



Aplicación web en tiempo real con
websockets destinada a la educación

JUAN ANTONIO BUENO CRUCERAS

Tutor
Isaac Lera Castro

Palma, 6 de septiembre de 2016

Índice

Capítulo 1. Introducción.....	1
1.1 Contexto	1
1.2 Motivación	1
1.3 Tareas	2
1.4 Metodología	4
1.5 Planificación inicial	5
Capítulo 2. Estado del arte	8
2.1 Aplicaciones similares	8
2.1.1 Socrative	8
2.1.2 Moodle	9
2.1.3 Atriviate	9
2.1.4 Autoescuela test DGT	10
2.2 Nombre de la aplicación.....	11
Capítulo 3. Tecnología utilizada	12
3.1 NodeJS.....	12
3.2 MongoDB.....	12
3.3 ExpressJS	13
3.4 Nodemon.....	13
3.5 NPM.....	14
3.6 Socket.IO	14
3.7 Jade	15
3.8 AngularJS.....	15
3.9 CSS.....	16
Capítulo 4. Arquitectura	18
4.1 Configuración del servidor	18
4.2 Patrón MVC	19
4.2.1 Modelo	19
4.2.2 Vista.....	20
4.2.3 Controlador	20
4.3 Base de datos	21
4.4 Comunicación entre sockets	22
4.5 Cookies y sesiones.....	23
Capítulo 5. Pruebas.....	26
Capítulo 6. Futuro trabajo.....	28
6.1 Ley de protección de datos	28
6.2 Panel de control y privilegios	28

6.3	Control del almacenamiento de datos	29
6.4	Nuevos tipos de partidas.....	29
6.5	Implementación de aplicación para móviles.....	29
6.6	Ajustar el diseño para múltiples navegadores	30
Capítulo 7. Conclusión y valoración personal		32
Anexo A. Manual de instalación.....		34
A.1	MongoDB	34
A.2	Node y NPM	34
A.3	Módulos de NodeJS (ExpressJS, socket-io...)	35
A.4	Nodemon	35
A.5	Iniciar servidor	36
A.6	Selenium.....	36
Anexo B. Manual de usuario		38
Bibliografía		50

Capítulo 1

INTRODUCCIÓN

En este capítulo introductorio se contextualizará al lector y se mostrará una visión general del proyecto, como pueden ser los objetivos o los beneficios que aporta al usuario.

Los últimos puntos de este capítulo tratarán sobre la planificación establecida para realizar este proyecto, como son las tareas a realizar, la metodología que se sigue o un cronograma.

1.1 Contexto

En la actualidad, las TIC se encuentra en prácticamente todo nuestro entorno, desde el trabajo hasta en nuestra vida cotidiana. Esto se debe a que la tecnología ha ido evolucionando a grandes pasos en los últimos años, teniendo cada vez más importancia en diversos ámbitos. Esta evolución es debida a la capacidad de conexión de las TIC con Internet. [7, 8]

La evolución de las TIC ha permitido implementar nuevas técnicas y metodologías para el ámbito de la enseñanza. Es en este campo en el que se encuentra el proyecto realizado.

El uso de la tecnología dentro de la educación y de la enseñanza mejora la eficiencia y productividad en las aulas, y aumenta el interés de los alumnos en tareas académicas gracias al uso de herramientas más interactivas.

Introducir la tecnología en la educación aporta varios beneficios, tanto a profesores como a alumnos. Para los profesores, es una mejora en su rendimiento, ya que pueden optimizar tareas rutinarias (realizar correcciones, compartir apuntes...), sin necesidad de volver a realizar a misma tarea una y otra vez. [7, 8]

La tecnología puede ayudar al alumno en varios aspectos. Uno de ellos es el de poder adaptarse al ritmo de aprendizaje de los alumnos. Esto quiere decir que los estudiantes más adelantados puedan disponer de contenidos adicionales, mientras que los que necesiten mayor esfuerzo puedan obtener material de apoyo. Otra ventaja es que aumenta la retención en el aprendizaje mediante simulaciones en escenarios reales o juegos, con el fin de que completen un mismo concepto para que puedan asimilar más fácilmente la información. Esto se aplicaría en juegos en el que han de realizar una tarea aplicando los conocimientos que se han adquirido en clase, por ejemplo. [7, 8, 14, 17, 18]

En el siguiente apartado se describe la motivación para llevar a cabo este proyecto.

1.2 Motivación

La motivación de realizar este proyecto es la de crear una aplicación que permita evaluar a los alumnos, además de fomentar el aprendizaje. Utiliza un sistema de partidas formado por una serie de preguntas, y que deben de ser contestadas con una respuesta corta por los participantes que se hayan unido.

Para aumentar el interés en el aprendizaje de los estudiantes, las preguntas tendrán un único

ganador, que será aquel alumno que responda correctamente la pregunta en el menor tiempo posible. Al finalizar la partida, se mostrará un ranking con las victorias de cada alumno. Añadir esta forma de competición es una manera de motivar a los alumnos a estudiar y llevar las lecciones al día. Otra forma de incentivar el aprendizaje es la de permitir que los alumnos puedan crear sus propias partidas, y sus compañeros puedan unirse a ellas. Por lo tanto, tanto alumnos como profesores pueden crear partidas.

Estas partidas le sirven al profesor para comprobar si está impartiendo correctamente el temario en las clases, ya que si en una partida hay muchas preguntas sin ningún alumno ganador, puede significar que debería cambiar la metodología o el ritmo de enseñanza.

Para poder llevar a cabo estos objetivos, es necesario que la aplicación sea capaz de responder en tiempo real a todas las peticiones de los usuarios de forma eficiente. Hay que tener en cuenta que pueden haber cientos de usuarios en cada partida, y cientos de partidas iniciadas. Cada respuesta de cada usuario genera una solicitud, por lo que es importante utilizar una tecnología capaz de gestionar toda esta actividad, sin disminuir el rendimiento.

Como la aplicación ha de ser en tiempo real y poder comunicarse entre ordenadores en distintos, es necesario utilizar alguna tecnología que permita esta condición. Para ello se utilizan los sockets [20], que permiten intercambiar datos de forma fiable y ordenada. Por lo tanto, hay que investigar alguna tecnología que implemente sockets y se puedan mostrar los cambios en tiempo real, sin necesidad de refrescar la página.

Otro aspecto importante es el diseño de una interfaz sencilla y fácil de aprender a utilizarla, ya que la aplicación puede ser utilizada por cualquier estudiante de cualquier edad.

El siguiente apartado muestra las tareas que se han establecido para realizar el proyecto.

1.3 Tareas

En este apartado se muestra un listado de las tareas que se han realizado para llevar a cabo la aplicación. Al definir diferentes tareas, ha sido más sencillo llevar a cabo el proyecto al tener objetivos más concretos. A continuación se muestran las tareas realizadas, junto a un código identificativo de cada una, que será utilizado en este documento para referenciar una tarea concreta. Este código está formado por una letra y dos dígitos. La letra indica en qué fase del desarrollo se realiza la tarea (en el apartado *1.4 Metodología* se explican estas fases). Puede ser A (Análisis), D (Diseño), I (Implementación), P (Pruebas) o E (Entrega). Los números sirven para diferenciar las diferentes tareas de cada fase, por lo que pueden ir desde el valor 01 hasta el 99. El formato será algo así: X00.

[A01] Requerimientos de la aplicación

El primer paso es el de definir qué aplicación se quiere conseguir. Para ello, se necesita saber que requerimientos ha de cumplir, mediante reuniones con los interesados, en este caso, con el tutor.

[A02] Definir el cronograma

Se ha definido un cronograma inicial en el que se marcan las tareas que se han de realizar, las dependencias y sus duraciones. Al acabar el proyecto, se compara con el cronograma real para observar cuánto ha variado el desarrollo del proyecto con lo planificado. En el apartado *1.5 Planificación inicial* se muestra más detallado.

[A03] Buscar aplicaciones similares

Una vez definido el tipo de proyecto y los objetivos a cumplir, se buscaron aplicaciones parecidas. Esta tarea ha servido para tener una idea de cómo enfocar el proyecto.

También ha ayudado a elegir qué plataforma es la más apropiada para la aplicación. Al ser

una aplicación que necesita la interacción con usuarios en diferentes ordenadores, se ha optado por una aplicación web, aunque también podría implementarse para móviles.

En el *Capítulo 2 – Estado del arte* se muestran algunas aplicaciones que han ayudado a definir nuestra aplicación.

[D01] Tecnología a utilizar

Este proyecto se ha realizado con NodeJS en el lado del servidor, y AngularJS para gestionar cambios y funciones en el lado del cliente. En el *Capítulo 3 - Tecnología utilizada* se explicarán ambas tecnologías.

En cuanto a la tarea, se ha decidido usar Node por el motivo de que se necesita gestionar un número considerable de conexiones concurrentes. Pueden haber cientos de jugadores en una sola partida, y haber muchas partidas iniciadas. Para solventar esta condición, Node ofrece las funcionalidades adecuadas, ya que es capaz de gestionar cientos de miles de conexiones a la vez sin tener que disminuir el rendimiento. Esto se debe a que crea un único hilo en vez de crear un hilo por cada conexión.

El motivo de la elección de Angular ha sido porque la aplicación ha de ser en tiempo real. Angular permite mostrar los cambios en las vistas del lado del cliente sin necesidad de actualizar la página cada vez ni de crear una función que haga cada cambio (mediante modelos y controladores, que se explicarán en el *Capítulo 3 – Tecnología utilizada*). Esta característica permite ahorrarnos tiempo de programación, y conseguir como resultado una ejecución en tiempo real en el lado del cliente.

[D02] Definir cómo guardar los datos

Una vez decidida la aplicación, se había de elegir la forma en que se almacenaría la información. En principio, se pensó almacenar la información en un fichero en formato JSON, pero al utilizar Node y Angular, se decidió implementar la pila MEAN [5]. Para implementarla, se optó por la base de datos MongoDB (la *M* de MEAN), una base de datos NoSQL que almacena los datos sin tablas ni relaciones. Además, los datos están en formato JSON, facilitando la integración con Node y Angular.

[D03] Realizar comunicación entre usuarios

Para realizar la comunicación entre los diferentes usuarios es necesario utilizar una tecnología que permita el envío de datos entre clientes y servidor en tiempo real. Para ello se han utilizado sockets (Socket.io de NodeJS, que se explicará que son en el *Capítulo 3 - Tecnología utilizada* y cómo funcionan, en el *Capítulo 4 - Arquitectura*), que permiten enviar mensajes entre los distintos usuarios.

[D04] Elegir arquitectura

La arquitectura permite estructurar el código del proyecto, y comunicar los diferentes módulos que lo componen. En la aplicación se conectarán usuarios a través de un navegador web, por lo que se ha definido la arquitectura tanto para el lado del cliente, como para el lado del servidor basada en el patrón MVC. En el *Capítulo 4- Arquitectura* se explica esta arquitectura.

[D05] Diseñar la estructura de las partidas

Las partidas son la parte principal de esta aplicación. En ellas es donde los usuarios crean y contestan preguntas. En esta tarea se ha especificado como serán las partidas. En principio iba a haber una única pregunta por partida, pero se optó por crear partidas con varias preguntas de respuesta corta.

Para aprovechar las ventajas de los sockets, se añadió la funcionalidad de poder crear nuevas preguntas una vez que la partida ha iniciado, haciendo más dinámica la interacción entre los usuarios.

También se añadió la funcionalidad de autocorregir las preguntas una vez finalizada la partida. Se puede ver en más detalle en el *Anexo B. Manual de usuario*.

[I01] Desarrollo de la parte del servidor

La parte del servidor permite gestionar las diferentes peticiones de los usuarios, además de poder almacenar, modificar o crear nuevos registros, por lo que es necesario que haya una interacción entre el servidor y la base de datos. También es necesario implementar un enrutador o encaminador encargado de redirigir las conexiones.

[I02] Desarrollo de la parte del cliente

La parte del cliente es la encargada de proporcionar al usuario las funcionalidades necesarias para que pueda interactuar con la aplicación y comunicarse con el servidor. Deberá mantener en tiempo real las páginas en las que el usuario acceda.

[I03] Instalar en el servidor

Una vez haya finalizado el proyecto y comprobado que todo funciona correctamente en el servidor local, se ha instalado en un servidor de la universidad, con el fin de conseguir que puedan conectarse diferentes usuarios desde cualquier ordenador. En el *Anexo A. Manual de instalación* se explica cómo se realiza dicha instalación.

[P01] Pruebas de integración

Una vez acabados todas las partes que forman la aplicación, es necesario comprobar el correcto funcionamiento conjunto. Es necesario comprobar que las partidas se crean correctamente, finalizan en el tiempo establecido, permite unirse a varias partidas, etc.

[P02] Hacer pruebas de carga

Consiste en hacer un *benchmark* para comprobar el rendimiento que la aplicación en un entorno real. Se explica en el *Capítulo 5 - Pruebas*.

[E01] Documentar el proyecto

Al finalizar el proyecto, se ha creado una memoria en la que se explica en qué consiste la aplicación, aspectos técnicos o posibles cambios que se pueden hacer en el futuro.

[E02] Realizar presentación

Es la tarea final, que consiste en realizar un conjunto de diapositivas para presentación el proyecto frente a un tribunal. Ha de contener los puntos más importantes del proyecto.

1.4 Metodología

La metodología utilizada en el proyecto se basa en el **desarrollo en cascada** [15]. Se ha optado por este enfoque porque permite secuenciar las fases del proceso de desarrollo del proyecto, además de poder volver a una fase anterior en caso de tener que hacer alguna modificación. La *Figura 1* muestra este esquema.

En las 5 fases que forman este tipo de desarrollo se han realizado las siguientes actividades.

- **Análisis:** En reuniones entre el alumno y el tutor se ha descrito el tipo de aplicación que se ha de desarrollar. Para ello, se han presentado qué objetivos debe cumplir y qué las funcionalidades debe tener.
- **Diseño:** En esta fase se ha definido la arquitectura de la aplicación, y la tecnología y herramientas que se usarán. Organiza las diferentes funcionalidades en módulos que permitan ser elaborados individualmente.
- **Implementación:** Consiste en implementar la aplicación acorde con las fases anteriores. Es la parte de codificar los diferentes módulos.
- **Pruebas:** Se realizan diversas pruebas de los elementos ya finalizadas, con el fin de comprobar que cumple con los requisitos establecidos y no tenga errores. Las pruebas de los

diferentes módulos se harán de forma paralela con la fase de implementación, con el fin de asegurar el correcto desarrollo. Al final de la implementación, se realizarán pruebas conjuntas de toda la aplicación para encontrar posibles fallos que hayan podido aparecer al integrar todos los módulos.

- **Entrega:** Es la fase final. Cuando la aplicación consigue pasar todas las pruebas, se realiza su documentación y se prepara la presentación.

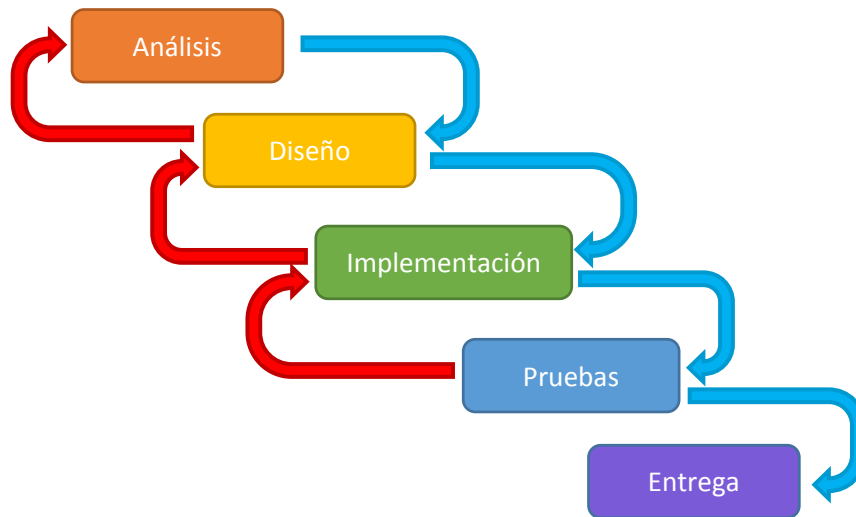


Figura 1. Esquema de desarrollo en cascada

1.5 Planificación inicial

El siguiente cronograma muestra la planificación de las tareas que se han de realizar para desarrollar este proyecto. Utiliza el esquema de desarrollo en cascada del apartado anterior.

La fecha de inicio es día 1 de junio de 2016, y se calcula que finaliza el día 11 de agosto de 2016. La *Figura 2* muestra una línea de tiempo con las diferentes fases del desarrollo del proyecto, sin entrar en detalle en la duración de las diferentes tareas. La fase que más dedicación necesita es la de implementación, ya que contiene tareas relacionadas con la codificación de la aplicación.



Figura 2. Cronograma de la planificación inicial

Una vez visto el cronograma de las diferentes fases, a continuación se muestra un diagrama de Grantt en la *Figura 3* y en la *Figura 4*, el cual contiene las dependencias entre las diferentes tareas y su duración. Las dos tareas que más tiempo consumirán son las de desarrollar la parte del cliente [I02] y la del servidor [I01], ambas en la fase de implementación. En estas dos tareas, a parte de la codificación de los módulos de la aplicación, se han de ir realizando pruebas para comprobar el correcto funcionamiento.

La mayoría de tareas se realizan de forma secuencial, aunque hay algunas, como la tarea de definir la forma de guardar los datos [D02] y la de realizar la comunicación entre usuarios [D03] que se realizan en forma paralela.

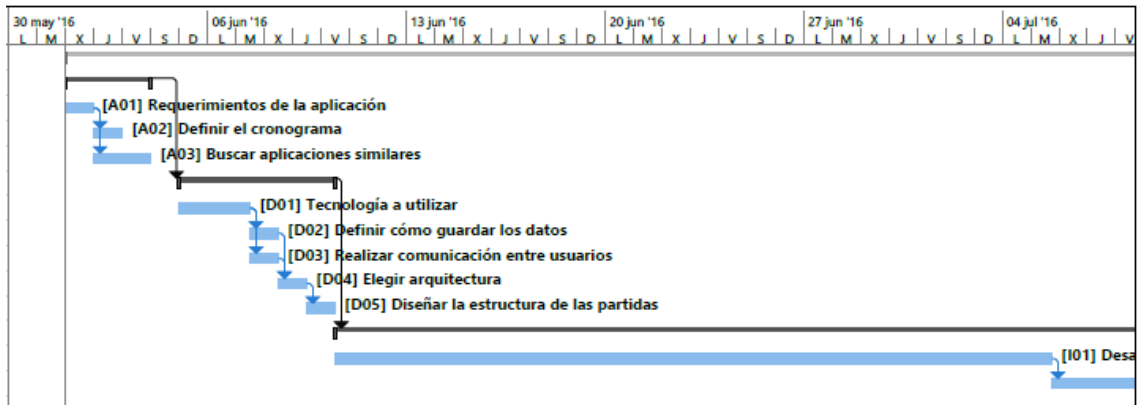


Figura 3. Diagrama de Grantt (Parte I)

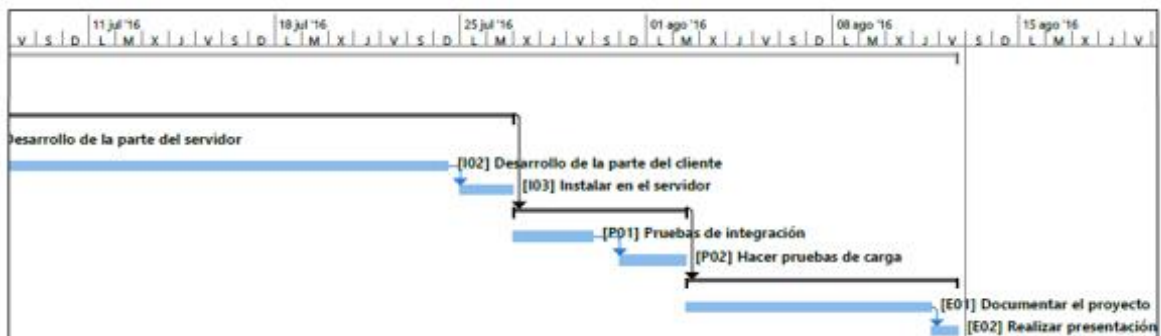


Figura 4. Diagrama de Grantt (Parte II)

Capítulo 2

ESTADO DEL ARTE

Para facilitar el desarrollo del proyecto, se han buscado aplicaciones con objetivos similares a la nuestra, con el fin de poder aplicar ideas que puedan resultar interesantes en la aplicación.

2.1 Aplicaciones similares

En este apartado se mostrarán algunas aplicaciones que tengan una funcionalidad u objetivos similares a los de nuestro proyecto.

2.1.1 Socrative

Socrative [19] es una aplicación en tiempo real que permite a los profesores evaluar a sus alumnos mediante diferentes tipos de pruebas, entre las cuales se incluyen exámenes, encuestas, test o incluso juegos didácticos, como muestra la *Figura 5*. Está disponible para móviles y para web. Consta de dos versiones, una gratuita y otra de pago que permite tener funcionalidades extras, como por ejemplo poder asignar más alumnos al curso. Su uso está destinado para escuelas, institutos y universidades, aunque también puede ser usado para otras organizaciones como corporaciones.

Estas pruebas son creadas por el profesor, y puede ver en tiempo real las respuestas de los alumnos. Cuando una prueba finaliza, se crean informes con los resultados de los alumnos. Se pueden mostrar tanto en gráficas, como ser descargados en varios formatos. También permite definir los resultados de una serie de alumno en concreto o de toda la clase.

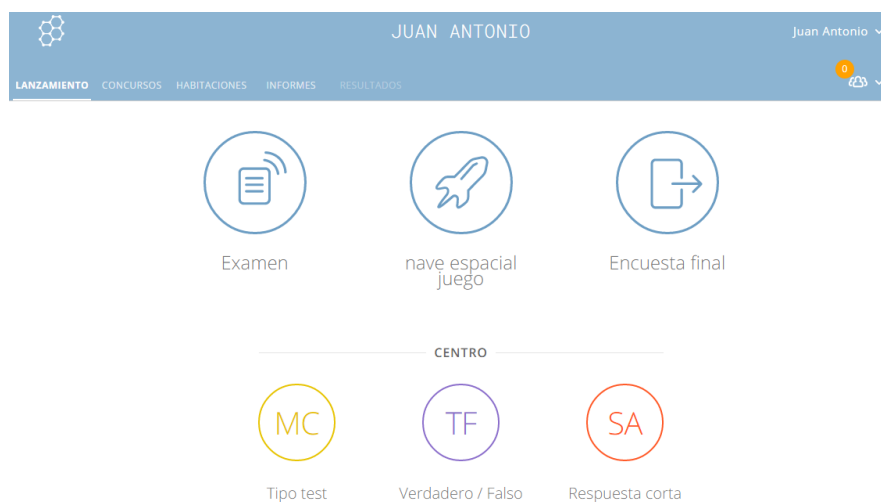


Figura 5. Vista principal de Socrative (profesor)

2.1.2 Moodle

Moodle [16] es una aplicación web utilizada en el ámbito de la educación, destinada a organizar y gestionar cursos. La *Figura 6* muestra la vista principal de un curso en *Moodle*. La idea de Moodle consiste en que el profesor cree un ambiente que fomente el aprendizaje de los estudiantes a través de la publicación de información propia del profesor o que considere oportunos.

Además de gestionar los cursos, permite la conversación entre alumnos y profesores o evaluar a los alumnos mediante preguntas realizadas por el docente, similar a un test.

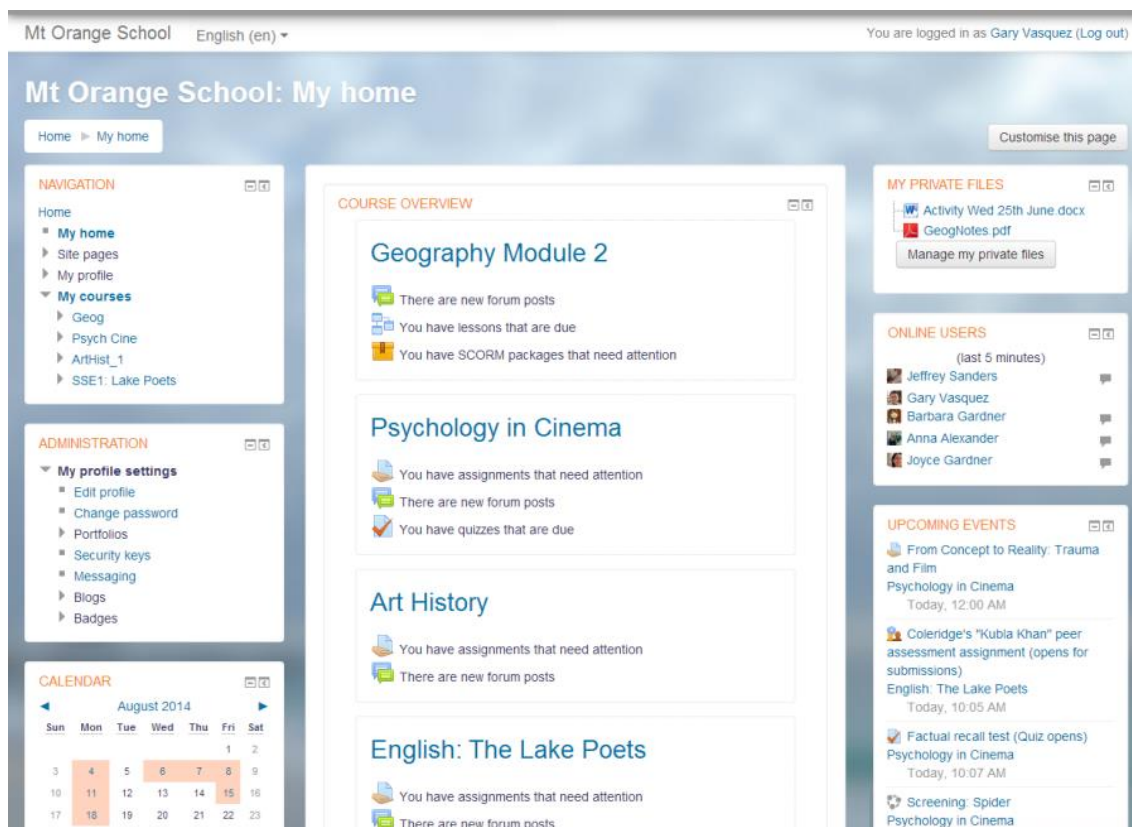


Figura 6. Vista de un curso en Moodle

2.1.3 Atriviate

Atriviate [22] es una aplicación para teléfonos móviles, que implementa varios modos de juego basados en el juego Trivial [21]. Se muestra en la *Figura 7*. Entre los modos de juego que dispone, destacan el estilo trivial, hacer un tres en raya contestando preguntas del rival, o duelos entre usuarios para ver quien contesta más preguntas correctas. Aunque esta aplicación no está enfocada para el ámbito de la enseñanza, el sistema de preguntas es muy similar al que se quiere implementar en el proyecto, por lo que es una aplicación que puede ser útil cómo modelo. El modo duelo es el que más se asemeja a lo que queremos implementar en el proyecto, por lo que se utilizará como referencia.

El modo duelo permite crear partidas con preguntas predefinidas que ningún usuario sabe con antelación. Se puede realizar entre amigos, o bien, asignando usuarios al azar. Se dispone de un tiempo limitado para contestar las 20 preguntas que forman la partida. El ganador del duelo será aquel jugador que consiga acertar el mayor número de preguntas en el menor tiempo posible.



Figura 7. Vista del juego Atrivate

2.1.4 Autoescuela test DGT

Esta aplicación online es utilizada en autoescuelas españolas, con el fin de realizar test sobre los temas que se han impartido. El alumno puede elegir los diferentes temas que aparecen en la aplicación. Consiste en una serie de preguntas relacionadas con la educación vial, en el que se muestran 3 posibles opciones de cada pregunta. Las preguntas suelen ir acompañadas de una imagen en la que se muestran señales de tráfico, maniobras o posibles situaciones en las que se puede encontrar en la vida real, y el alumno ha de responder qué haría o qué significa dicha imagen.

Al final de cada test, se muestran las respuestas acertadas e incorrectas, junto a una pequeña explicación sobre cada respuesta. El alumno puede ver cada resultado de cada pregunta navegando a través del paginado que aparece debajo. La *Figura 8* muestra una pregunta de un test realizado por dicha aplicación.

Esta aplicación es útil para comprobar si el alumno está preparado para examinarse de la parte teórica del carnet que se quiera examinar, sustituyendo a las pruebas tipo test sobre papel. La ventaja de esta aplicación es el ahorro en papel, la rapidez en tiempo de corrección o la posibilidad de tener actualizadas las preguntas.

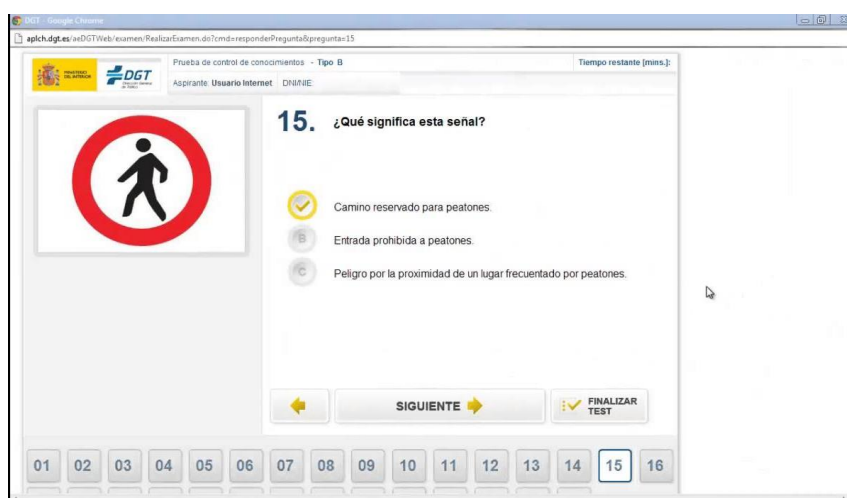


Figura 8. Autoescuela test DGT

2.2 Nombre de la aplicación

El nombre de la aplicación es **Búho**. Su nombre es debido a que se quería un nombre corto y fácil de recordar. Este nombre hace referencia al animal, ya que tradicionalmente se le ha considerado como símbolo de la sabiduría y se ha relacionado con la enseñanza, teniendo un papel de profesor en varias fábulas.

Teniendo en cuenta estos detalles, parecía un nombre apropiado para la aplicación, ya que estará enfocada para el sector de la educación. La *Figura 9* muestra el logo de la aplicación, un búho con gafas sujetando un libro.



Figura 9. Logo de la aplicación

Capítulo 3

TECNOLOGÍA UTILIZADA

Para el desarrollo de esta aplicación, todas las herramientas utilizadas son *open source*. Se ha usado el *stack MEAN* [5], un conjunto de subsistemas basados en JavaScript y de código abierto. A continuación se explicarán detalladamente sus componentes, que son MongoDB, ExpressJS, AngularJS y NodeJS; además de otras tecnologías y herramientas que se han utilizado en el proyecto.

3.1 NodeJS

Node es un intérprete de JavaScript que se ejecuta en el lado del servidor. Está *orientado a eventos* [9], es decir, que su estructura de ejecución se define por los sucesos que ocurren en el sistema o provocados por el usuario.

Un aspecto importante de Node es la capacidad de soportar cientos de miles de usuarios de forma concurrente sin causar grandes costos asociados al cambio de contexto [10]. Esto se consigue con un funcionamiento de evaluación de un único hilo de ejecución (en vez de crear un hilo por cada conexión, como pasa en PHP o Java) [11].

Las operaciones que realicen peticiones de E/S han de tener una función de *callback* encargada de obtener los resultados de la solicitud.

3.2 MongoDB

Es el sistema gestor de bases de datos. MongoDB es una base de datos NoSQL. Esto quiere decir que no hace uso de tablas ni relaciones. Al no necesitar de una estructura predefinida (tabla) como en una base de datos relacional, podemos insertar los campos a nuestra conveniencia.

Se estructura en documentos y colecciones. Los *documentos* son la unidad en que se guardan los datos, en este caso, en JSON. También se les conoce como *objetos*. Las *colecciones* permiten guardar los documentos, los cuales no necesariamente han de tener la misma estructura o los mismos campos. Para entender mejor estos conceptos, en la *Figura 10* se muestra una colección que contiene 3 documentos. De forma análoga a una base de datos relacional, los documentos serían los registros almacenados, mientras que las colecciones serían algo parecido a las tablas, aunque hay que tener claro que son cosas distintas.



Figura 10. Colección con varios documentos

Los datos son almacenados en formato BSON, una representación binaria de JSON, facilitando a Node el manejo de éstos. Esta ventaja es debida a que Node está basado en JavaScript, siendo JSON su formato de texto para operar con los datos. Esto permite unificar el entorno de desarrollo JavaScript, evitando tener que hacer conversiones de tipos.

El módulo `mongoose` de Node es el encargado de realizar la conexión entre Node y la base de datos MongoDB, además de definir las estructuras y operaciones de los *modelos* (se explicarán en el *Capítulo 4 – Arquitectura*).

3.3 ExpressJS

Se utiliza la versión 4. ExpressJS es un framework de desarrollo de aplicaciones web para Node.js [6]. Simplifica algunas funcionalidades como el ruteo de URL (GET, POST, PUT...) o poder elegir entre varios lenguajes para plantillas (EJS, Jade, Dust.js...).

Otra característica es la de poder guardar sesiones, seleccionar patrones para las rutas o guardar cookies. Si no se usase Express, se debería de implementar el código que gestionase estas tareas.

3.4 Nodemon

Esta herramienta se utiliza durante el desarrollo de la aplicación. Su función es monitorear cambios en el código del servidor, y reiniciándolo de forma automática. De esta forma ahorramos tiempo a la hora de desarrollar la aplicación, ya que no hemos de reiniciar el servidor manualmente cada vez que haya algún cambio en algún fichero del servidor.

Tiene varias características, como son la de detectar los ficheros con una extensión predefinida, ignorar ficheros o directorios a monitorear o solo utilizar un directorio específico.

Su uso es muy parecido a ejecutar el comando `node`, por lo que es muy sencillo de utilizar. Para ejecutarlo, es necesario que sea desde el directorio raíz, ya que esta herramienta busca los cambios en ficheros que estén por debajo del directorio en el que se ejecuta. Para iniciar el

servidor con nodemon, basta escribir en el directorio raíz del proyecto:

```
nodemon <nombre_aplicacion>
```

3.5 NPM

Es el gestor de paquetes utilizado por NodeJS (del inglés, *Node Package Manager*). Es ejecutado por línea de comandos mediante el comando `npm`. Algunas acciones que permite son iniciar el servidor o instalar paquetes para NodeJS. Estos paquetes se pueden guardar en el fichero *package.json*, que facilita instalaciones futuras de los paquetes necesarios para ejecutar el servidor, ya que contiene información sobre la versión de cada paquete o sus dependencias.

3.6 Socket.IO

Es un módulo que implementa los websockets [11], permitiendo realizar las comunicaciones de forma bidireccional y en tiempo real entre los usuarios.

Se ejecuta en el servidor y en el cliente, por lo que se han de implementar las diferentes funcionalidades en ambos lados. Desde el cliente se pueden hacer peticiones al servidor o recibir respuestas del servidor. Para enviar datos a otro cliente, primero ha de enviar la información al servidor, y el servidor se encargará de reenviar los datos al usuario (o usuarios) destinatarios.

Para utilizar el socket en el **lado cliente**, se ha de incluir la librería `socket.io` en el documento HTML. Permite hacer la conexión con el servidor y utilizar las funcionalidades del socket. Es importante que nuestras funciones con el socket se incluyan después de la librería `socket.io`, sino nos dará errores de funciones y variables no definidos. El siguiente código muestra este ejemplo:

```
<script src = "/socket.io/socket.io.js"></script>
<script src = "mi_script_socket.js" ></script>
```

Una vez incluida la librería, se tiene acceso a las funciones de los sockets. En este caso, el código del programador iría en el fichero `mi_script_socket.js`. Primero ha de establecer la conexión con el servidor, pasándole el nombre del host y el puerto:

```
var socket = io.connect(host + ":" + puerto);
```

Para realizar la emisión o recepción de datos a través de los sockets, se utiliza el objeto `socket` que se devuelve con la conexión. Este objeto tiene varias funciones, como son `on()` o `emit()`. La primera es para recibir datos y ejecuta una función *callback* al recibirlos. La segunda función permite enviar datos en el segundo parámetro. Pueden ser objetos, arrays, variables, etc.

```
socket.on("evento_recibir", callback);
socket.emit("evento_enviar", data);
```

Para configurar el socket del **lado del servidor** se realizan unos pasos muy similares. Para poder utilizarlo, hay que importarlo haciendo usando la función `require()` de Node. Hay que pasarle el servidor para que pueda obtener el host y el puerto de escucha. De esta forma se establece la conexión.

```
var io = require("socket.io")(server);
```

Una vez importado el módulo `socket.io`, la conexión se guarda en el objeto `io`. Este objeto estará esperando conexiones de usuarios mediante el evento *connection*. Cada nueva conexión crea un socket nuevo, que tendrá funcionalidades muy similares a las del lado del cliente, por lo que no es necesario explicarlas. Algunas funciones extras del socket del lado del cliente es la de poder unirse a salas, enviar en broadcast, detectar las desconexiones, etc. El siguiente código el

listener de conexiones de usuarios, devolviendo un socket que es pasado a una función anónima.

```
io.on('connection', function (socket){ . . .});
```

3.7 Jade

Jade es un motor de plantillas que se utiliza para generar las vistas HTML de forma dinámica. Tiene la particularidad de definir el DOM (*Document Object Model*) mediante indentación (ya sea por tabulaciones o por espacios, aunque con tabulaciones es más visual).

Al usar tabulaciones, no se han de cerrar los *tags*, al contrario de lo que ocurre con HTML. Los niveles del DOM se van definiendo añadiendo una indentación más al nodo hijo.

Otra característica de Jade es que permite definir estructuras de control (bucles o condicionales) basados en JavaScript. También permite crear variables JavaScript y operar con ellas o mostrarlas en la vista.

Para facilitar la creación de vistas dinámicas, Jade tiene la opción de heredar plantillas con una plantilla predefinida, y poder sobrescribir bloques de contenido predefinidos.

Junto con NodeJS puede generar los ficheros HTML. Antes de llamar a renderizar la vista, NodeJS pasa las variables que se han de utilizar en la plantilla Jade. Junto a la estructura de la plantilla y las variables pasadas, Jade genera el fichero HTML para que pueda ser entendido por el navegador. En la *Figura 11* se muestra este proceso.



Figura 11. Generación de fichero HTML con JADE

3.8 AngularJS

Se utiliza la versión 1.5.2 de AngularJS. Es un framework de JavaScript para el Front End, que permite crear y mantener aplicaciones SPA (*single-page application*) [4].

Utiliza un patrón MVC para el lado del cliente, como el que se muestra en la *Figura 12*. Los modelos y controladores están escritos en Javascript. Los modelos son los objetos que contienen los datos. Los controladores son los que interactúan con estos objetos y modifican las vistas.

Junto con el motor de plantillas Jade, convierte la plantilla en una vista. Para generar estas vistas, añade nuevos atributos (empiezan por el prefijo *ng-*) que permiten mostrar los datos de los modelos, realizar estructuras de control u obtener entradas del usuario.

Angular permite mostrar los cambios de los modelos en tiempo real de forma automática. Esto es muy útil, ya que evita tener que refrescar la página, o bien, hacer explícitamente una llamada a una función que se encargue de actualizar los cambios.

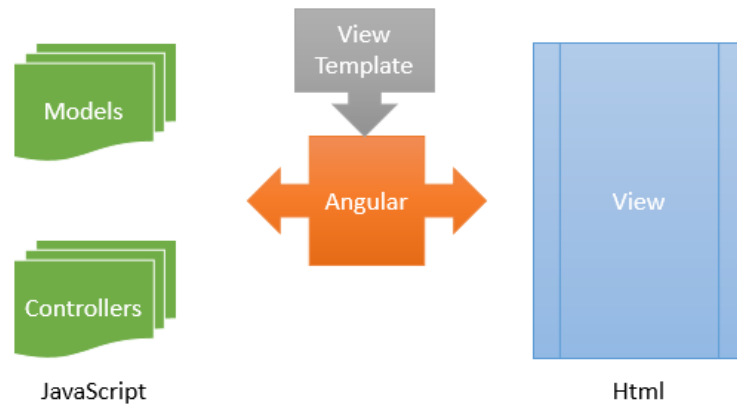


Figura 12. Patrón MVC de AngularJS

3.9 CSS

Es el lenguaje utilizado para organizar el aspecto de las vistas. Utiliza los *tags* o etiquetas de HTML, además de clases, identificadores o estados de los tags (*hover*, *checked*...) para aplicar los estilos y permite hacer que las vistas sean *responsive*.

Capítulo 4

ARQUITECTURA

En este capítulo se explicarán las diferentes arquitecturas utilizadas para el servidor, como para el cliente. También se mostrará la forma en que interactúan los sockets y cómo configurar e inicializar el servidor.

4.1 Configuración del servidor

El fichero donde se crea el servidor es `./bin/www`. En este fichero se indica información referente a la inicialización del servidor, como puede ser el nombre del host y el puerto al que escucha. Dicha información la obtiene de un fichero llamado `config.js`, y puede ser configurada por el administrador de forma manual antes de ejecutar el servidor. Se ejecuta con NodeJS.

Para gestionar las peticiones o añadir los módulos que trabajaran en el lado del servidor se utiliza ExpressJS, en el fichero `app.js` del directorio raíz (no confundir con el fichero `app.js` de AngularJS). En este fichero se declaran los ficheros públicos, el motor de vistas, el manejador de sesión, el manejador de cookies o el gestor de peticiones entre otros.

La conexión con la base de datos se realiza con el módulo `mongoose`, ya comentado anteriormente. Se carga en el fichero `app.js` y establece la conexión con la base de datos MongoDB. Necesita que se le pase el nombre del servidor y el puerto, además del nombre de la base de datos.

El fichero `app.js` también carga dinámicamente los módulos de NodeJS que contienen los controladores de la carpeta `./controller/classes/` (Más adelante se explicarán que son estos controladores). Utiliza el módulo `File System` de Node. Al hacerlo dinámico, no hay que preocuparse en actualizar el fichero `app.js` si se añade o elimina algún controlador.

La *Figura 13* muestra el esquema de cómo se inicializa que se acaba de explicar.

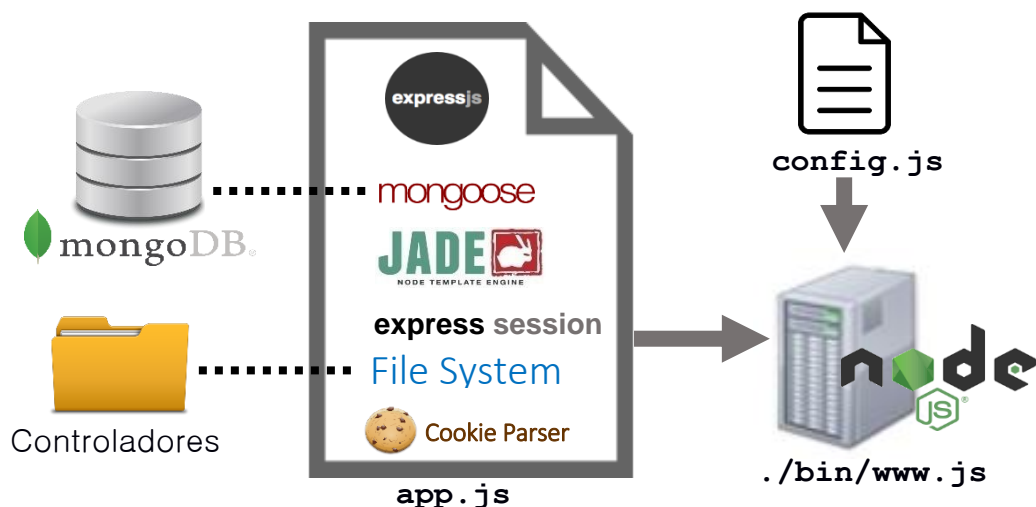


Figura 13. Inicialización del servidor

4.2 Patrón MVC

La arquitectura que se ha elegido para implementar la aplicación ha sido el patrón MVC (modelo-vista-controlador), con el fin de tener independencia entre los datos, los eventos y la interfaz de usuario.

A grandes rasgos, su funcionamiento consiste en que el **cliente** hace peticiones al controlador. A partir de estas peticiones, el **controlador** llama al modelo. El **modelo** interactúa con la base de datos, y devuelve los datos resultantes al controlador. A continuación, el controlador pasa estos datos a una plantilla para crear la **vista**. Esta vista será la respuesta que obtiene el usuario, que puede ser un HTML, un JSON, etc. Este funcionamiento se ilustra en la *Figura 14*.

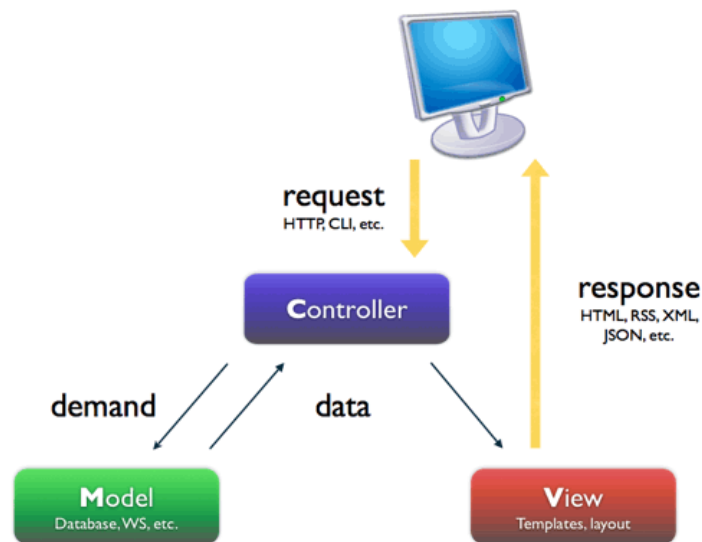


Figura 14. Esquema de funcionamiento del patrón MVC

4.2.1 Modelo

El modelo es la parte encargada de interactuar con la base de datos y retornarlos, si es necesario. Hay 2 modelos, uno relacionado con los datos de las partidas, y el otro con los datos del usuario. En ambos se define la estructura del modelo y los métodos que se les pueden aplicar. Para conectarse con la base de datos, hacen uso del módulo `mongoose` de NodeJS. Las estructuras comentadas se crean con el método `mongoose.Schema()`, definiendo los campos y de qué tipo son cada uno. Se definen como un objeto JSON.

Una vez definida la estructura, se pueden crear objetos de estos esquemas. Al crear objetos, podemos definir métodos de instancia (`schema.methods`) o métodos estáticos (`schema.static`). Los primeros son los que se ejecutan sobre instancias del *modelo* que se haya creado. Un ejemplo de operación podría ser la de actualizar un campo del objeto y guardarlo en la base de datos. En cambio, los métodos *statics* sirven para realizar acciones sobre el *Model* (similar a lo que es una clase en programación orientada a objetos), cómo puede ser buscar varios registros que cumplan una condición.

Los modelos hacen las operaciones de forma asíncrona. Esto significa que aunque la ejecución del controlador que ha llamado al modelo haya finalizado, la operación en el modelo puede estar aún en proceso. Por esta razón, a todos los métodos creados se les pasa una función *callback*, que será ejecutada cuando finalice el modelo, y el controlador pueda hacer uso de los datos.

4.2.2 Vista

Las vistas son las plantillas que generarán una interfaz al usuario. Los datos que presenta son del modelo, pero no vienen directamente de él. El controlador es el encargado de obtener los datos del modelo y pasárselos a la vista.

El motor de plantillas es *Jade*, que permite generar código HTML de forma más rápida. Utiliza la indentación para generar los diferentes niveles del DOM del HTML, por lo que es muy importante la tabulación que damos a cada *tag* de la plantilla de *jade* para crear la jerarquía del DOM.

Para actualizar y obtener datos de forma asíncrona, se utiliza el framework AngularJS. Permite ver los cambios que haya en los modelos (*de Angular*) en tiempo real. El uso de los sockets también ayuda a realizar estas actualizaciones. Los archivos de angular están organizados en la carpeta pública *AngularJS*. Está estructurada en diferentes módulos (socket-cliente, partidas y usuarios). Cada módulo crea un servicio para operar con los controladores. En el caso de los módulos de *partidas* y *usuarios*, sus métodos hacen llamadas a controladores para obtener datos del modelo de la base de datos. En cambio, el módulo del *socket* se utiliza para enviar y recibir información entre los diferentes sockets existentes, enviando dicha información al servidor, y a partir del servidor se reenviará a los demás clientes si es necesario. En el apartado de *Comunicación entre sockets* se detallará el funcionamiento de los sockets.

Para hacer uso de estos módulos, el fichero `app.js` de AngularJS crea un módulo que importa los servicios anteriores, y crea controladores de angular (los llamaremos *CA* para no confundir con los controladores del patrón MVC). Los CA son insertados en la plantilla con el atributo *ng-controller*. Una vez insertado, crea un objeto llamado `$scope`, que es accesible para todos los hijos de la etiqueta del DOM al que se asignado.

El `$scope` contiene los datos y funciones que pueden ser llamados desde la plantilla. Las funciones del `$scope` pueden hacer llamadas a los métodos de los servicios antes creados, permitiendo actualizar los datos. Esta variable es accesible únicamente desde el nodo del DOM que se ha creado, hacia sus nodos hijos, por lo que es importante tenerlo en cuenta para estructuras de control como bucles (se genera un `$scope` por cada repetición).

El fichero `app.js` tiene otra parte importante. Es la inicialización del `$rootScope`, similar al `$scope`, pero accesible por cualquier elemento del DOM que utilice el módulo '`app`'. El `$rootScope` se inicializa de tal forma que permite obtener las cookies del usuario. De esta forma, cualquier nodo que haga uso de éste módulo, podrá mostrar información referente del usuario.

4.2.3 Controlador

Los controladores son los encargados de hacer de enlace entre el modelo y las vistas. Son llamados por URL siguiendo este el patrón: **`http://servidor/controlador/metodo/argumentos`**

Para poder utilizar los controladores, se cargan dinámicamente desde el fichero `app.js` de Express. Utiliza el módulo *filesystem* de Node para realizar la carga asíncrona de los ficheros de controladores. De esta forma, no es necesario cargar cada controlador de forma manual.

Existen 4 controladores para gestionar distintos tipos de datos: usuarios, partidas, subida de fichero y gestor de sesión.

Los controladores de usuarios y partidas permiten realizar acciones sobre los modelos de usuario y partidas respectivamente. Cuando realizan una llamada a una operación de dichos modelos, envían la información a la vista. En ocasiones, simplemente devuelven los datos en un JSON, ya que puede que no sea necesario visualizar la información en una vista. Este último caso suele ser usado para hacer llamadas por asíncronas desde AngularJS con el módulo `$http`.

El controlador de las subidas de ficheros permite comprobar que el tipo de fichero que el

usuario quiere enviar al servidor es correcto. Actualmente solo soporta imágenes jpg, gif y png. Para comprobar que verdaderamente son imágenes de dichos tipos, se mira el MIME del fichero y se compara con la firma hexadecimal [24] para verificar que realmente es una imagen.

El controlador de gestión de sesión es el encargado de controlar tanto la sesión del usuario en el servidor, como las cookies. Permite mantener la sesión activa aunque el servidor se tenga que reiniciar.

4.3 Base de datos

El gestor de la base de datos es MongoDB. Como ya se indicó en el capítulo anterior, utiliza documentos y colecciones. Existen 2 colecciones: una para guardar datos sobre las partidas y otra para usuarios. La estructura de estos documentos no tiene necesariamente todos los campos, pero se han definido todos campos posibles que puede tener cada uno.

A continuación se muestra la estructura de los documentos en formato JSON. Se indica el nombre del campo, seguido por el tipo de datos del campo. En caso de que el campo pueda ser algo confuso, se añade un comentario que explica en qué consiste dicho campo. Todos los documentos tendrán el campo *_id*, añadido por defecto, y que permite diferenciarlos. Las estructuras de los documentos que se guardan en la colección de *partidas* pueden tener los siguientes campos:

```
{
  creador: String,
  titulo: String,
  numJugadores: Number, // Número máximo de jugadores que pueden haber.
  tiempo: Number,       // Duración de la partida (minutos).
  ctime: Date,          // Fecha de creación, por defecto la fecha actual.
  iniciada: Date,       // Hora de inicio de la partida.
  acabada: Boolean,
  solucionada: Boolean,
  autoinicio: Boolean, // Inicia partida automáticamente al llegar al máximo
                        // de jugadores (campo numJugadores).
  jugadores: [String], // Array que contiene el nombre de usuario de los
                        // jugadores de la partida.
  abandono: [{         // Array con el nombre del usuario que ha abandonado
    usuario: String,   // la partida, junto a la hora que la abandonó.
    hora: Date
  }],
  preguntas: [{        // Preguntas que componen la partida.
    ctime: Date,
    pregunta: String,
    ganador: String,   // Nombre de usuario del ganador de la pregunta.
    respuesta: String, // Define la respuesta por defecto al crear la pre-
                        // gunta.
    respuestas: [{     // Respuestas de los usuarios a la pregunta
      usuario: String, // Nombre de usuario
      resultado: Number, // Sirve para identificar al ganador. 0 = Per-
                        // dedor, 1 = Ganador
      historial: [{    // Historial de respuestas del jugador. Se
                        // ordenan por tiempo.
        respuesta: String,
        tiempo: Date
      }
    ]
  }
  ]
}
```

La estructura de la colección de los *usuarios* es esta:

```
{
  usuario: String,      // Nombre de usuario. Ha de ser único.
  password: String,
```

```

nombre: String,
apellidos: String,
nacimiento: Date,
avatar: String,           // Foto del usuario. Por defecto usa la foto de usua-
                           // rio DEFAULT_IMG definida en las constantes.
notificaciones: [{       // Array con las notificaciones pendientes.
  ctime: Date,
  partida: String,
  tipo: Number           // Números identificativo. Se muestran en la constante
                        // NOTIFICAR de modules/global.js.
}]
}

```

4.4 Comunicación entre sockets

La comunicación entre los sockets consta de dos partes: una en el servidor y otra en el cliente. Permite comunicar a los diferentes usuarios de la aplicación entre ellos y con el servidor. Los llamaremos ‘socket servidor’ al que se encuentra en el lado del servidor, y ‘socket cliente’ al que se encuentra en el lado del cliente.

El **socket cliente** es el encargado de enviar información cuando el usuario realiza algún cambio que afecte a uno o varios usuarios, como puede ser crear una partida nueva, responder una pregunta o iniciar una partida.

El código del socket cliente se encuentra en el fichero `./public/js/angularjs/modulos/socket-client.js`. Es un servicio que se ha creado para AngularJS, ya que será utilizado en el fichero `app.js` de AngularJS. Establece la conexión con el servidor para poder realizar la comunicación. También contiene *listeners* o actuadores que están a la espera de recibir el evento del socket servidor para recibir los datos y realizar las operaciones necesarias.

El **socket servidor** realiza dos funciones. La primera es la de notificar a los demás usuarios cuando ocurran eventos que deban de ser notificados, como por ejemplo, que una partida ha finalizado. Para ello, envía la los datos en *broadcast* (a todos los usuarios) o en *unicast* (a un usuario concreto) a los sockets que estén unidos en la sala. Para ello, es necesario que el usuario se haya unido previamente a la sala.

El proceso de unirse a la sala es totalmente transparente para el usuario, ya que es el servidor el que se encarga de unir o abandonar al usuario a las partidas. Este proceso se realiza por cada vez que el usuario refresca la página, ya que una característica de los websockets es que no son persistentes (se crea un nuevo socket por cada nueva conexión, eliminando el socket previamente existente).

La otra funcionalidad del socket servidor es la de servir de nexo entre la comunicación entre dos clientes, como muestra la *Figura 15*. En este ejemplo, el cliente A quiere enviar datos a los clientes B y C. El proceso consiste en que el cliente A envía los datos desde el *socket cliente* al servidor. El *socket servidor* tiene un *listener* que recibe las solicitudes de los clientes, y las reenvía en a todos los usuarios a los usuarios indicados. En este proyecto, se reenvía a los usuarios de la misma sala (partida), por lo que en el ejemplo de la *Figura 7*, los clientes A, B y C estarían jugando la misma partida. Los clientes destinatarios tienen un *listener* del evento que emite el servidor para obtener los datos.

Algunos ejemplos en el proyecto que utilice este tipo de comunicación es para crear nuevas preguntas en una partida iniciada (creador) y los demás usuarios puedan visualizarla (los jugadores); o que un usuario abandone o se una a una partida aún no iniciada.

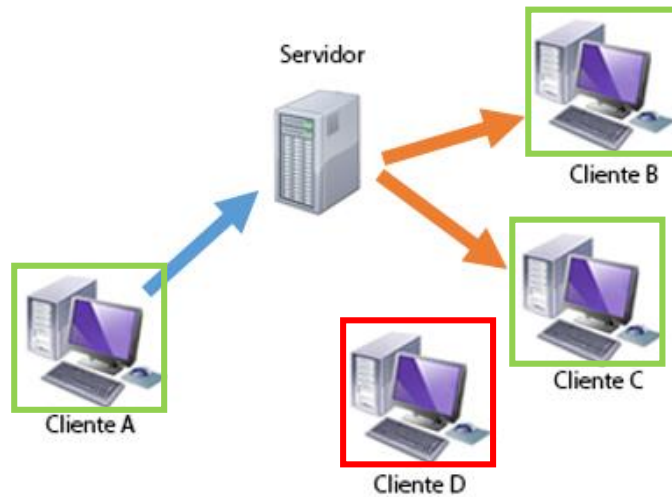


Figura 15. Funcionamiento de websocket en broadcast. Los recuadros verdes indican que están en la misma sala. La flecha azul indica que envía el socket cliente, y la naranja el socket servidor.

Algunas veces puede ser necesario que el servidor no tenga que notificar a todos los usuarios de una misma partida. Esto ocurre cuando un jugador responde una pregunta, y basta notificarlo al usuario creador, que es quien puede ver las respuestas de todos los usuarios en tiempo real. Para lograr esta función, únicamente el creador de la partida está suscrito a un evento que le enviará el mensaje solamente a él una vez que suceda. La *Figura 16* muestra este funcionamiento. Cuando un jugador de una partida responde una pregunta (cliente A), notifica a través del socket cliente al servidor. Una vez que el servidor recibe la respuesta, el socket servidor avisa al creador (cliente B) para que se actualice el historial de respuestas. El resto de jugadores de una misma sala no recibirán la notificación (en este caso, el cliente C).

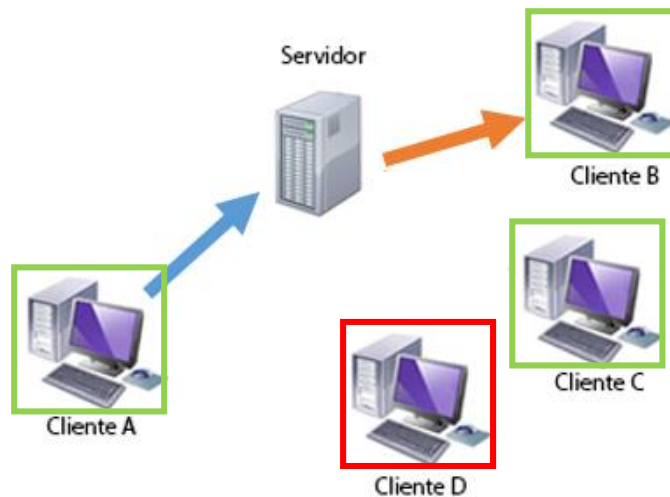


Figura 16. Funcionamiento de websocket en unicast.

4.5 Cookies y sesiones

Para poder acceder a la aplicación, es necesario disponer de un nombre de usuario y de una contraseña, e introducirlos en el formulario de inicio de sesión. Una vez que el usuario entra en la aplicación, es necesario que el servidor mantenga la sesión activa para evitar tener que introducir las credenciales cada vez. Para ello, se utilizan las cookies y la sesión.

La **sesión** almacena la información del usuario (nombre de usuario, nombre y apellidos, fecha de nacimiento...) en la parte del servidor. Para acceder a las diferentes funcionalidades de la aplicación, es necesario que dicha sesión haya sido establecida. En caso contrario, se redirige al usuario al formulario de inicio sesión o registro. Esta tarea para comprobar la sesión es llevada a cabo por una función que se encuentra en el fichero `app.js`, llamada `checkSession`. Es pasada a todos los *manejadores de ruta* [29] que necesiten haber iniciado sesión previamente, y se ejecuta antes que el código de cada manejador. En caso de que la sesión esté establecida, ejecutará el contenido del manejador. También se encarga de actualizar las cookies y la sesión, ya que algún campo puede haber cambiado.

La estructura de un manejador de ruta que hace uso de dicha función es muy similar al ejemplo que se muestra a continuación:

```
app.get("/url", checkSession, function(req, res){ ... })
```

Las **cookies** contienen información prácticamente igual que la sesión, pero en el lado del cliente. Permite mantener la sesión del usuario en el navegador, por lo que se puede cerrar el navegador, y al abrirlo, seguirá la sesión activa sin necesidad de volver a introducir el nombre de usuario y contraseña.

Otra posibilidad que aporta el uso de las cookies es la de acceder a la información del usuario sin necesidad de tener que hacer una petición cada vez que se necesite alguno de sus campos. Esta funcionalidad es aprovechada por AngularJS para crear el modelo referente al usuario. Basta utilizar el módulo de angular llamado `$cookies` para trabajar con las cookies. En caso de no haber usado cookies, se tendría que hacer una petición con el módulo `$http` de Angular, para pedir la información del usuario. De esta forma, se libera trabajo en el lado del servidor.

Capítulo 5

PRUEBAS

En este capítulo se muestran las pruebas de carga que se han realizado. Todas estas pruebas consisten en hacer peticiones de forma paralela con varios usuarios conectados e interactuando a la vez. En principio se pretendía hacer las pruebas con 100, 1.000 y 10.000 usuarios a la vez, pero por limitaciones de la herramienta Selenium, se ha tenido que bajar a 10 el máximo número de instancias. Por lo tanto, estas pruebas harán referencia a 1, 5 y 10 usuarios a la vez. Los ejecutables para hacer estas pruebas se encuentran en la carpeta *tests* del directorio raíz de la aplicación.

Los resultados obtenidos pueden variar dependiendo de varios factores, como son la velocidad de la conexión a internet o el equipo que se utiliza. Para esta prueba se ha usado un portátil ACER con 8GB de RAM y un CPU de doble núcleo a 1'6GHz. La conexión a internet es de 20Mbps de descarga.

Los datos usados para crear usuarios y partidas se encuentran en el fichero */tests/config.js*. Todos los usuarios tienen el mismo nombre, contraseña o apellidos, ya que estos valores no afectan al rendimiento. Los nombres de usuarios se generan dinámicamente, con el prefijo *usuario_* seguido de un número que se va incrementando.

La primera prueba consiste en registrar varios usuarios a la vez. Es el fichero **registro.js** de la carpeta *tests*. Este script introduce los datos en la página de inicio, y calcula el tiempo que tarda en enviar la información, hasta que el servidor la guarda. Al ejecutar este script tres veces con 1, 5 y 10 usuarios a la vez, los tiempos medios son 4.062, 21.013 y 41.326 ms respectivamente. Este valor contiene el tiempo en refrescar el navegador, por lo que es normal que crezca tanto aunque haya tan pocos usuarios. Si solo tenemos en cuenta el tiempo de enviar los datos de registro y que el servidor los haga la redirección a la página principal (esto significa que el usuario se ha registrado), el tiempo medio es de unos 926 milisegundos de media.

Otra prueba que se ha intentado realizar es la de comprobar cómo afecta el rendimiento de una partida con varios usuarios jugando, pero que no se ha podido llevar a cabo porque no se ha encontrado la forma de obtener el tiempo en transferir la respuesta a través de los sockets con la herramienta Selenium.

Capítulo 6

FUTURO TRABAJO

En el proyecto realizado hay varios aspectos que son interesante que se implementasen en futuras versiones de la aplicación. En este apartado se explicarán algunas mejoras o actualizaciones que no se han añadido de inicio, pero que se han tenido en cuenta y podrían ser introducidas en trabajos futuros.

6.1 Ley de protección de datos

La Ley Orgánica de Protección de Datos (*LOPD*) [2] es una ley orgánica española, que tiene como objetivo garantizar y proteger las libertades, privacidad y derechos de las personas físicas en lo referente en el tratamiento de los datos personales, independientemente del soporte en el que sean tratados.

En esta aplicación se piden varios datos personales (nombre, apellidos y fecha de nacimiento), y en un futuro, podrían ser necesarios otros datos (como el sexo, centro educativo, correo electrónico...). Por lo tanto, es necesario cumplir con la *LOPD* [1] para evitar sanciones que pueden rondar entre los 600€ hasta los 600.000€.

Una medida a añadir sería la de informar a los usuarios de los datos que se recogen, cómo se van a utilizar y obtener su consentimiento. Además, se ha de permitir al usuario tanto modificar o visualizar sus datos, como poder darlos de baja.

Para proteger los datos de los usuarios, se han de aplicar una serie de medidas de seguridad para garantizar que un intruso o persona no autorizada no pueda acceder a los datos. Utilizar contraseñas (renovadas cada año) para acceder a los datos y encriptar dichos datos pueden ser medidas de seguridad oportunas para proteger la información del usuario.

Las copias de seguridad de estos datos también son obligatorias para evitar su pérdida. Estas copias han de ser semanalmente, y no podrán contener información que no se hayan acordado en las condiciones aceptadas por el usuario.

6.2 Panel de control y privilegios

Al ser una aplicación destinada para el ámbito de la educación, es necesario aplicar un sistema de control del tipo de imágenes que el usuario puede subir o los nombres de usuarios elegidos.

Una solución podría ser la de tener un panel de control, en el que un administrador con privilegios pueda ver las fotos subidas o los nombres de usuarios utilizados, y aceptarlos o rechazarlos.

Con la introducción de este panel, sería necesario asignar privilegios a los usuarios por roles (por ejemplo: moderador, profesor, alumno...). Esta tarea también se tendría que implementar

en versiones futuras, ya que actualmente no diferencia entre roles. No sería una tarea complicada, ya que bastaría añadir un nuevo campo a los usuarios para que identifiquen su rol.

Con esta actualización, al haber diferentes privilegios, se podría restringir a los profesores a crear partidas, y que lo alumnos solo puedan unirse a las partidas. Actualmente, al no disponer de diferentes privilegios, cualquiera que se registre en la aplicación puede crear y unirse a las partidas. Aun así, en la aplicación actual se diferencia entre el creador de la partida y los jugadores, para impedir realizar algunas acciones como que un creador pueda unirse a una partida creada por él, que un jugador pueda seleccionar las respuestas correctas de una partida que no haya creado o que un jugador no pueda ver las preguntas de una partida iniciada y que no se unió.

6.3 Control del almacenamiento de datos

A medida que se va utilizando la aplicación, y se van creando o modificando partidas o usuarios, se va guardando información que puede llegar a no ser necesaria, ocupando espacio de almacenamiento. Por ejemplo, una partida que haya sido creada hace mucho tiempo y ya no nos interesa su información, seguirá en la base de datos. Algo similar pasa con usuarios que ya no utilizarán la aplicación (e.g. han acabado el curso o se han dado de baja del centro) están ocupando espacio del disco, además de estar utilizando un nombre de usuario que podría ser usado por un nuevo usuario.

Para poner remedio a este problema, se puede utilizar el panel de control visto anteriormente, en el que un administrador podría eliminar las partidas o usuarios que vea conveniente.

6.4 Nuevos tipos de partidas

En versiones futuras sería interesante añadir más tipos de partidas a la aplicación, como puede ser elegir entre varias imágenes, preguntas con múltiples posibles respuestas o hacer preguntas referentes a un vídeo que se haya subido. Para las preguntas con un video, se podría optar por una aplicación de terceros (por ejemplo, YouTube) que tenga una API y permita integrarla en nuestra aplicación. De esta forma, no se tendría que almacenar los videos en un servidor propio, malgastando espacio de memoria.

Otro aspecto interesante sería el de mostrar las partidas del centro académico al que pertenece el alumno. Actualmente se muestran todas las partidas, ya que no se ha implementado la opción de poder definir el centro en el que se encuentra el alumno.

6.5 Implementación de aplicación para móviles

Con el fin de que sea más fácil poder utilizar la aplicación, se puede desarrollar en otras plataformas. La primera sería para móviles, debido al gran número de personas que tienen este dispositivo.

Habría que tener en cuenta en qué sistemas implementarlo (Android, iOS, Windows Phone...), ya que cada uno necesita un desarrollo diferente. Sería necesario hacer un estudio de mercado que nos mostrará qué tipo de dispositivo suelen usar los alumnos y profesores que hacen uso de esta aplicación. Se podría hacer mediante encuestas online a los usuarios registrados, preguntando el tipo de dispositivo que utilizan.

6.6 Ajustar el diseño para múltiples navegadores

El diseño de la aplicación es *responsive*, lo que quiere decir que se adapta al tamaño de la ventana de nuestro navegador, dependiendo de la resolución. Al no ser una aplicación comercial, se ha tenido en cuenta el diseño para el navegador *Google Chrome* para Linux. El ajuste para los demás navegadores y sistemas no es una tarea complicada, pero sí que necesita mucho tiempo de dedicación para comprobar los distintos navegadores. Al ser una aplicación desarrollada únicamente por una persona, en esta versión es suficiente que el diseño sea correcto en al menos un navegador.

Un ejemplo es el navegador FireFox, que hay problemas con algunos tamaños de fuentes, las cuales son más grandes. Por otro lado, el navegador Opera muestra algunas entradas de texto ligeramente más pequeñas que en el diseño original. Como se puede ver, no son desajustes graves, pero estaría bien solucionarlos en futuras versiones.

Capítulo 7

CONCLUSIÓN Y VALORACIÓN PERSONAL

Este proyecto está destinado para el ámbito de la enseñanza, como ya se ha explicado en el documento. Al estar enfocado en dicho sector, se ha buscado información sobre cómo la tecnología ayuda al aprendizaje y la enseñanza. Al realizar una simple búsqueda en *google* sobre temas relacionados sobre e-learning, TIC en enseñanza o sus beneficios, obtenemos una gran cantidad de artículos. Muchos de estos artículos muestran la importancia de añadir la tecnología en la enseñanza, ya que aporta varios beneficios, entre ellos, aumentar el interés del alumno, facilitar el acceso a la educación o mejorar su rendimiento académico, entre otros beneficios.

Esta aplicación ayudará a los profesores a evaluar los conocimientos de los alumnos mediante preguntas de respuesta corta. De esta forma, el profesor podrá obtener conclusiones sobre la evolución de sus alumnos, y poder hacer ajustes en la metodología o el temario impartido, en caso de que sea necesario.

Para que el alumno también se vea beneficiado, el sistema de la aplicación que permite definir ganadores en cada pregunta hace que los estudiantes compitan entre ellos para obtener la mejor puntuación en cada partida. Para ello, deben de haber prestado atención en las clases y estudiar sus temarios.

Referente a las tareas planificadas, la mayoría se han podido llevar a cabo en los tiempos establecidos. El uso de la metodología de desarrollo en cascada ha facilitado conseguir los objetivos, ya que permite secuenciar las tareas una detrás de otra y facilitando el seguimiento del desarrollo.

Las tareas de la fase de análisis (indicar los requerimientos de la aplicación, estudio de aplicaciones similares y la definición del cronograma) se realizaron sin muchos problemas. Únicamente hubo un cambio en los requisitos iniciales durante la fase de desarrollo, en la que se introdujo nuevas funcionalidades a las partidas que no se contemplaron en los requisitos de la tarea A01 (*Requerimientos de la aplicación*). Aun así, supuso un retraso total en el proyecto de 5 días, concretamente en las tareas D05 (*Diseño de la estructura de las partidas*), en la I01 (*desarrollo de la parte del servidor*) e I02 (*Desarrollo de la parte de cliente*). Este retraso ha afectado a la fase de implementación, y por consiguiente, a las fases de pruebas y entrega. El cronograma final ha quedado como el que se muestra en la *Figura 18*.



Figura 17. Cronograma final

Las tareas de la fase de diseño y de implementación se pudieron realizar en los tiempos marcados, ya que no hubo cambios en la planificación, excepto el comentado en el párrafo anterior.

En la fase de pruebas, en un principio, se pretendía hacer una prueba de usabilidad con alumnos de la universidad, pero debido a que el proyecto se ha realizado en verano, ha sido imposible realizar esta prueba, por lo que no se ha añadido tan si quiera en la planificación. Esta prueba sería interesante realizarla al comenzar el curso lectivo, y así, poder hacer los cambios oportunos en versiones futuras para facilitar el uso de los alumnos.

En esta fase de pruebas, se ha hecho una prueba de carga del servidor, utilizando la herramienta Selenium. El problema ha sido al intentar ejecutar de forma concurrente más de 15 usuarios en la misma máquina, ya que por cada usuario, Selenium abre un nuevo navegador. Al abrir tantos navegadores en un mismo ordenador, no era posible realizar las pruebas que se hubiesen querido hacer, por lo que en caso de volver a hacer estas pruebas, debería buscar otra herramienta que permitiese hacer varias peticiones en paralelo. Esta es la única tarea que no se ha conseguido realizar de todas las que se marcaron en el principio.

En cuanto a la valoración personal sobre el desarrollo de este proyecto, he aprendido a realizar comunicaciones en tiempo real entre usuarios conectados en distintos ordenadores con ayuda de los sockets. En varias asignaturas de la carrera relacionadas con las redes hemos visto de forma teórica cómo realizar esta comunicación, pero con este trabajo he puesto en práctica lo aprendido en estas asignaturas.

También me ha aportado el conocimiento de nuevos frameworks de desarrollo web, como son el uso del *stack* MEAN. Por un lado, he aprendido a utilizar bases de datos no relacionales, que no las habíamos visto durante la carrera. Su aprendizaje no ha sido muy duro, debido a que existen diversos tutoriales en internet que explican cómo funcionan. En cuanto a Node, me ha enseñado cómo funciona la programación orientada a eventos, además de aprender cómo configurar un servidor desde cero. Gracias a los tutoriales que hay en internet y los conocimientos que tenía de PHP y JavaScript, su aprendizaje no fue complicado. Por último, el framework AngularJS sí que ha sido algo más complicado, debido a que incluye nuevos conceptos para organizar el código del lado del cliente.

Yo como estudiante, he utilizado aplicaciones similares durante la carrera. Una de ellas ha sido el uso de cuestionarios a través de *moodle* en asignaturas como matemática discreta, en las que se preguntaba sobre algún tema visto en clase, y debíamos escribir o seleccionar la respuesta correcta. Su uso ayudaba a absorber mejor los conceptos, ya que tenías que prepararte para estas pruebas. Por lo tanto, aplicaciones similares como la que se ha desarrollar en este proyecto nos ayuda a los estudiantes a entender mejor los temas. En algunas asignaturas de la carrera en las que hay muchos conceptos teóricos, como las de la rama de ingeniería del software, hubiesen sido más sencillas si se hubiese utilizado una aplicación como *Búho*.

Como se vio en el *Capítulo 3 - Tecnología utilizada*, para realizar el diseño de la página se utilizó únicamente CSS, utilizando mucho tiempo para conseguir el diseño. La inclusión de una librería como *bootstrap* para realizar el diseño de la aplicación habría ahorrado mucho tiempo. Además, como *bootstrap* está mantenido por una comunidad, los fallos que puedan tener son menores que los que puedan aparecer en un diseño propio.

Referente al uso de la tecnología para estructurar la página, cambiaría al uso de HTML en vez de Jade. Aunque con Jade no has de escribir tantas líneas de código, el sistema de tabulación puede llegar a ser algo confuso si necesitas hacer cambios en una plantilla o incorporar código de terceros. Además, el lenguaje HTML dispone de mucha más información que Jade, por lo que habría agilizado este proceso.

ANEXO A

MANUAL DE INSTALACIÓN

El objetivo de este manual es mostrar cómo instalar la aplicación. Se instalará sobre un Ubuntu 12.04 en 64 bits. Para realizar la instalación, serán necesarios los siguientes programas y herramientas:

- MongoDB v3.2
- NodeJS v4.4.6
- NPM
- Módulos de NodeJS
- Nodemon
- Selenium

Conviene instalar todos los componentes en el orden que se van explicando en este manual para evitar tener problemas.

A.1 MongoDB

Para instalar la base de datos **MongoDB**, es necesario seguir una serie de pasos. Se puede consultar la referencia [25] para instalar en otras versiones de Linux o en otros sistemas operativos. Los pasos para realizar la instalación de MongoDB 3.2.8 en Ubuntu 12.04 son:

1. Primero se ha de importar la clave pública usada por el gestor de paquetes, que permite autenticar el paquete que se va a descargar.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
```

2. Después se añade el repositorio en el fichero *list*. Depende de la versión utilizada de Ubuntu, esto puede variar.

```
echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

3. Para poder descargar MongoDB, es necesario actualizar la base de datos de paquetes.

```
sudo apt-get update
```

4. Finalmente, se instala la última versión estable de Mongo. En este caso, se utiliza la versión 3.2.

```
sudo apt-get install -y mongodb-org
```

A.2 Node y NPM

Node será el framework que se encarga del lado del servidor, como se explicó en el *Capítulo 3*.

Al instalar Node, también tendremos su gestor de paquetes NPM, que permite instalar nuevos módulos en Node. [26]

Node necesita varias dependencias, entre ellas *rlwrap*. No entraremos en detalles en que consiste este programa, ya que es una dependencia propia de Node. Esta dependencia puede darnos error al instalar Node, por lo que conviene instalarla previamente. Para ello, la añadimos en el repositorio, actualizamos el sistema e instalamos *rlwrap*:

```
echo "deb http://us.archive.ubuntu.com/ubuntu vivid main uni-  
verse" | sudo tee /etc/apt/sources.list  
sudo apt-get update  
sudo apt-get install rlwrap
```

Una vez instalada manualmente esta dependencia, podemos instalar la versión 4.4.6 de NodeJS. Para ello, añadimos el repositorio que lo contiene a nuestro sistema:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv 68576280  
apt-add-repository 'deb https://deb.nodesource.com/node_4.x pre-  
cise main'
```

Después se ha de actualizar el Sistema para que pueda acceder a dicho repositorio, y finalmente instala node.

```
sudo apt-get update  
sudo apt-get install nodejs
```

Otra forma de instalar node es utilizando el comando *wget*, que descarga un script que contiene los comandos que se ejecutarán automáticamente, en vez de añadir el repositorio. Bastaría escribir estas dos líneas:

```
wget -qO- https://deb.nodesource.com/setup_4.x | sudo bash -  
sudo apt-get install nodejs
```

Para comprobar que todo ha ido correctamente, podemos escribir en la pantalla de comandos `node -v` o `npm -v` para que nos devuelva la versión instalada.

A.3 Módulos de NodeJS (ExpressJS, socket-io...)

El fichero *package.json* contiene el nombre de todos los módulos que se han utilizado en la aplicación. Este fichero se encuentra en el directorio raíz de la aplicación, que es donde nos situaremos. Para que se instalen todos los módulos, basta escribir:

```
npm install
```

Automáticamente se instalarán todos los módulos. En caso de que alguna versión de alguna dependencia no exista, se puede instalar de forma manual mediante el comando:

```
npm install <nombre_paquete> --save
```

A.4 Nodemon

Es la herramienta utilizada para agilizar el proceso de desarrollo de la aplicación, por lo que su instalación no es necesaria si no se van a realizar varios cambios en el código. Se utiliza el gestor de paquetes de Node (*npm*):

```
sudo npm install -g nodemon
```


A.5 Iniciar servidor

Una vez instalados todos los componentes, se puede iniciar el servidor con nodemon o con Node. Si estamos desarrollando o haciendo cambios en la aplicación, conviene usar nodemon, ya que reiniciará el servidor de forma automática cuando encuentre cambios. Se ejecuta desde el directorio raíz, escribiendo el siguiente comando:

```
nodemon bin/www
```

Si por el contrario tenemos la aplicación como versión final, o no vamos a hacer ningún cambio, se puede ejecutar con Node con el siguiente comando:

```
node app.js
```

Finalmente se puede ir al navegador para comprobar que funciona correctamente. En caso de haberlo instalado en el servidor local y esté escuchando el puerto 3000, se tendría que escribir:

```
http://localhost:3000/
```

A.6 Selenium

Selenium [28] permite realizar acciones web de forma automáticas sobre Node. Se utiliza para realizar pruebas de carga sobre la aplicación creada, y comprobar la evolución del rendimiento dependiendo de la cantidad de usuarios conectados de forma paralela.

Para instalarlo, se puede utilizar el gestor de paquetes de Node, NPM:

```
npm install selenium-webdriver
```

A continuación hay que descargar el ejecutable del navegador, en este caso, chromedriver, desde esta [URL](#). La versión utilizada es la [2.23](#) para Linux.

Una vez descargado, hay que extraerlo en el siguiente directorio para que pueda tener acceso a este ejecutable:

```
/usr/bin
```

Finalmente, se puede ejecutar el código de prueba que hay en la carpeta *tests* del directorio raíz de la aplicación. Antes de iniciar algún fichero, es necesario iniciar el servidor. Para ejecutar el fichero, se utiliza el comando *node*:

```
node <nombre_programa>.js
```


ANEXO B

MANUAL DE USUARIO

Este apartado muestra un manual de usuario con ejemplos reales de la aplicación. Se seguirá el diagrama de flujo mostrado en la sección X del *Capítulo 4 – Arquitectura*. Las acciones triviales, como el registro o inicio de sesión, edición de usuario o visualización de notificaciones no se explicarán en detalle, ya que es muy similar a cualquier aplicación web. Este manual no pretende entrar en detalles técnicos, solamente ser una guía para poder utilizar la aplicación correctamente.

Lo primero que hay que hacer es iniciar sesión con nuestra cuenta. En caso de no tenerla, podemos registrando completando le formulario de registro. El nombre de usuario tiene que ser único.

Al iniciar sesión, se mostrarán todas las partidas que haya disponibles para unirse. Toda la aplicación mantendrá la estructura que se muestra en la *Figura 19*. En la cabecera, aparecen varias opciones. La primera es la de poder buscar partidas manualmente en el cuadro de texto. Se pueden buscar partidas creadas por usuarios o por títulos de partidas. El nombre de usuario ha de ser exactamente igual que el que queremos buscar. Para buscar por título, basta escribir una aproximación del nombre de la partida, ya que este tipo de búsqueda se realiza con una expresión regular. Por ejemplo, para buscar la partida “*Preguntas de historia*” bastaría escribir “*historia*”, y devolvería todas las partidas que contengan esta cadena de caracteres.



Figura 18. Partidas disponibles

A la derecha del cuadro de búsqueda aparece un botón rojo que permite crear nuevas partidas. Al hacer clic, se abrirá un modal como el de la *Figura 20*, con un formulario para crear la partida. Todos los campos son obligatorios, excepto los de “Respuesta”, que se explicarán más adelante. El campo *Título* sirve para identificar más fácilmente la partida entre los usuarios, y poder buscarlo en el cuadro de búsqueda anteriormente explicado. El campo *Tiempo* define la

duración de la partida en minutos. Es el tiempo que tienen los jugadores (alumnos) para contestar todas las preguntas. El campo *Jugadores* permite al creador definir el número máximo de jugadores que puede tener la partida. La opción *Autoinicio* sirve para empezar una partida automáticamente cuando el número de jugadores que se han unido sea igual al número de jugadores máximos definidos en el campo *Jugadores*. De esta forma no es necesario que el creador inicie la partida de forma manual. Finalmente, está el campo *Pregunta*, que sirve para añadir preguntas a la partida. Es obligatorio definir al menos una pregunta para crear la partida.

En la sección *Preguntas creadas* se muestran todas las preguntas que se han añadido a la partida. Se les puede asignar una respuesta de forma optativa. En caso de definirla, al finalizar la partida, se corregirá automáticamente utilizando el algoritmo de la distancia de *Levenshtein* [23], que compara dos cadenas de caracteres para ver la similitud entre ambas. En este caso, compararía la respuesta del jugador y la respuesta definida por el creador. Más adelante se explicará este proceso.

The image shows a web interface for creating a game. At the top, there is a navigation bar with a search icon and the text 'Buscar partida.', a red button 'Crear partida', and links for 'Mis partidas' and 'Partidas disponibles'. The user's name 'Juanan' is visible in the top right. The main content area is titled 'Crear Partida' and contains the following form elements:

- Titulo:** A text input field containing 'Programación'.
- Tiempo (minutos):** A numeric input field containing '30'.
- Jugadores:** A numeric input field containing '4'.
- Autoinicio:** A checkbox that is checked with a green checkmark.
- Pregunta:** A text input field with the placeholder 'Añadir pregunta' and a blue plus icon to its right.

Below the form is a section titled 'Preguntas creadas' which lists three questions with their respective answer fields and 'Eliminar' buttons:

- Question 1:** '¿Qué lenguaje es orientado a objetos?'
Answer: 'Respuesta (Optativa)'
- Question 2:** '¿Qué es un condicional?'
Answer: 'Respuesta (Optativa)'
- Question 3:** '¿Cuántos bytes ocupa un "char"?'
Answer: '1' (with a clear 'x' button)

Figura 19. Modal para crear nueva partida

Una vez creada la partida, se redirige a la pantalla de espera de la *Figura 21*. Se muestra información sobre la partida, y permite iniciar manualmente la partida o eliminarla. Para iniciar la partida manualmente, es necesario que haya al menos algún jugador. En caso de que se haya seleccionado la opción *Autoinicio* al crear la partida, ésta empezará en cuanto alcance el número máximo de jugadores.

La otra opción que tiene el creador de la partida es la de eliminarla con el botón rojo. Es posible borrarla aunque haya jugadores unidos, ya que se notificará a éstos de que la partida ha sido borrada. No se podrá borrar una vez que haya iniciado la partida.



Figura 20. Pantalla de espera (Vista del creador).

Cuando se vayan uniendo jugadores, la opción *Iniciar partida* estará disponible y podrá empezar la partida. Se puede ir viendo en tiempo real los usuarios que se unen. Los jugadores tienen una vista prácticamente igual a la de la *Figura 22*, con la diferencia de que solo les aparece la opción *Abandonar partida*. Si el jugador selecciona este botón, saldrá de la partida.

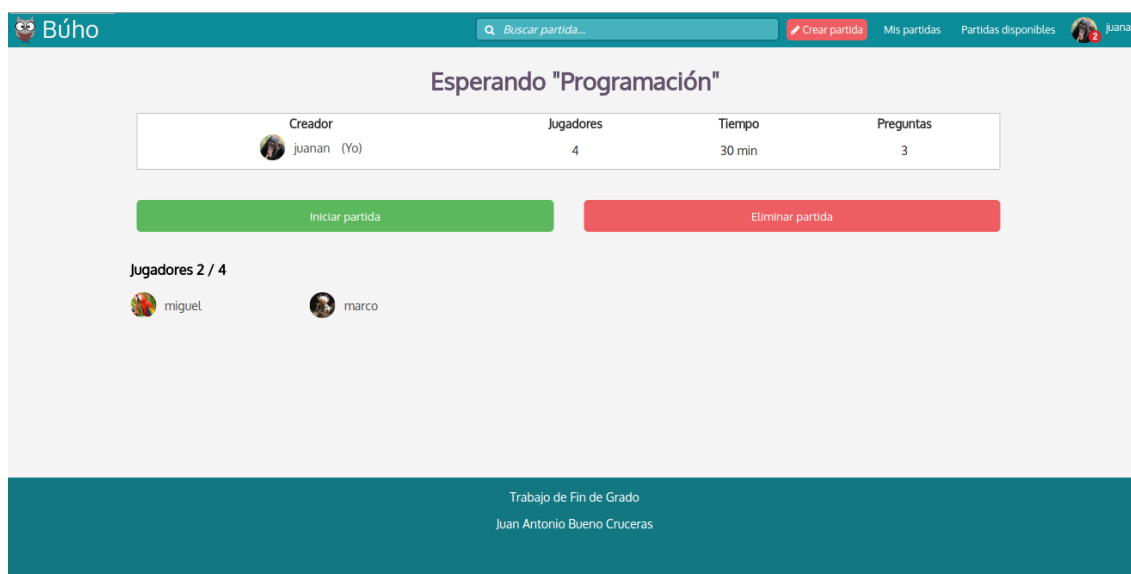


Figura 21. Usuarios unidos a la partida en espera (Vista del creador).

Al iniciar la partida, el creador y los jugadores tendrán vistas diferentes. Por un lado, los jugadores tendrán una vista como el de la *Figura 23*. En ella se muestra en la parte superior el título de la partida y el tiempo restante (en segundos). Más abajo, están las preguntas que forman la partida, junto a un campo que permite responderlas. Los jugadores pueden responder tantas veces como quiera, pero solo contabilizará la última respuesta. En caso de que la pregunta haya sido contestada, aparecerá un campo debajo de la pregunta que pone *Mi respuesta*, con la última respuesta enviada.

Por último, el jugador puede abandonar una partida que ha sido iniciada. Una vez abandonada, no puede volver a unirse ni cambiar las respuestas. Las respuestas que haya realizado hasta antes de haber abandonado se contabilizarán, pero cuando se muestren los resultados, aparecerá que el jugador abandonó la partida antes de que finalizase.



Figura 22. Jugar partida (vista jugador)

La vista de las partidas iniciadas para el creador es ligeramente diferente al de los jugadores. Se puede ver en la *Figura 24* Puede ver el tiempo que le queda a la partida o el título, al igual que en la vista de los jugadores. La diferencia está en que puede añadir nuevas preguntas en tiempo real. Estas preguntas no pueden asignárseles una respuesta, por lo que no se corregirán automáticamente. Una vez añadida, se enviará una notificación a los jugadores que estén unidos a la partida.

Debajo de este campo, aparecen las preguntas que forman la partida. No pueden ser editadas. Debajo de cada pregunta, se muestran las respuestas de los usuarios, ordenadas por tiempo de respuesta (aún no corrige si es correcta o incorrecta). Al lado de la respuesta aparece el tiempo que han tardado en responder, en formato MM:SS.



Figura 23. Jugar partida (vista creador)

Al llegar a su fin la partida, se les informará a los jugadores que no pueden añadir más respuestas mediante un mensaje. El creador de la partida será redirigido a una vista para seleccionar las respuestas correctas, como muestra la *Figura 25*. Hay preguntas que se corregirán automáticamente, utilizando el algoritmo de Levenshtein. Para tomar una respuesta como correcta, el nivel de similitud entre la respuesta del creador y la del jugador han de ser mayor o igual que 0.85 sobre 1.00. En caso de que hayan dos respuestas con misma puntuación, y que superen este límite, se escogerá la del usuario que antes respondió. Aún así, el creador puede seleccionar manualmente la respuesta que crea más conveniente, aunque este haya sido seleccionada por el algoritmo de Levenshtein. En caso de que ningún jugador haya acertado, el creador ha de introducir la respuesta en el campo de texto que aparece debajo de cada pregunta. En este campo se muestra la respuesta que se definió por defecto, en caso de que exista.

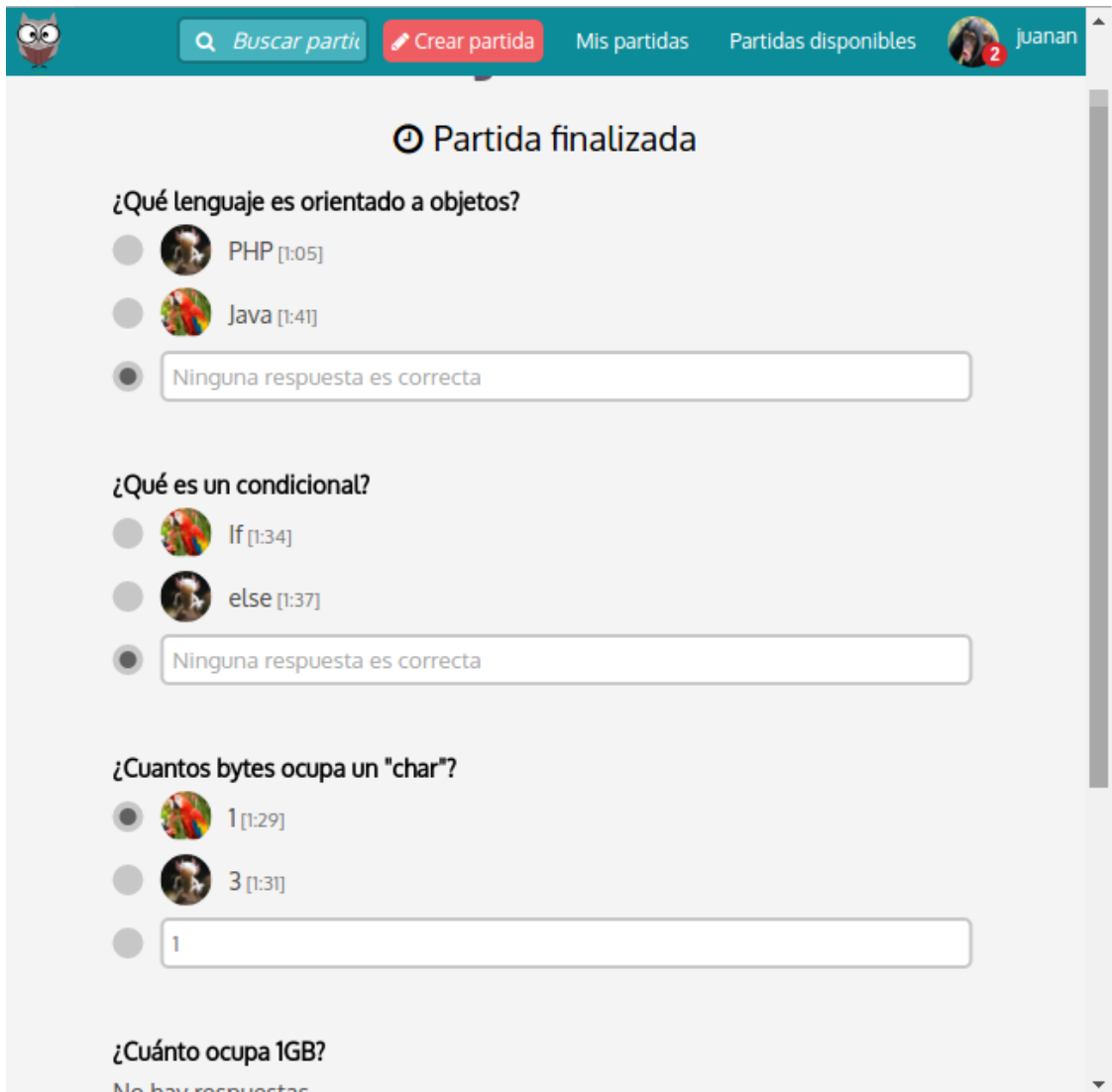




Figura 24. Seleccionar respuestas correctas de la partida finalizada.

Una vez solucionada la partida, tanto el creador como los jugadores pueden ver los resultados como en la *Figura 26*. La parte superior muestra información sobre la partida, y permite descargar los resultados en formato CSV. El creador podrá descargar todo el historial de los jugadores, mientras que los jugadores solo podrán descargar el fichero con la última respuesta de cada jugador.

Más abajo muestra el apartado *Jugadores*, que es un ranking ordenado por el número de respuestas correctas. Cada trofeo indica el número de preguntas ganadas. También se muestran los usuarios que abandonaron la partida, aunque se mostrarían los trofeos en caso de que haya ganado alguna pregunta.




Finalmente, están las preguntas que componen la partida, junto a las respuestas de cada jugador y la respuesta ganador (en caso de que exista). Justo debajo de cada pregunta, aparece la respuesta por defecto si se definió, o la respuesta ganadora del jugador. Por ejemplo, en la pregunta *¿Qué es un condicional?*, aparece la respuesta *"If-else"*, que fue la respuesta definida por el creador, pero la respuesta que se da cómo correcta es la del primer usuario, en este caso, la respuesta *"If"*.


Buscar partida
Crear partida
Mis partidas
Partidas disponibles
juanar 

Resultado de "Programación"



Creador	Jugadores	Tiempo	Preguntas	Descargar
juanar (Yo)	2 / 4	30 min	4	Descargar resultados

Jugadores



-  miguel 
-  marco (Abandono a 3:04)

Preguntas y Respuestas



P : ¿Qué lenguaje es orientado a objetos?
R : Java

-  marco (1:05): PHP
-  miguel (1:41): **Java** (Ganador)

P : ¿Qué es un condicional?
R : if-else

-  miguel (1:34): **If** (Ganador)
-  marco (1:37): else

P : ¿Cuántos bytes ocupa un "char"?
R : 1

-  miguel (1:29): **1** (Ganador)
-  marco (1:31): 3

P : ¿Cuánto ocupa 1GB?
R : 1024MB (Sin ganador)

Figura 25. Resultados de la partida

Volviendo a la cabecera, la siguiente opción que aparece después del botón para crear partidas es *Mis partidas*. En esta sección se muestran las partidas a las que ha jugado y las partidas que ha creado el usuario, dependiendo de la pestaña superior que tenga abierta del cuadro que aparece. En la *Figura 27* se muestran las partidas en las que ha jugado el usuario. Se pueden filtrar las partidas a mostrar, pudiendo visualizarlas todas, las que han acabado o las que están en proceso. Utiliza diferentes colores para que sea más fácil visualizar el estado de la partida.



Figura 26. Mis partidas jugadas

Si se selecciona la pestaña *Creadas*, se pueden ver las partidas que se hayan creado. El funcionamiento es idéntico que para las para las partidas jugadas, pudiendo filtrar por partidas acabadas, iniciadas o mostrarlas todas. La *Figura 28* muestra esta vista de las partidas creadas.

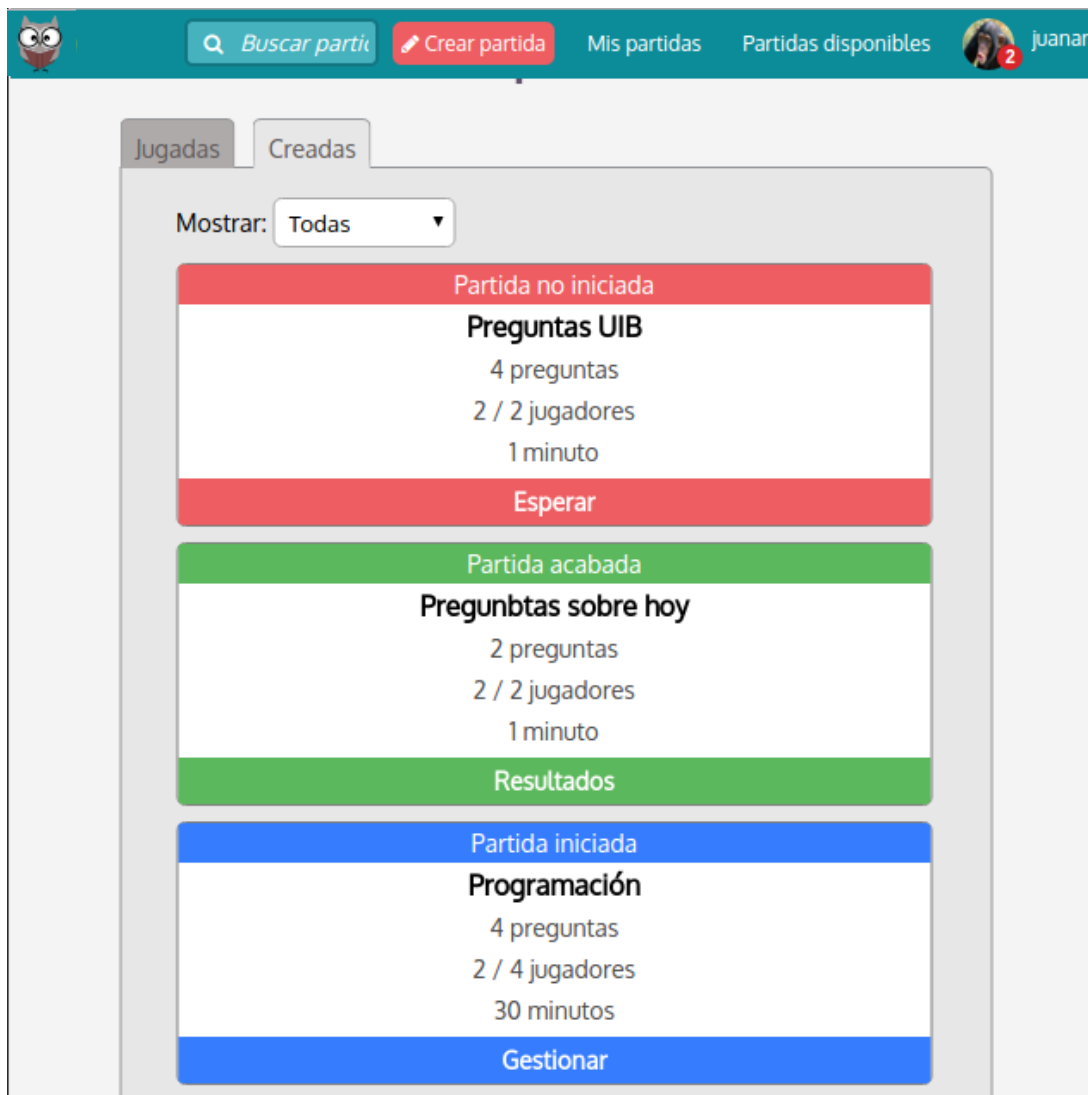


Figura 27. Mis partidas creadas

La siguiente opción de la cabecera es *Partida disponibles*. Si la seleccionamos, nos mostrará todas las partidas a las que un jugador puede unirse. Es la misma vista que se mostró en la *Figura 19*. Muestra las partidas por orden de fecha de creación descendente. Cada partida tiene información sobre el número de jugadores que hay unidos y el máximo, el título de la partida, el número de preguntas o el creador de la partida.

Por último, está el nombre de usuario con el que hemos iniciado sesión junto a la foto de perfil. Al seleccionarla, abre un menú que permite elegir entre editar el perfil, mostrar las notificaciones o mostrar las estadísticas. Las notificaciones se indican en el círculo rojo que aparece junto a la foto de perfil de la cabecera. Si queremos ver en más detalles las notificaciones, seleccionamos la opción *Notificaciones* que aparece al hacer clic en el nombre de usuario. Se abrirá una ventana como la que se muestra en la *Figura 29*. Hay 5 tipos de notificaciones: para indicar que una partida ha iniciado o ha acabado, para avisar que se ha añadido una nueva pregunta a una partida en curso, para notificar que se ha publicado el resultado de la partida y para informar al creador de la partida que ya hay el número máximo de usuarios. Para que estas notificaciones

sean más visuales, se les acompaña de un icono, junto a una pequeña descripción.

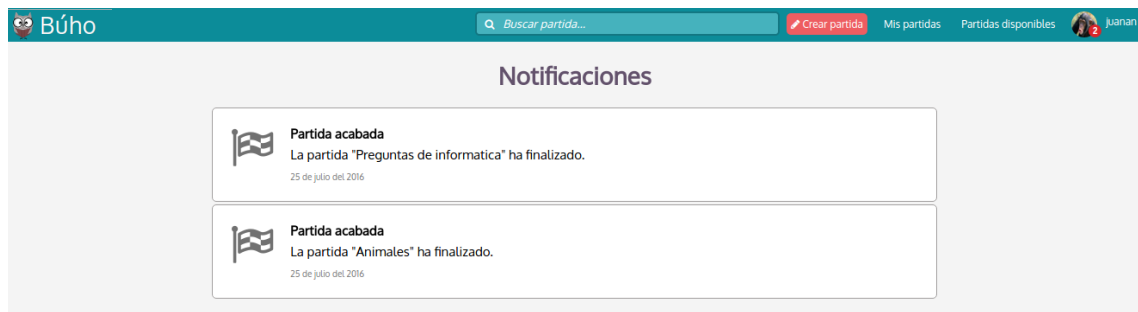


Figura 28. Vista de notificaciones.

Otra opción que aparece en el menú desplegable es la de poder editar el perfil de usuario. Se muestra en la *Figura 30*. Los campos que se pueden cambiar son el nombre, apellidos, fecha de nacimiento, contraseña y foto de perfil. Para cambiar la contraseña es necesario que ambas sean iguales.

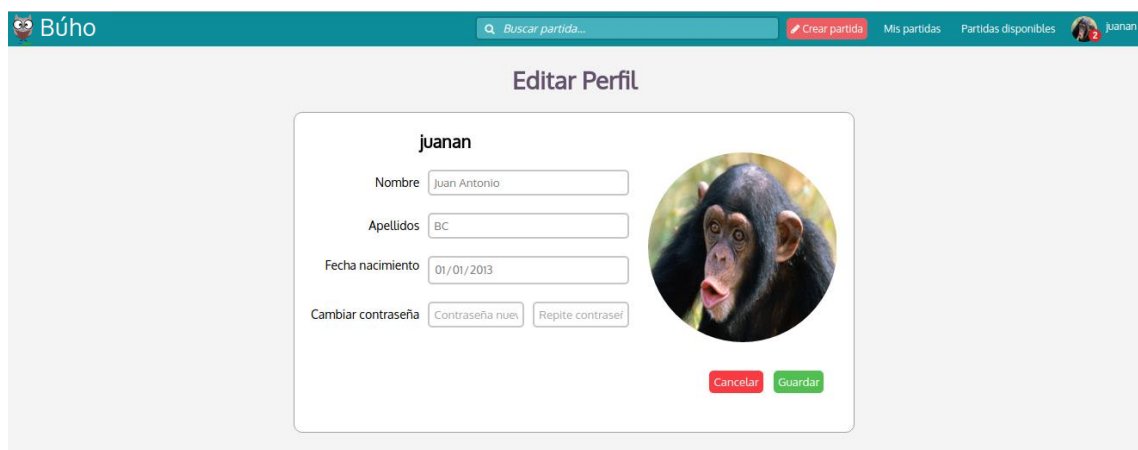


Figura 29. Editar perfil de usuario

La sección de *estadísticas* permite al usuario tener una vista resumida de su actividad en la última semana, mes o año, dependiendo del rango de tiempo que elija. Consta de 3 partes. La primera son 4 recuadros de diferentes colores que muestra el número total de cada estadística en el rango seleccionado. En la *Figura 31* se muestra la cantidad total de partidas jugadas, abandonadas y creadas en el último mes, además del número de preguntas que ha sido seleccionado como respuesta correcta (victorias).



Figura 30. Estadísticas generales del último mes

La siguiente parte de las estadísticas es un gráfico que muestra el día en el que se registró la actividad. Para ver la evolución de las victorias que se han tenido, se representa con una línea de color verde.

La agrupación de barras roja y azul representa la cantidad total de partidas que ha jugado ese día el usuario. La parte roja indica las partidas en las que el jugador abandonó la partida, mientras que las de color azul son las partidas que ha acabado. Justo al lado aparece (en caso de que exista) una barra de color amarillo que muestra el número de partidas que creó ese día.

Para ver más detallado la información de cada barra o punto del gráfico, basta con situar el cursor encima del objeto elegido, y se mostrará el valor y su fecha. La *Figura 32* muestra el gráfico explicado.

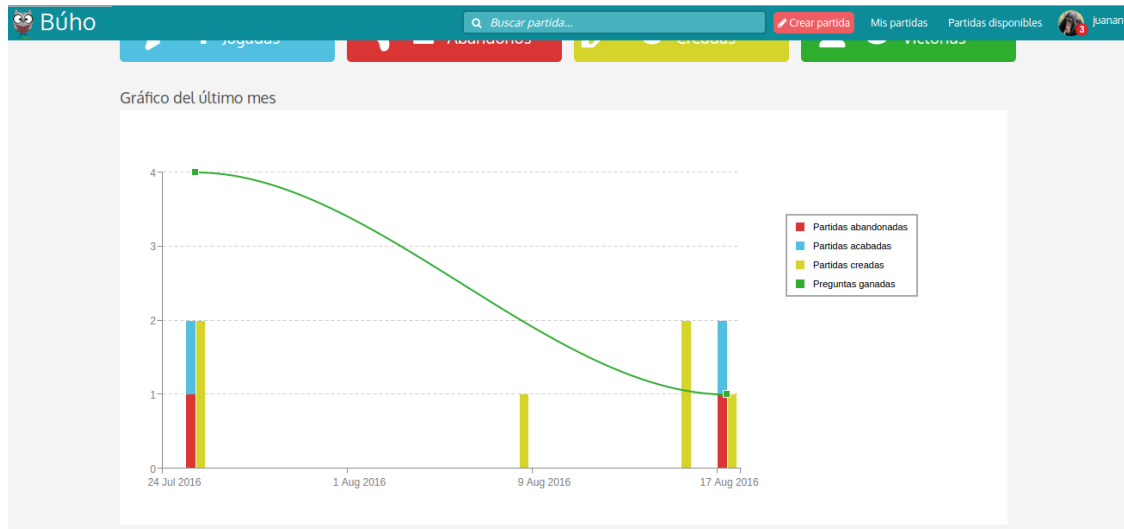


Figura 31. Gráfico del último mes

Por último, en la sección de estadísticas aparecen 2 tablas que muestran información resumida de cada partida jugada y creada. Muestra el título de la partida, la fecha, el número de victorias o el estado de una partida creada. Además, permite visitar la partida haciendo clic en el botón “Ver” de color azul. Se muestra en la *Figura 33*.

Partidas jugadas					
Nombre	Fecha inicio	Victorias	Abandonada	Visitar	
Animales	25-07-2016	2	SI	Ver	
Preguntas de informatica	25-07-2016	2	NO	Ver	
Sistemas Operativos	17-08-2016	1	NO	Ver	
Preguntas de gimnasia	17-08-2016	0	SI	Ver	

Partidas creadas					
Nombre	Fecha creada	Iniciada	Acabada	Resuelta	Visitar
Preguntas UIB	25-07-2016	NO	NO	SI	Ver
Preguntas del tema 3	25-07-2016	SI	SI	SI	Ver
Programación	08-08-2016	SI	NO	SI	Ver
Exámen final	15-08-2016	NO	NO	NO	Ver
Preguntas de historia	15-08-2016	NO	NO	NO	Ver
Preguntas de biología	17-08-2016	NO	NO	NO	Ver

Figura 32. Tablas con estadísticas de las partidas creadas y jugadas el último mes

BIBLIOGRAFÍA

- [1] Ley de Protección de Datos de Carácter Personal, <https://www.boe.es/buscar/act.php?id=BOE-A-1999-23750>
- [2] Ley Orgánica de Protección de Datos de Carácter Personal (España), [https://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_\(Espa%C3%B1a\)](https://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_(Espa%C3%B1a))
- [3] *Javier*, “NOSQL vs SQL. Diferencias y cuando elegir cada una”, 18 de noviembre de 2015, <http://blog.pandorafms.org/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/>
- [4] Single Page Application, https://en.wikipedia.org/wiki/Single-page_application
- [5] Stack MEAN, mean.io
- [6] ExpressJS, <http://expressjs.com/>
- [7] *D. Tabares*, “La actualidad de las TIC”, 19 de junio de 2012, <http://es.slideshare.net/duvan890131/la-actualidad-de-las-tic>
- [8] *M. Ruiz*, “Las TIC’s en la Actualidad”, 19 de octubre de 2011, <http://diarium.usal.es/mariaruiz/2011/10/19/las-tic%C2%B4s-en-la-educacion/>
- [9] Event-driven programming, https://en.wikipedia.org/wiki/Event-driven_programming
- [10] Cambio de contexto, https://es.wikipedia.org/wiki/Cambio_de_contexto
- [11] Node (Aspectos técnicos), https://es.wikipedia.org/wiki/Node.js#Aspectos_t.C3.A9cnicos
- [12] Socket.io, <http://socket.io/>
- [13] List of file signatures, https://en.wikipedia.org/wiki/List_of_file_signatures
- [14] *A. Felipe García*, “Los beneficios de la Tecnología en la educación”, 8 de abril de 2015, <http://www.labrechadigital.org/labrecha/Articulos/los-beneficios-de-la-tecnologia-en-la-educacion.html>
- [15] Desarrollo en cascada, https://es.wikipedia.org/wiki/Desarrollo_en_cascada
- [16] Moodle, <https://es.wikipedia.org/wiki/Moodle>
- [17] *J. Ferriman*, “7 awesome advantages of ELearning”, 10 de diciembre de 2016, <http://www.learndash.com/7-awesome-advantages-of-elearning/>
- [18] *C. Pappas*, “7 Tips For eLearning Professionals To Enhance Knowledge Retention”, 19 de octubre de 2014, <https://elearningindustry.com/7-tips-elearning-professionals-enhance-knowledge-retention>
- [19] Socrative, www.socrative.com
- [20] Socket de Internet, https://es.wikipedia.org/wiki/Socket_de_Internet
- [21] Trivial Pursuit, https://es.wikipedia.org/wiki/Trivial_Pursuit
- [22] Atriviate, <https://play.google.com/store/apps/details?id=aul.irm.triviados&hl=es>
- [23] Distancia de Levenshtein, https://es.wikipedia.org/wiki/Distancia_de_Levenshtein
- [24] List of signatures, https://en.wikipedia.org/wiki/List_of_file_signatures
- [25] MongoDB (Tutorial – Install MongoDB), <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>
- [26] Instalación NodeJS, <https://nodejs.org/en/download/package-manager/>
- [27] Lenguaje unificado de modelado, https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado
- [28] Webdriver, <http://webdriver.io/>
- [29] ExpressJS - Routing, <http://expressjs.com/es/guide/routing.html>