadir imágenes en las preguntas correspondientes a la sesión que está realizando.
- **Quiz**: es el componente principal para la realización de sesiones. Está compuesto por tantos comoponentes *Question* como preguntas haya en el sistema. Matiene en su estado interno y envía el *socket* correspondiente para que sea recibido por el servidor de *sockets* con la respuesta del alumno. las respuestas del usuario

- **QuizAdmin**: este componente permite al profesor acceder a todas las funcionalidades para poder administrar la sesión. También, incorpora la tabla de resultados en tiempo real y los mecanismos necesarios para abrir la vista de los alumnos. En el caso de acceder a la sesión como administrador a través del comoponente *Quiz*, se renderizará el comoponente *QuizAdmin*.
- **QuizList**: renderizado cuando el usuario se autentica en el sistema, muestra todos los quizs que tiene asociados el profesor y hayan sido creados anteriormente.
- **Session**: componente inicial de las sesiones, aunque por el momento solo existen sesiones del tipo *quiz*, en un futuro se pretenden implementar nuevas para dotar de mayor variedad al sistema.
- **SessionPick**: componente que aparece en la página de inicio del sistema que nos permite introducir mediante un campo de texto el pin de la sesión a la que queremos acceder.
- **UserProfile**: muestra información relativa al usuario que se ha autenticado como su foto, o nombre.

La parte del *back-end* ya ha sido explicada anteriormente, haciendo énfasis en la comunicación de los componentes y las distintas funcionalidades que son responsabilidad del mismo.

En la tabla 5.2 podemos ver la jerarquía de los ficheros utilizada para la parte correspondiente al *back-end*.

Back-end	app.yaml	Archivo de configuración general.
	cliente/	Contendor donde se incluye los archivos relacionados con la parte del cliente
	config.js	Archivo de configuración para algunas dependencias de Google App Engine
	config.json	Archivo de configuración con las claves para OAuth2
	key.json	Archivo de configuración para OAuth2
	Makefile	Archivo para facilitar el inicio del sistema
	node_modules/	Carpeta creada automáticamente con dependecias de node.js
	package.json	Archivo de configuración donde se incluyen distintas dependencias y scripts
	quiz.js	Contiene el código encargado de manejar los quizs en datastore
	server.js	Contiene el código del servidor del sistema
	socket_server.js	Contiene el código del servidor de sockets implementado en Compute Engine
	user.js	Contiene el código encargado de manejar los usuarios en datastore

Tabla 5.2: Jerarquía utilizada en el *back-end*

5.8.1 Estructura de Datastore

El uso de Google App Engine nos da diferentes opciones para el almacenamiento de datos[3]: Persistent Disk (almacenamiento en bloques), Bigtable (bases de datos NoSQL de tabla amplia), Datastore (bases de datos NoSQL de documentos completamente administrada y escalable para aplicaciones web), Storage (almacenamiento de BLOBs), SQL (bases de datos

[3]https://cloud.google.com/storage-options/?hl=es-419

MySQL), Spanner (bases de datos relacional y esencial), BigQuery (*Enterprise Data Warehouse*) y Enterprise (espacio colaborativo para almacenar, editar y compartir archivos).

De entre todas las opciones que ofrece, las que más se adecúan y nos interesan para nuestro sistema como ya se han mencionado anteriormente son **Datastore** para las bases de datos del sistema y **Datastorage** para el almacenamiento de imágenes.

Sin embargo, *Datastore* no sigue un modelo relacional, sino que es una base de datos NoSQL jerárquica orientada a objetos. Algunas de las principales diferencias que podemos encontrar entre ellas son:

- Se sustituyen las relaciones por **tipos**, registros por **entidades** y los campos por **propiedades**.
- Cada nueva entidad genera una clave o ***key*** que es el equivalente a la clave primaria en las bases de datos relacionales. Además, cada entidad de cada tipo puede hacer referencia a otras entidades del mismo u otro tipo, utilizando una propiedad especial que permite establecer las jerarquías.
- La **flexibilidad** que ofrece Datastore con respecto a las bases de datos SQL es mucho mayor. Permite almacenar objetos de tipo JSON o listas como valor de una propiedad de la entidad. Debido a su orientación a objetos, facilita la lectura de la entidad y su posterior instancia en el sistema.
- Las **consultas** que se pueden realizar sobre la base de datos son más restringidas, por ejemplo, operaciones como la unión (*join*) no están permitidas.

Una de las principales ventajas que nos proporciona *datastore* es la **escalabilidad**, es decir, el rendimiento de esta no decrece aún cuando los datos que maneja van creciendo. En caso de que se quiera hacer una escritura, los datos son distribuidos y para las escrituras, la restricción en las consultas y el sistema de índices preconstruidos hace que las búsquedas sean igual de rápidas independientemente de la cantidad de datos que se manejen. Permite además el uso de transacciones atómicas y tiene muy buena integración con lenguajes de scripting y basados en objetos.

Al igual que se ha hecho con la sección del *front-end*, en la figura 5.7 se muestra en modo de diagrama de clases, los modelos implementados en *datastore* para aclarar las relaciones existentes entre los distintos modelos.

Se puede ver claramente la relación existente entre los 3 modelos, por un lado, el modelo *User* posee un *Array* de referencias a preguntas pertenecientes al modelo *Quiz* utilizando el ID generado automáticamente por *Datastore* al crear nuevas entidades. Por otro lado, el modelo *Quiz* tiene otro *Array* con referencias a las preguntas de las que está compuesto al modelo *Question*, al igual que con el anterior modelo, estas referencias se consiguen utilizando el ID proporcionado por *Datastore*.

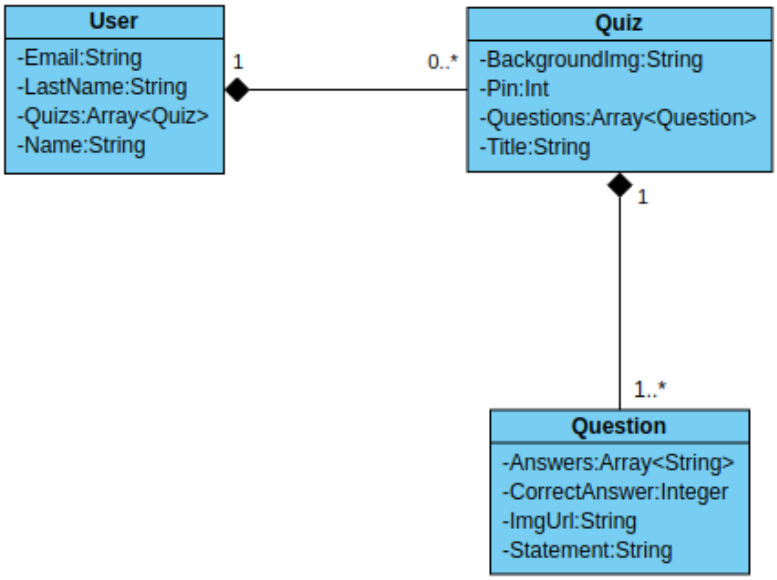

Figura 5.7: Diagrama de clase correspondiente a los modelos utilizados en *datastore*

Capítulo 6

Resultados

En capítulos anteriores se han discutido los problemas que se pretenden resolver con el sistema, el estado del arte, los métodos de trabajo que se han llevado a cabo y la arquitectura que se ha implementado para el correcto funcionamiento del sistema. En este capítulo se mostrarán los resultados obtenidos al realizar el trabajo anteriormente expuesto y se plantearán los costes que ha supuesto.

6.1 Interfaz y funcionalidad del sistema

Desarrollado el sistema y probado en local, donde gracias a Google App Engine podemos simular el entorno al completo, se desplegará el sistema en los servidores que nos ofrece Google. En principio, Google proporciona una prueba gratuita de 300 dólares para utilzar las distintas funcionalidades de la plataforma, aunque mientras se use esta prueba gratuita, las funcionalidades serán limitadas[1]. Cuando se nos agoten los 300 dólares de prueba, Google nos comunicará mediante factura los recursos utilizados y nos cobrará automáticamente por sus servicios.

Para poder desplegar el sistema, primero deberemos hacer una *build* del mismo en local y utilizar esa versión. Estando en el directorio del cliente o *front-end*, ejecutamos la siguiente instrucción desde terminal:

```
npm run-script build
```

Una vez se ha completado la *build*, desde el directorio raiz del sistema, previa instalación y configuración de `gcloud`, ejecutamos el comando que se muestra a continuación:

```
gcloud app deploy
```

Posteriormente, para acceder al sistema a través de un navegador web, Google App Engine nos proporciona una url con el siguiente formato `https://<nombre_del_proyecto>.appspot.com`. En nuestro caso, la url proporcionada es: `https://gamificateemall.appspot.com`.

[1] `https://cloud.google.com/appengine/pricing`

Figura 6.1: Interfaz de la página inicial de GEA

Página inicial y acceso a sesiones Al acceder el sistema se mostrará la página de inicio (ver figura 6.1). Desde aquí podremos introducir el pin de la sesión y el nombre con el que queremos aparecer en la tabla de resultados. Además, desde el menu horizontal en la parte superior, los profesores se pueden autenticar en el sistema y crear nuevas sesiones o realizar una en clase.

Al ser *responsive* el sistema, si se accede a él a través de un *smartphone*, la interfaz se redimensionará y se ajustará para mostrar correctamente todos los elementos, como podemos ver en la figura 6.2

Figura 6.2: Interfaz de la página de inicio del sistema mostrada desde un *smartphone*

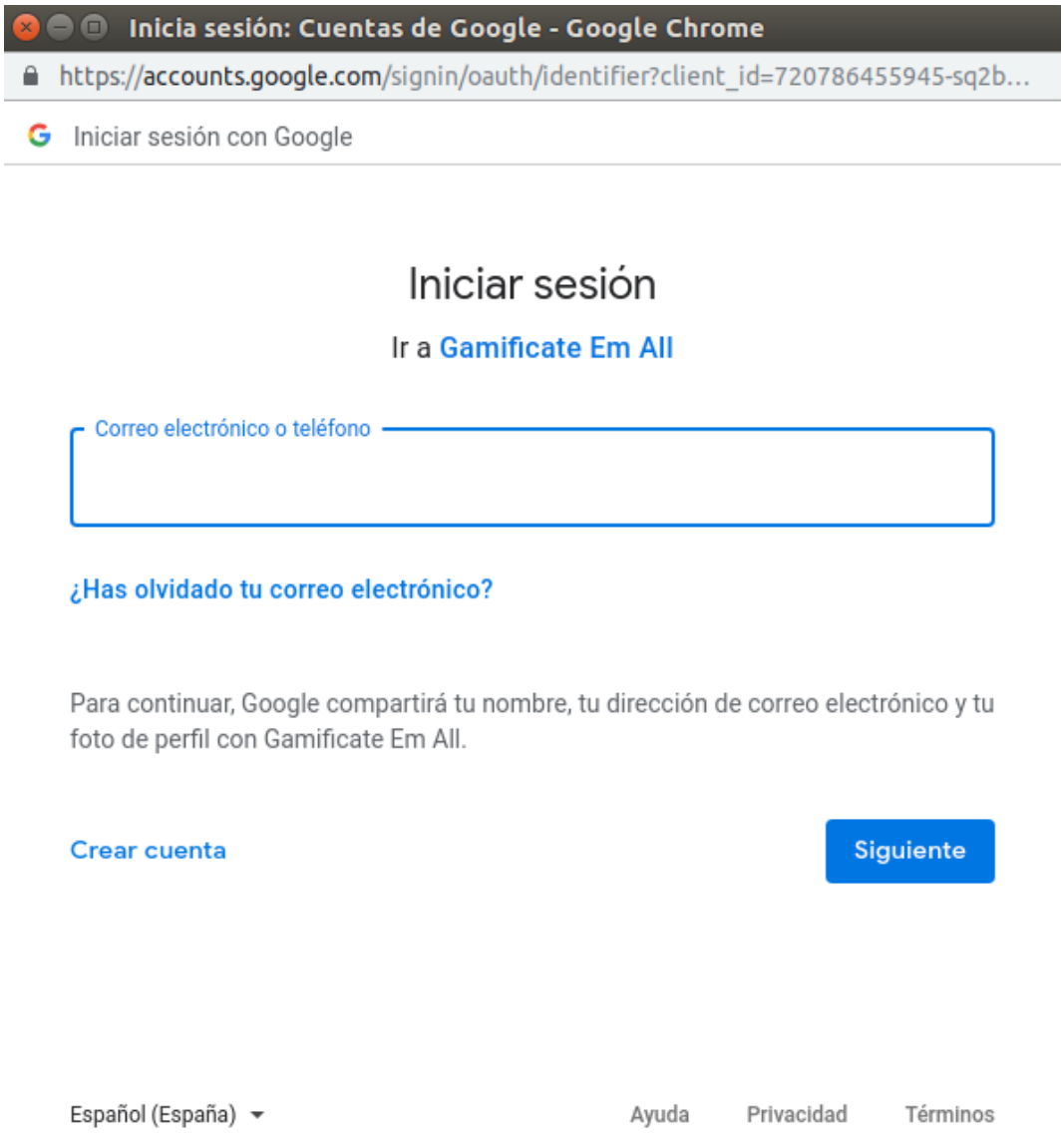

Figura 6.3: Formulario mostrado usando Google como proveedor seguro en OAuth2

Autenticación En el menú horizontal podremos hacer *login* utilizando nuestra cuenta de correo de Google. Al intentar hacerlo, nos aparecerá el siguiente formulario (ver figura 6.3) donde se piden los credenciales del usuario en Google.

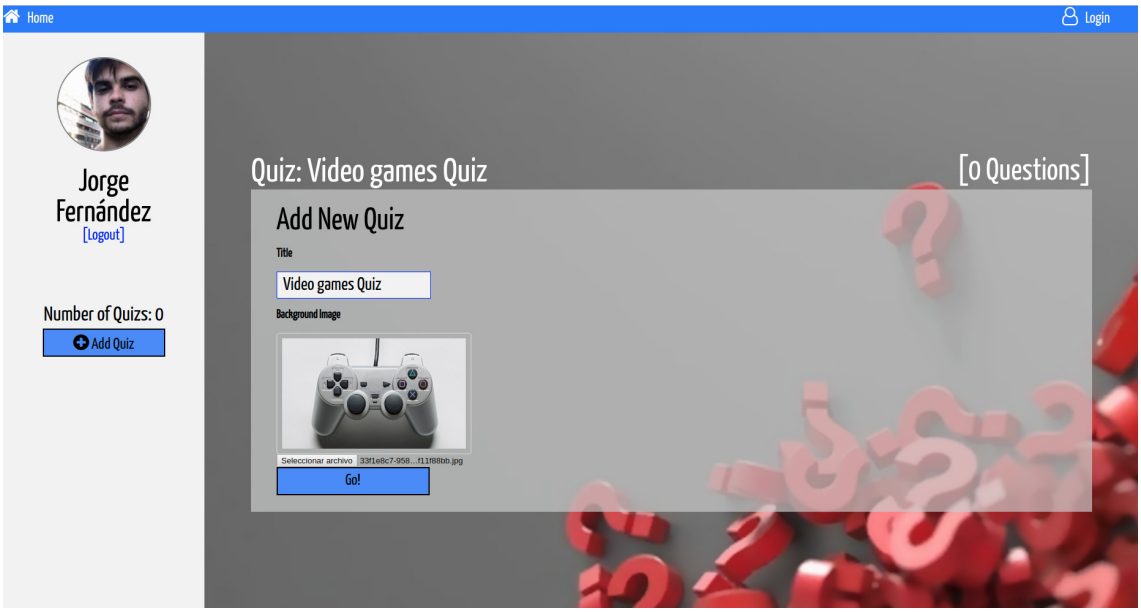

Figura 6.4: Formulario presentado para añadir el título y la imagen de fondo de la sesión

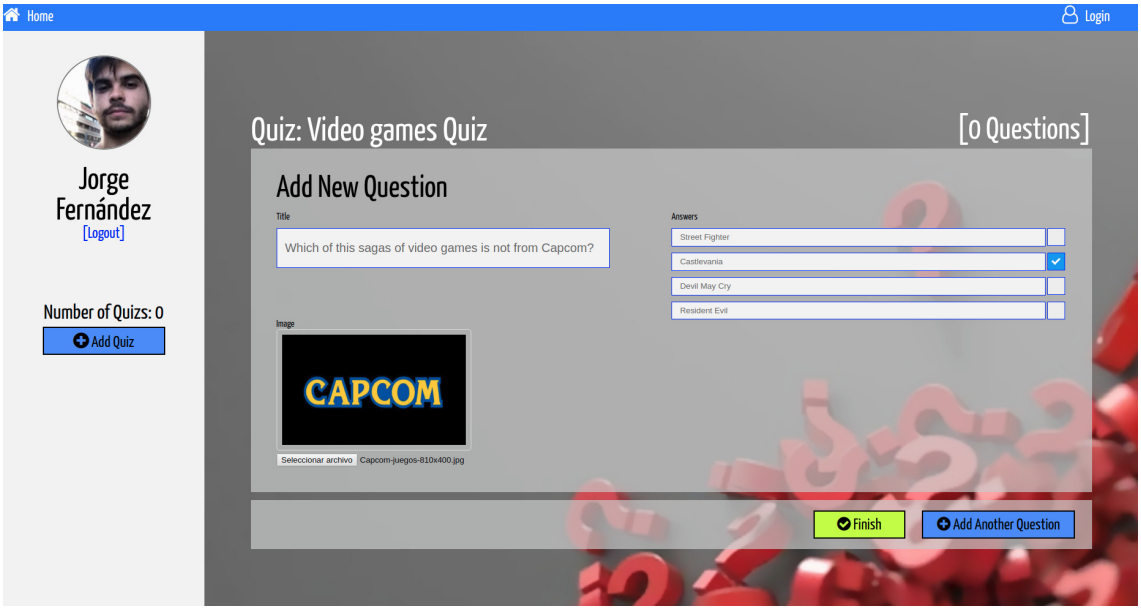

Figura 6.5: Formulario presentado para añadir preguntas en una sesión

Adición de sesiones de tipo Quiz Cuando el usuario desea crear una nueva sesión, se le mostrará primero un formulario para que añada el título y la imagen de fondo de la sesión (ver figura 6.4). Tras esto, el sistema renderizará el segundo formulario donde el profesor incluirá las preguntas que crea necesarias para la sesión (ver figura 6.5).

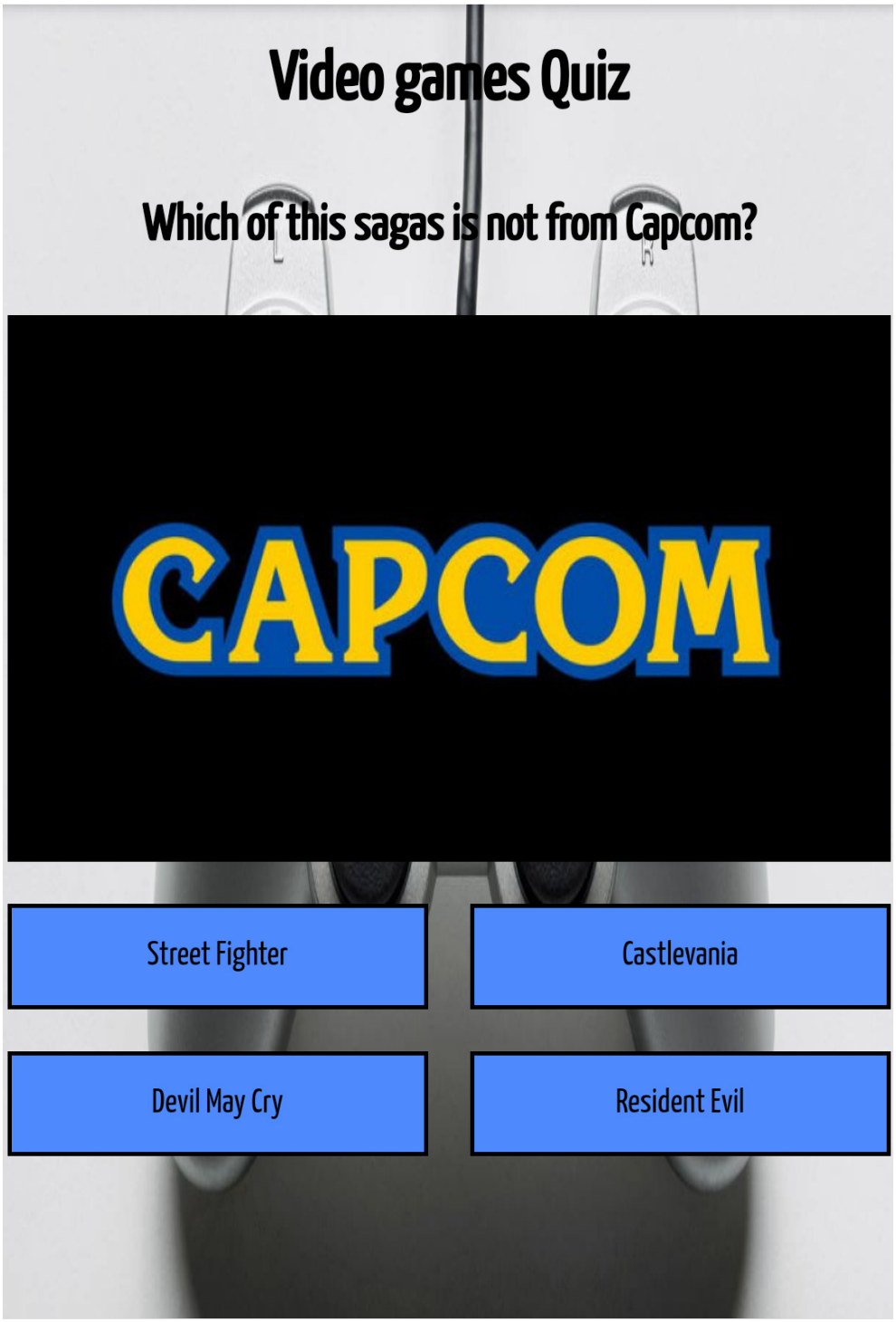

Figura 6.6: Pregunta mostrada en *smartphone*

Realización de sesiones Cuando los alumnos acceden a las sesiones a través de su *smartphone* introduciendo el pin correspondiente, se les mostrará la interfaz que se ve en la figura 6.6, desde la cual pueden seleccionar la respuesta y esperar a que el profesor cambie la pregunta.

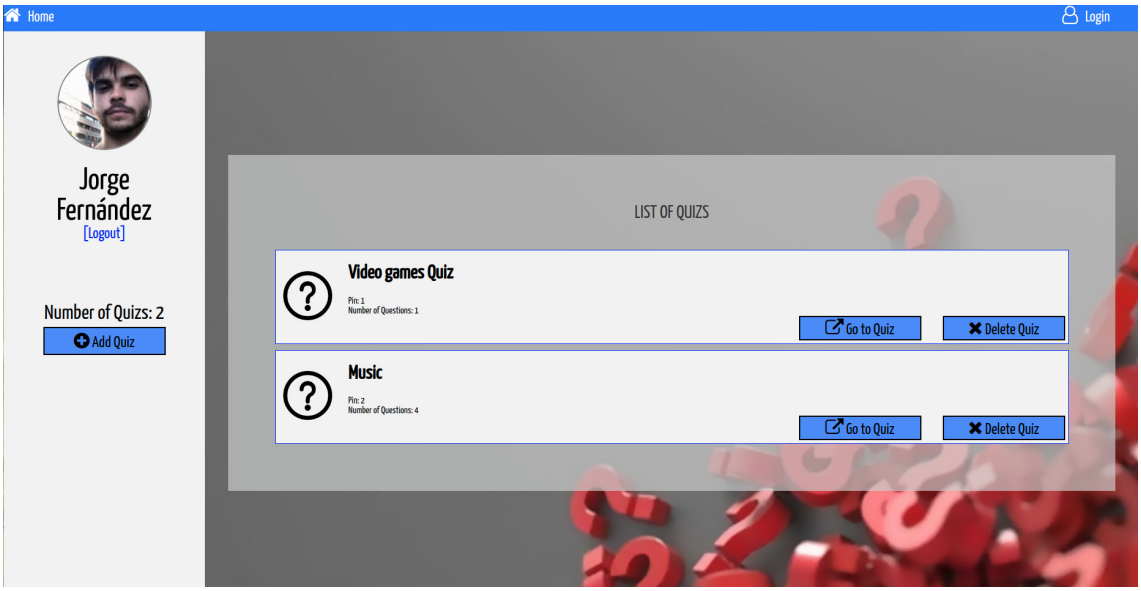

Figura 6.7: Interfaz mostrada al profesor cuando se autentica en el sistema

Listado de sesiones creadas La figura 6.7 nos muestra la interfaz que se le aparece al profesor cuando se autentica en el sistema. A la izquierda aparece claramente el botón para poder crear nuevas sesiones y en el resto de la pantalla, obviando el perfil del profesor, aparecen las sesiones que han sido creadas por él, dándole la opción de acceder a ellas como administrador o eliminarlas.

Figura 6.8: Interfaz del profesor administrando la sesión

Vista administrador En la figura 6.8 se puede ver la interfaz mostrada al profesor cuando realiza las funciones de administrador de la sesión. Se le proporcionan los mecanimos necesarios para retroceder o avanzar a la siguiente pregunta, reiniciar la sesión o ver el estado actual de la misma.

Además de estas funcionalidades básicas, el profesor dispone de una tabla de resultados en tiempo real, con las respuestas de los alumnos (utilizando un código de colores, verde si la respuesta es correcta o rojo si ha fallado) y la puntuación total que llevan. Debajo de este, tenemos un botón para poder descargarnos los resultados en formato .csv y poder consultarlos más adelante en caso de que fuera necesario.

6.2 Análisis de costes

Como se ha mencionado en secciones anteriores, la mayoría de los recursos utilizados, aunque disponen de unos días de prueba gratuita, son de pago, por lo que hay que considerar estos gastos a la hora de realizar el proyecto.

Según estudios recientes de mercado[2], el salario de un desarrollador *full-stack* es de una media de aproximádamente 75.000 dólares anuales. Haciendo el cambio a euros, y suponiendo un salario más bajo debido a que el desarrollo se hace en España y que no ha sido realizador por un graduado, se estima que el coste por hora es de 15 euros (en comparación a los casi 50 dólares por hora en EEUU).

En la tabla 6.1 se pueden ver los gastos económicos del proyecto desglosados, según los recursos utilizados. Destacar que, estos gastos son orientativos, ya que a la hora de calcular un presupuesto más profesional, hay que tener en cuenta otros factores, como podría ser el lugar de trabajo y los costes que este suponga.

Costes de Infraestructura	
Ordenador del alumno	1.200,00 €
Smartphone del alumno	199,00 €
Google App Engine	151,49 €
Datastorage	0,45 €
Compute Engine	1,46 €
Subtotal	1.552,40 €

Costes de Desarrollo	
Número de semanas	30
Número de horas por semana	35
Saldo medio del programador	15 €/h
Subtotal	15.750,00 €

Coste total	
Total	17.302,4 €

Tabla 6.1: Costes económicos del proyecto

[2] https://www.payscale.com/research/US/Job=Full_Stack_Developer/Salary

Capítulo 7

Conclusiones

En la realización de este proyecto, se han cumplido todos los objetivos que se proponían al realizar este sistema en el capítulo 2. En este capítulo, se discutiran en detalle cómo y por qué se han cumplido, además de posibles líneas de trabajo futuro.

7.1 Objetivos cumplidos

El objetivo principal del proyecto era el **desarrollo de una plataforma para la gamificación de sesiones magistrales**. Para cumplir estos objetivos, existen varios subobjetivos que hay que cumplir para poder realizar dicho sistema.

La **implementación básica de un sistema web** se ha llevado a cabo satisfactoriamente como se explica más detalladamente en el capítulo 5. Se requiere de un *front-end* y un *back-end* para el desarrollo completo de un sistema web, el *front-end* correspondiente a lo que ve el usuario, ha sido realizado utilizando *React*, un *framework* de *JavaScript*, al igual que *Node.js*, que en este caso ha sido utilizado para el *back-end*. Este sistema web debe ser fácilmente accesible, por lo que se optó por utilizar Google App Engine para poder desplegar el sistema en los servidores de Google y así se pudiera utilizar el sistema desde cualquier navegador incluso desde un *smartphone* para los alumnos. Se ha conseguido la autenticación mediante un servicio de terceros, con el uso de *OAuth2*.

La **mejora de la interacción entre el profesor y alumno** se ha realizado través del desarrollo de formularios amigables tanto para los alumnos como para el profesor, para que la adición de nuevas sesiones sea sencilla y los alumnos no tengan que centrarse en cómo responder a las preguntas, si no, que sea de manera fácil e intuitiva. El control de las sesiones reside completamente en el profesor, por lo que podrá aportar mayor dinamismo a las clases, no forzándose a terminar las preguntas rápidamente y profundizar en ellas si lo desea. Además, se le proporciona un *feedback* constate al profesor de la sesión.

El **tiempo variable entre interacciones** se consigue dando la opción al profesor de elegir qué hacer en cada momento. Para esto, se debe establecer canal de comunicación entre el dipositivo del profesor y los dispositivos de los alumnos que se hayan unido a la sesión que esté controlando el profesor en este momento. Dicho canal de comunicación se ha conseguido a través del uso de *sockets*. Implementando un servidor de *sockets*, el cual se ejecuta en

Google Compute Engine, se puede establecer la comunicación entre ámbos roles de forma que las respuestas dadas por los alumnos se envían de manera automática e inmediata al profesor.

La **generación de informes** implica que se puedan recoger las respuestas de los alumnos y su puntuación total, mostrar los resultados correctamente de los alumnos unidos a la sesión y poder obtener un fichero con estos datos para mantener un seguimiento de las sesiones realizadas. Para mostrar los datos correctamente en tiempo real, se ha implementado una tabla de resultados, que indica al profesor las respuestas de los alumnos, si estas son correctas o incorrectas y además, establece un *ranking*, ordenando de mayor a menor según la puntuación de los alumnos en tiempo real. La posibilidad de descargar un fichero ha sido implementada mediante un componente de *React*.

Todas las **bibliotecas, herramientas y componentes** externos utilizados para el desarrollo de GEA poseen una **licencia libre permisiva**, la mayoría de ellos extraídos de `https://www.npmjs.com/package/` todos con licencia **MIT**.

7.2 Trabajo futuro

Las líneas de trabajo futuro que puede seguir este proyecto son varias. Entre las distintas opciones podemos destacar:

- **Inclusión de nuevos proveedores para la autenticación**. Aunque la autenticación utilizando el proveedor de Google sea tal vez la opción más acertada por ser la más extendida, puede ser que algunos usuarios prefieran utilizar otro tipo de proveedores como podría ser *Facebook*.

  Además, se podría dar la opción a los profesores de autenticarse en el sistema con sus credenciales de la organización a la que perteneza, por ejemplo, utilizando el dominio de la UCLM.

  La implementación de esta funcionalidad no debería requerir mucho tiempo. Al igual que con Google, existe un componente en *React* que te permite la autenticación en el sistema utilizando el proveedor de Facebook, llamado `react-facebook-login`[1]. Esta funcionalidad se debería incluir como una nueva opción en el menú horizontal del sistema, en el mismo sitio que se encuentra la funcionalidad para autenticarse utilizando Google.

  En el listado 7.1 se puede ver un ejemplo de la implementación básica del componente anteriormente mencionado. Se debería adaptar al sistema para recoger únicamente los datos requeridos por el mismo, es decir, foto del usuario y nombre.

[1] `https://www.npmjs.com/package/react-facebook-login`

```
<FacebookLogin
appId="1088597931155576"
autoLoad={true}
fields="name,email,picture"
onClick={componentClicked}
callback={responseFacebook} />
```

Listado 7.1: Componente para la autenticación utilizando Facebook

- **Soporte para más tipos de multimedia**. El sistema da la opción al profesor de añadir imágenes en las preguntas de sus sesiones. Sería interesante dar la posibilidad de añadir también archivos de vídeo, aunque debido a su tamaño, que supera al de una imagen, la inclusión de estos vídeos podría hacerse con un reproductor de vídeos en red, utilizando los enlaces de estos vídeos a las plataformas correspondientes como pueden ser *Youtube* o *Vimeo*.

  Esta funcionalidad se podría extender también a archivos de tipo audio. Esto aportaría un mayor dinamismo a las sesiones y mayor variedad en cuanto a las sesiones. El código del sistema está preparado para añadir este nuevo tipo de funcionalidades, *React* nos ofrece esa versatilidad implementando nuevos componentes que se pueden incluir fácilmente en los ya existentes, o añadiendo opciones para mostrar unos u otros.

- **Nuevos tipos de preguntas**. En el capítulo 3 se ha visto como otros *Class Response System* implementaban otro tipo de sesiones, diferentes a las preguntas tipo test. Por ejemplo, se podría implementar una sesión donde el alumno tuviera que señalar distintas zonas, por ejemplo para geografía o biología.

  Esta funcionalidad se podría llevar más allá, como se ha dicho anteriormente bajo la supervisión de un docente, estudiando qué nuevas mecánicas serían más útiles para estas sesiones, además, la implementación de sesiones mixtas, que incluyan preguntas de distinto tipo sería también interesante.

  En lo que respecta a la implementación de esta futura funcionalidad, habría que crear un nuevo modelo para *datastore* para poder almacenar de manera correcta los nuevos tipos de sesiones. Por ejemplo, en el caso de señalar zonas en un mapa, se almacenarían las coordenadas x,y del canvas de la iamgen y el alumno dibujaría un círculo. Si la posición está dentro del círculo dibujado por el alumno, la respuesta se considera correcta.

- **Ampliación de datos del informe**. Al terminar cada sesión, el profesor puede descargar un fichero .csv con los datos de la sesión: nombres de los alumnos, respuestas dadas, puntuación final... Aunque es un informe completo bastante útil para el profesor, se podrían dar otros datos como la respuesta más acertada o fallada, el porcentaje que de alumnos que han conseguido superar el test satisfactoriamente, alumnos que

destaquen o se hayan quedado por detrás...

Esta funcionalidad no implicaría demasiado trabajo. En lo que respecta a la implementación no debería durar más de una semana, sin embargo, habría que estudiar la manera más apropiada para mostrar estos datos al profesor. Aunque un fichero de texto plano sería la solución más rápida, no sería un diseño amigable para el profesor.

- **Subscripción de pago** Al incluir esta funcionalidad en el sistema, se le permitiría al profesor, previo pago, personalizar en mayor medida sus sesiones. Por ejemplo, cambiar el formato de los botones, las fuentes o la manera en la que se muestran las preguntas, por ejemplo, colocando a la izquierda la imagen y a la derecha las posibles respuestas.

  En principio, se le darían al usuario distintas plantillas de las que poder partir para realizar sus sesiones. Esta sería la funcionalidad que más tiempo requeriría, se requeriría de un gran esfuerzo de diseño de las distintas plantillas, además de implementar los archivos .css correspondientes y ver qué más opciones podría personalizar el profesor.

  En este caso, se limitarían varias funciones del sistema como delimitar el número de sesiones que puede tener almacenadas, aunque no afectaría a funcionalidades básicas del mismo.

ANEXOS

Contenido del CD

En este anexo se detalla el contenido del disco compacto que acompaña a este documento

- **SRC**: Contiene todo el código del proyecto con la estructura explicada en el capítulo 5.
- **Doc**: este directorio contiene el PDF generado de este proyecto y los fuentes LaTeXutilizados, donde se incluyen también las imágenes.
- **Prototipos**: incluye imágenes de algunos prototipos y diseños descartados de las primeras fases de desarrollo.

Conclusión personal

La realización de este proyecto supone el fin de la etapa universitaria, al menos por el momento. A lo largo del tiempo que he pasado en la universidad, he adquirido los conocimientos necesarios para saber realizar tareas nuevas, con tecnologías y/o lenguajes que nunca he utilizado, lo cual resulta extremadamente útil, casi diría que necesario, en el ámbito laboral. Termino esta etapa con año y medio de experiencia profesional gracias a las becas de colaboración y al equipo de Furious Koalas.

El hecho de haber adquirido estos conocmientos ha sido crucial a la hora de enfrentarme a este proyecto, puesto que eran todo tecnologías nuevas que nunca había usado y apenas tenía una ligera noción de lo que implicaba, pero no lo veo como algo negativo, más bien positivo, es dificil encontrarte en una situación en la que estés trabajando en lo que te gusta y contento en tu lugar de trabajo.

La carrera como tal, me ha enseñado a no subestimar las tareas a realizar, aunque de primera mano parezca algo sencillo, tienes que ser consciente de que pueden surgir dificultades inesperadas, fallos de versiones, bibliotecas que se actualizan a un día de la entrega y deja de funcionar todo el proyecto, etc.

Recalcar la importancia del continuo estudio en la informática, actualmente es un campo que se encuentra en todos sitios, constantemente actualizándose y para ser verdaderamente competente, debes estar al día. Esto también ofrece sus ventajas, al estar presente en todos los campos de la ciencia, es fácil que encuentres algo que verdaderamente te guste.

También, me gustaría hablar de las asignaturas impartidas a lo largo de la carrera. No soy quién para cuestionar qué se debe dar en las clases o qué no, pero sí es cierto que he notado un descontento general con la cantidad de asignaturas que existen obligatorias de una misma rama. Hay muchos aspectos básicos requeridos en el ámbito laboral que apenas se mencionan en la carrera y varias asignaturas optativas que son muy demandadas en el mundo actual, como por ejemplo *Machine Learning* o gráficos por computador.

Como conclusión, aunque haya ciertos momentos en los que no puedas más y quieras dejar la carrera, tienes que ser consciente que es una oportunidad demasiado buena como para dejarla pasar debido a todas las posibles salidas laborales que la carrera de informática ofrece. Con esfuerzo y constancia, todo es posible.

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual

or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and

the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both

covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to

the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.
5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

5. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder expli-

citly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

10. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of

another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Referencias

[Arn14] Brian J Arnold. Gamification in education. *ASBBS Proceedings*, 21(1):32, 2014.

[Ber06] Gerald Bergtrom. Clicker sets as learning objects. *Interdisciplinary Journal of E-Learning and Learning Objects*, 2(1):105–110, 2006.

[BL01] Ray A Burnstein and Leon M Lederman. Using wireless keypads in lecture classes. *The Physics Teacher*, 39(1):8–11, 2001.

[BL03] Ray A Burnstein and Leon M Lederman. Comparison of different commercial wireless keypad systems. *The Physics Teacher*, 41(5):272–275, 2003.

[Bru09] Derek Bruff. *Teaching with classroom response systems: Creating active learning environments*. John Wiley & Sons, 2009.

[Cal07] Jane E Caldwell. Clickers in the large classroom: Current research and best-practice tips. *CBE-Life sciences education*, 6(1):9–20, 2007.

[Cho13] Yu-Kai Chou. Octalysis: Complete gamification framework. *Yu-Kai Chou & Gamification*, 2013.

[DGL+96] Robert J Dufresne, William J Gerace, William J Leonard, Jose P Mestre, and Laura Wenk. Classtalk: A classroom communication system for active learning. *Journal of computing in higher education*, 7(2):3–47, 1996.

[DKND11] Sebastian Deterding, Rilla Khaled, Lennart E Nacke, and Dan Dixon. Gamification: Toward a definition. In *CHI 2011 gamification workshop proceedings*, volume 12. Vancouver BC, Canada, 2011.

[FM06] Carmen Fies and Jill Marshall. Classroom response systems: A review of the literature. *Journal of Science Education and Technology*, 15(1):101–109, 2006.

[JBR00] Ivar Jacobson, Grady Booch, and James Rumbaugh. *El proceso unificado de desarrollo de software/The unified software development process.* Number 004.41. Pearson Educación,, 2000.

[JS02] Eugene Judson and Daiyo Sawada. Learning from past and present: Electronic response systems in college lecture halls. *Journal of Computers in Mathematics and Science Teaching*, 21(2):167–181, 2002.

[KL09a] Robin H Kay and Ann LeSage. Examining the benefits and challenges of using audience response systems: A review of the literature. *Computers & Education*, 53(3):819–827, 2009.

[KL09b] Robin H. Kay and Ann LeSage. Examining the benefits and challenges of using audience response systems: A review of the literature. *Computers Education*, 53(3):819 – 827, 2009. ISSN 0360-1315. URL `http://www.sciencedirect.com/science/article/pii/S0360131509001134`.

[LH11] Joey J Lee and Jessica Hammer. Gamification in education: What, how, why bother? *Academic exchange quarterly*, 15(2):146, 2011.

[Maa13] Jitendra Maan. Social business transformation through gamification. *arXiv preprint arXiv:1309.7063*, 2013.

[MAG03] Redonda G Miller, Bimal H Ashar, and Kelly J Getz. Evaluation of an audience response system for the continuing education of health professionals. *Journal of Continuing Education in the Health Professions*, 23(2):109–115, 2003.

[Mas17] V Mashevskiy. Front-end web development. 2017.

[Ném15] Tamás Németh. Gamification in education. 2015.

[Rau13] Marta Rauch. Best practices for using enterprise gamification to engage employees and customers. In *International Conference on Human-Computer Interaction*, pages 276–283. Springer, 2013.

[RBL+05] Neville W Reay, Lei Bao, Pengfei Li, Rasil Warnakulasooriya, and Gordon Baugh. Toward the effective use of voting machines in physics lectures. *American Journal of Physics*, 73(6):554–558, 2005.

[Sat18] Brandon Satrom. Choosing the Right JavaScript Framework for Your Next Web Application. Technical report, 2018.

[SBD04] Susan AJ Stuart, Margaret I Brown, and Stephen W Draper. Using an electronic voting system in logic lectures: one practitioner's application. *Journal of Computer Assisted Learning*, 20(2):95–102, 2004.

[VMK+70] Mikhail Vasilievich Vinichenko, Alexander Vasilievich Melnichuk, Andrei Vladimirovich Kirillov, Sergey Anatolyevich Makushkin, and Yulyia Alexandrovna

Melnichuk. Modern views on the gamification of business. *The Journal of Internet Banking and Commerce*, 1970.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfg** (versión 0.20190205) que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`