ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN
INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PROYECTO FIN DE CARRERA

Triviales: Investigación sobre HTML5 y CSS3.

Autor: Luis Alonso Blázquez
Tutor: Gregorio Robles Martínez

Curso académico: 2012/2013

Proyecto Fin de Carrera

Triviales: Investigación sobre HTML5 y CSS3.

Autor: Luis Alonso Blázquez
Tutor: Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2013, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2013

"The true test of a man's character is what he does when no one is watching."
- John Wooden

Agradecimientos

Resumen

La manera de consumir contenidos en internet ha cambiado en los últimos años. Inicialmente se utilizaban contenidos planos, webs estáticas y, por qué no decirlo, feas.

Con la mejora de las redes de comunicación esta tendencia ha cambiado radicalmente. A día de hoy, queremos contenidos bonitos, en páginas agradables y con temas de lo más variado. Es por esto por lo que ha cambiado la manera de hacer páginas web. Ya no son meros portales de información sobre una empresa, sino que además esa empresa puede querer tener un foro de soporte a sus usuarios, una galería de imágenes que muestre cómo son sus oficinas y unos vídeos corporativos. Todo dentro de la misma página y con un objetivo claro: atraer clientes.

Por todo ello, he decidido hacer mi proyecto final de carrera sobre dos tecnologías que están en auge: HTML5 y CSS3. Debido a que estas tecnologías no están asentadas del todo y que los navegadores actuales no han implementado los estándares de HTML5, no he podido obtener todo el resultado deseado, pero sí que he aprendido que ya nadie utiliza páginas estáticas, las animaciones no son imágenes GIF sino un fragmento de código CSS3 y que los vídeos ya no son un quebradero de cabeza, son una mera etiqueta más del árbol HTML.

En este proyecto me he centrado en cuatro elementos: la etiqueta vídeo, la etiqueta canvas, las nuevas etiquetas de división de la página y las animaciones con CSS3. La primera porque es la que más puede atraer a un *frontend developer*, evitarse el dolor de cabeza de los distintos requerimientos de cada navegador con sus vídeos. La segunda porque nos da la capacidad de hacer diseños en tiempo de ejecución de nuestras aplicaciones, por ejemplo, podemos ir pintando los «rombitos» de nuestro Triviales a medida que se van consiguiendo con un código Javascript. La penúltima por ver cómo cambiaba la dificultad con respecto a

otros trabajos y prácticas de la formación de la página web, y la última porque me parecía una característica muy útil de CSS3, y porque queda muy vistoso.

Índice General

Índice de Figuras

Parte I

Introducción

Capítulo 1

Introducción

En estos capítulos se realizá una breve introducción de todas las tecnologías utilizadas, explicando sus conceptos básicos y parte de su historia. Además, se aclaran los significados de las siglas que se van a utilizar a lo largo de toda la memoria.

Para concluir la introducción, explicaré alguna de las características de Python, el lenguaje utilizado, y de Django, el *framework* que he usado para el desarrollo.

Capítulo 2

HTML como lenguaje web

«HyperText Markup Language» («lenguaje de marcado hipertextual») es un lenguaje basado en etiquetas.

Fue utilizado por primera vez a comienzos de los años 90 y rápidamente se extendió para el desarrollo de páginas web y sin embargo se considera «recomendado» solamente a partir de la versión 4.0 por la W3C (*World Wide Web Consortium*).

2.1. HTML5

Figura 2.1: Logo de HTML5

HTML5 es la quinta revisión importante del lenguaje HTML. La W3C comenzó a trabajar en ello en torno al año 2006, como consecuencia del «caos» que había entonces para la creación de las páginas web con varios modelos usándose al mismo tiempo (XHTML 1.0, DOM2 HTML y HTML4).

Previa a la entrada de la W3C al proyecto, Apple, Mozzilla y Opera ya estaban trabajando en algo parecido. Su proyecto se denominó bajo las siglas WHATWG

(HyperText Application Technology Working Group) y sólo pasó a denominarse HTML5 una vez entrada la W3C. Tras una serie de conversaciones, las tres empresas antes citadas cedieron a la W3C la especificación que fue posteriormente publicada bajo licencia W3C y continuaron ligadas al proyecto.

Después de una serie de conflictos acerca de los objetivos del proyecto, la W3C quería marcar un punto y aparte con respecto a la evolución de HTML, y WHATWG quería continuar generando parches para el HTML actual. En 2011 se separan las dos organizaciones pero continúan trabajando en paralelo y ayudándose mutuamente. De este modo, la W3C creó un nuevo equipo de desarrollo para guiar HTML5 hacia donde ellos querían mientras que WHATWG continua sacando parches para HTML que aprovechan también en la W3C.
Entre las mejoras y cambios introducidos podemos destacar los siguientes:

- **Estructura del cuerpo del documento**: se han generado etiquetas para partes típicas de las páginas (header, footer, etc.).
- **Etiquetas para contenido multimedia específico**: se han añadido etiquetas para audio y vídeo, lo que permite facilitar el uso de tecnologías multimedia en la web.
- **Canvas**: es un nuevo componente que permite utilizar funcionalidades que estaban en *Flash* pero sin la necesidad de usar un plugin para ello. Acepta el dibujo por funciones y la interacción con el usuario.
- **Drag and drop**: nueva inclusión de esta característica que permite modificar o agregar elementos simplemente con arrastrarlos a la zona deseada. Muy utilizada en sistemas de ficheros en la nube.
- **Nuevos tipos de dato para formularios**: se han añadido tipos de datos muy utilizados como el *email*, la *URL* o el tipo *datetime*, incluyendo a su vez validadores para simplificar las tareas.
- **Nuevas APIs**: se han incluido para facilitar la interacción con el usuario. Como ejemplo cabe destacar la API de geoposicionamiento para localizar el dispositivo desde el navegador, o la API Storage, que permite el almacenamiento local de datos.
- **WebSockets**: permiten la comunicación bidireccional entre el cliente y el servidor. Con estos *sockets*, basados en los del lenguaje de programación *C*, se pueden enviar comunicaciones desde el servidor al cliente para, por ejemplo, actualizar la página o mandar información nueva como una notificación.

2.1.1. El elemento Canvas

Su traducción al castellano sería «lienzo» y probablemente sea el elemento más innovador si nos centramos en el diseño de la página web. Su concepción viene dada de la necesidad de evitar plugins en los navegadores para hacer que las páginas sean más accesibles a todo el mundo. Este elemento permite delimitar un espacio de la página donde, mediante *scripts*, podemos dibujar y renderizar imágenes.

A día de hoy, su aceptación en los navegadores no es total. Algunos como *Internet Explorer* sólo lo soportan a partir de la versión 9, aunque hay proyectos de plugins para versiones más antiguas de ese navegador que permiten utilizar dicho elemento. Por otro lado, el resto de navegadores más conocidos (*Chrome*, *Firefox*, *Opera* y *Safari*) sí que lo soportan para sus últimas versiones.

El elemento *Canvas* permite tanto el dibujo como la escritura de textos dentro de él. Utilizando el lenguaje *Javascript* para generar los *scripts* de pintado, podemos acceder a sus funciones muy fácilmente y hacer su contenido dinámico, variando según la necesidad que haya en cada momento.

Se puede encontrar más información sobre este componente en esta página [1]

2.1.2. El elemento Vídeo

Hasta la especificación de HTML5 no existía un estándar para presentar vídeos en una página web. La mayoría de las soluciones pasaban por utilizar algún *plugin* como *Flash*, lo que generaba el problema de adaptar el vídeo para los distintos navegadores.

Con este elemento vídeo dejamos atrás esos métodos y tenemos una nueva manera de incrustar vídeos en nuestra web. Basta con codificar el vídeo en *MP4* y *OGG* (cubriendo así todos los navegadores más populares) e incluirlos como atributos *source* del elemento. Para la inclusión de los controles de *Play*, *Pause* y *Stop*, basta con añadir un atributo *controls* dentro de la declaración del elemento.

2.1.3. Las nuevas etiquetas

Entendiendo por nuevas etiquetas aquellas que se utilizan ahora en HTLML5 como sustitutas de otros trozos de código comunes (cabeceras, pies de página, etc.). En este proyecto, por las características del mismo, he usado tres: *header*, *nav* y *footer*.

Usando estas nuevas marcas he definido la barra superior de la apliación como un *header* y la parte inferior, donde sitúo la posibilidad de mandar un email al administrador, como un footer con un *nav*. Pero primero voy a definirlos para mayor claridad.

- **Header**: Estructura de cabecera de la página. Típicamente se suelen incluir elementos como botones, links, textos resaltados e imágenes de cabecera.
- **Nav**: Etiqueta creada para definir partes de las páginas donde se acumulan muchos elementos de navegación (links en su gran parte).
- **Footer**: Se utiliza para definir el pie de página. En mi caso he utilizado la etiqueta para permitir un acceso directo a enviar un email para contactar con el administrador del juego.

Capítulo 3

CSS como lenguaje de estilos.

«Cascading Style Sheets» u hojas de estilo en cascada en su traducción literal al castellano. Este lenguaje de definición de estilos se convirtió en un estándar de la W3C en 1996 por la capacidad de separación de la apariencia de la página y su funcionalidad. Tras esto se expandió rápidamente y su uso permitió una mejora significativa en la calidad de las páginas web dado que inicialmente eran prácticamente solo texto y pasaron a ser mucho más llamativas a la vez que fáciles de cargar. Esto se consiguió gracias a que son ficheros de texto y por lo tanto son poco pesados para transmitir por la red, y todo el procesamiento se pasa al navegador.

CSS se utiliza para dar estilos a los diferentes elementos del documento *HTML*. Entre estos estilos caben los colores, los tamaños de las fuentes, las posiciones en pantalla y muchas cosas más. Para incluir las diferentes hojas de estilo, se utiliza la etiqueta *style* en el documento con la definición del tipo de documento que estamos enlazando.

3.1. CSS3

Figura 3.1: Logo de CSS3

Esta última versión del sistema de estilos más utilizado aún no se ha definido del todo. En la página oficial [2] se encuentra un *draft* de noviembre de 2012 aún en progreso. De todas maneras, se han definido ya unas cuantas características que he utilizado en el proyecto y que paso a enumerar:

- **Gradientes**: Para no tener la necesidad de generar imágenes con degradados de color para utilizar como fondo, se crea este tipo de deformación del color en el que solamente es necesario definir los dos colores a emplear y el porcentaje de ambos para que el navegador los pinte en una transición suave de uno a otro.

- **Transiciones**: Antiguamente hacía falta un código Javascript para poder tener efectos y animaciones sobre los componentes de las páginas. Ahora con la opción de transformación y transiciones de CSS3 podemos hacerlos mucho más fácil. Estas nuevas transiciones nos permiten cambiar la apariencia de los elementos sin necesidad de usar código Javascript y además están optimizados para cada navegador usando los diferentes recursos que ofrece cada uno (WebKit, Mozilla, etc.).

- **Selectores**: Nos permite definir estilo común para todos los tipos de etiqueta disponibles en HTML de manera conjunta. Podemos definir la apariencia de las tablas para que todas nos queden iguales, y todo con solo escribirlo una vez. Por ejemplo: *li:first-child* {*color:red;*} nos definiría que el primer hijo (primer elemento) de la lista será de color rojo.

Capítulo 4

Javascript

Figura 4.1: Logo de Javascript

Javascript es un lenguaje interpretado en el lado del cliente. Javascript nació con la necesidad de permitir a los autores de sitios web crear páginas que puedieran interactuar con los usuarios, ya que se necesitaba crear webs de mayor complejidad y el HTML sólo permitía crear páginas estáticas donde se podía mostrar textos con estilos.

Creado en 1995 de la unión entre Netscape y Sun Microsystems (creador de Java), inicialmente Microsoft intentó competir contra él creando su propia versión para el navegador Internet Explorer, pero con el paso de los años acabó aceptando el modelo propuesto por el resto de compañías.

4.1. AJAX

Figura 4.2: Logo de Ajax

«Asyncghronus Javascript And XML» es una manera de utilizar la comunicación cliente-servidor. Se basa en peticiones asíncronas desde el lado del cliente al servidor, normalmente usadas para cargar partes de la página o actualizar datos en el servidor.

Las técnicas de carga asíncrona de partes de las páginas web no es algo totalmente novedoso, lleva usándose más de 10 años. A mediados de los años 90 se empezó a incluir un elemento llamado *iframe* en las páginas que permitía cargar un archivo Javascript y así modificar la página que se estaba mostrando, consiguiendo un efecto parecido al AJAX.

Normalmente, en el caso de este proyecto por ejemplo, el lenguaje desde el que se lanza la petición AJAX es Javascript, mientras que el acceso a los datos se hace casi siempre mediante la tecnología XMLHttpRequest, que es un objeto disponible en los actuales navegadores basado en una estructura XML.

En este proyecto se han usado una serie de peticiones AJAX en estos casos:

- Carga de preguntas y posibles respuestas: Se ejecuta una petición *get* a una *URL* especificando el tipo de pregunta.
- Actualización de posición en el tablero: Una vez que se pierde el turno se envía la posición al servidor.
- Actualización de los «rombitos»: Cuando conseguimos responder correctamente una pregunta de este tipo se actualizan los datos en el servidor con el fin de denominar cuándo se acaba la partida.

Capítulo 5

Python

Figura 5.1: Logo de Python

Python es un lenguaje basado en scripts y orientado a objetos. Proporciona la simplicidad y facilidad de uso de un lenguaje interpretado, así como las más avanzadas herramientas de programación propias de lenguajes destinados al desarrollo de sistemas (como C o C++). Destaca por su facilidad de aprendizaje y su portabilidad entre las distintas plataformas, tan sólo condicionada a la presencia de un intérprete disponible.

Por destacar algunas de las propiedades que caracterizan a Python como lenguaje de alto nivel podriamos hablar de:

- **Tipado dinámico y fuertemente tipado**: No requiere que nos tomemos la molestia de declarar variables ya que reconoce su tipo desde que se asigna un valor por primera vez. Eso sí, durante el tiempo de vida de una variable ésta sólo puede pertenecer a un tipo.
- **Estructuras de datos flexibles y sencillas de usar**: como listas, diccionarios y strings, son parte intrínseca al lenguaje. Para procesar cada uno de estos tipos de objeto, incorpora un conjunto de operaciones que ahorrarán

tiempo y esfuerzo al implementar tareas de uso habitual como ordenaciones y búsquedas.

- **Administración automática de la gestión de memoria**: Se encarga de solicitar memoria en la creación de objetos y de liberarla cuando no van a volver a ser utilizados.
- **Facilidad en la construcción de sistemas de gran tamaño**: al incorporar herramientas como clases, módulos, y excepciones. Permite modularizar los componentes para una mayor claridad en el desarrollo.

Con Python es perfectamente viable el desarrollo de proyectos software de gran entidad, ejemplo de ello son el servidor de aplicaciones *Zope* y el sistema de intercambio de ficheros *BitTorrent*.
Otro aspecto que nos interesa particularmente es la incorporación, dentro de la completa librería que ofrece, de módulos que manejan estándares comunes en Internet (HTML, FTP, XML, HTTP, ...) y APIs para la comunicación con bases de datos (para gestores como PostgreSQL, Oracle o MySQL).

Vamos a hablar ahora del *framework* que he utilizado para el desarrollo de este proyecto: Django.

5.1. Django

Figura 5.2: Logo de Django

Django es un framework de código abierto escrito en Python. Este framework se creó inicialmente para facilitar el desarrollo de aplicaciones con una meta en mente: "Don't repeat yourself", es decir, no repitas cosas que ya has escrito anteriormente. Con esta idea se crea el sistema de plantillas para las vistas de las distintas páginas. Este sistema se basa en la inclusión de trozos de código HTML en otras páginas menos específicas permitiendo asi dar una apariencia común a los elementos básicos de la aplicación.

Otra cosa a destacar es el sistema de vistas y el tratamiento de las URLs. Django utiliza un fichero llamado urls.py en el que se definen qué vistas se van a

servir para qué URLs. De este modo nos permitiría incluso crear un fichero por vista para dejar bien claro qué es cada cosa y usarlo como una aclaración más para la documentación.

Además, en el caso de este proyecto, se ha usado una base de datos relacional aprovechándose de las facilidades que ofrece Django. En este framework las tablas de la base de datos se generan mediante modelos, que se definen en el fichero models.py, y en los atributos de dichos modelos se declaran las propiedades de cada columna de la tabla. Por defecto Django añade un campo ID por lo que no es necesario mantener una columna extra con el objetivo de localizar las entradas.

Para más información sobre el framework, esta es la página web del proyecto [3].

Capítulo 6

Tecnologías secundarias

Durante el desarrollo y puesta en producción de la aplicación, he querido utilizar además estas tecnologías.

6.1. Git

Figura 6.1: Logo de Git

`Git` es un sistema de control de versiones muy utilizado en el mundo empresarial junto con Subversion. En este proyecto me he servido de un repositorio en la plataforma GITHub, gratuita, para poder trabajar con una copia de seguridad del código.

Una de las cosas que, en mi opinión, hacen interesante a este tipo de sistemas, es la posibilidad de crear distintas ramas de desarrollo, permitiendo así un desarrollo en paralelo entre varias personas y reduciendo los tiempos de generación de versiones.

Otra cosa reseñable es la capacidad de `Git` de poder volver a *commits*, cambios en el código, anteriores para poder recuperar una cosa que antes funcionaba

y con los últimos cambios ha dejado de funcionar, evitando horas de búsqueda. En este sentido, GITHub dispone de un comparador de ficheros online en el que se pueden ver en verde las inclusiones de nuevas líneas y en rojo las eliminadas en esta última versión. La figura 6.2 muestra un ejemplo claro de cómo se observarían los ficheros de cara a una posible revisión de código.

Figura 6.2: Vista de líneas modificadas en git

La herramienta GITHub dispone además de un diagrama de ramas con sus respectivos *commits* etiquetados con el texto asociado al mismo y marcados como puntos en la rama correspondiente. En el caso de mi proyecto solamente he utilizado dos ramas: la master y la develop.

En la rama master tenía las versiones funcionales de la aplicación, aquellas con módulos enteros funcionando. Por otra parte, en la develop, tenía la evolución de los nuevos módulos que estaba implementando y aún no estaban testados. En mi repositorio [4] se puede encontrar y descargar el código fuente de la aplicación mediante un fork.

6.2. Servidor Apache

Figura 6.3: Logo de Apache

Es un servidor de código abierto que se puede utilizar en prácticamente todos los sistemas operativos actuales. Su uso está muy extendido entre la comunidad dada su facilidad de uso y la potencia de sus órdenes y restricciones.

Desde 1996 es el servidor HTTP más utilizado, pero no por ello está libre de críticas. La mayor crítica que se ha hecho a este servidor es la carencia de una interfaz gráfica que facilite su configuración, cosa en la que estoy de acuerdo, y yo incluyo el no tener un tutorial adecuado en su página web [5] teniendo que buscar la información en páginas externas y que no siempre están adaptadas a las últimas versiones del software.

Entre sus competidores podemos encontrar el IIS de Microsoft, pero este servidor, al ser privativo tiene un elevado coste que no ayuda a su expansión. Debido a la gratuidad de Apache, éste es el preferido no sólo por las grandes compañías (Google tiene su capa frontal en una versión personalizada de Apache), sino también por los pequeños proyectos personales.

Parte II

Objetivos

Capítulo 7

Objetivos

En esta sección repasaré los objetivos y metodologías utilizadas para el desarrollo del proyecto. Además, hablaré sobre los distintos requisitos que tenía al principio del proyecto y comentaré un poco qué me llevó a elegir esta clase de proyecto de fin de carrera.

7.1. Descripción del problema

Este proyecto puede tomarse como una manera de profundizar y adquirir conocimientos en las nuevas tecnologías de desarrollo de aplicaciones web. Dichas tecnologías son el presente y futuro cercano del desarrollo web, por lo que tener una base sólida de cara a entrar en el mercado laboral me parecía un objetivo primordial a la hora de elegir el tema del proyecto.

Uno de los problemas iniciales fue documentarme para empezar con la tecnología HTML5. Pensaba que iba a haber poca información dado que no consideraba que fuese una tecnología tan expandida, pero cual fue mi sorpresa cuando vi que no, que había cientos y cientos de páginas y recursos online donde aprender y sacar cosas para el proyecto.

Una vez localizadas las páginas desde las que partir, empecé a diseñar la estructura que tendría la página web: ¿qué pantallas iban a ser necesarias?, ¿cómo iban a ser las transiciones de una a otra? y ¿cómo sería la lógica de la creación de partidas? Cuando tuve esto claro, empecé a desarrollarlo siguiendo los siguientes criterios.

Capítulo 8

Requisitos

El proyecto deberá cumplir ciertos requisitos básicos:

- *Utilizar las nuevas etiquetas HTML5*: Una de las funcionalidades que más utilizaré en el futuro si me dedico a esto, con lo cual convenía tenerlas claras después de este proyecto.
- *Utilizar las nuevas funcionalidades de HTML5*: Tanto el canvas como la etiqueta de vídeo son elementos muy potentes y por ello quería probarlos en profundidad.
- *Utilizar animaciones de CSS3*: Probablemente sea lo que más problemas me ha dado, pero he de reconocer que son muy vistosas y ahorran mucho código JavaScript.
- *Definir una funcionalidad similar al juego Trivial*: Basándome en la jugabilidad conocida por todos quería tener una curva de aprendizaje muy baja para así poder sacar mejor resultado de los comentarios de las pruebas.
- *Ofrecer una experiencia de usuario atractiva*: Dado que la aplicación es un estudio sobre CSS3, quería que el proyecto tuviera una apariencia moderna y con sentido.

Parte III

Diseño e implementación

Capítulo 9

Modelo de desarrollo

Para el desarrollo de cualquier proyecto de software se realizan una serie de tareas entre la idea inicial y el producto final. Ese desarrollo sigue una determinada metodología o modelo de desarrollo, el cual establece el orden en el que se llevan a cabo las tareas en el proyecto y nos provee de requisitos de entrada y salida para cada una de las actividades.

El desarrollo de este proyecto se ha basado en un modelo de desarrollo en cascada. Se optó por este modelo ya que las condiciones de desarrollo así lo requerían puesto que no tenía tiempo para trabajar en varias cosas a la vez y me decidí por este modelo lineal aunque lo fui dividiendo en módulos para facilitar el desarrollo del mismo.

El modelo en cascada ofrece también la posibilidad de mantener un desarrollo claro y de ir probando cada parte independientemente. Con esto conseguimos que el proceso de depuración de la aplicación completa sea prácticamente nulo, gracias a la depuración de cada componente aislado. Esto nos permite, además, tener los distintos módulos separados y así poder reutilizarlos en futuras aplicaciones, reduciendo pues el trabajo futuro.

El conjunto de módulos en que dividí el proyecto para el modelo de desarollo escogido fueron los siguientes:

- **Creación del registro de usuarios**: Como parte muy importante del proyecto, quería hacer una gestión de usuarios casi propia. Para ello definí los datos a almacenar del usuario para posteriormente poder hacer la gestión del login-logout de la plataforma.

- **Creación del login de usuarios**: Era la continuación lógica a la parte anterior. Necesitaba saber quién entraba para la recuperación de las partidas.
- **Diseño de la base de datos**: Tuve que definir una base de datos adecuada al proyecto. Hice un primer diseño con tres tablas: Usarios, partidas, preguntas y rombitos. Posteriormente vi que la última tabla era innecesaria puesto que se podía utilizar un objeto *JSON* para almacenar el estado de cada usuario guardándolo en la tabla de partidas.
- **Creación de la base de las vistas**: Una vez hecha la lógica de la aplicación en el lado del servidor con el guardado y recuperación de partidas, empecé a diseñar el aspecto de la aplicación. Tras unos cuantos cambios de idea conseguí el header, footer y el fondo de la aplicación.
- **Diseño e implementación del tablero de juego**: Parte principal del juego. Fue la parte más larga debido a la dificultad de colocar cada casilla en el lugar correspondiente. Al final recurrí a posiciones absolutas fijadas con distancias en píxeles para poder hacerlo funcional en todos los tamaños de pantalla actuales.
- **Diseño e implementación de la lógica del juego**: Al ser una versión del conocido *Trivial Pursuit*, la lógica del juego no varía mucho, salvo que aquí, y para mayor claridad, el primero que consigue todos los rombitos es el ganador. Además decidí hacerlo en un pentágono para darle más velocidad a las partidas y con preguntas de conocimiento general, pero algunas más difíciles a mi parecer para que el juego no sea monótono.

Hay más pasos en el desarrollo de la aplicación, pero al ser más pequeños he preferido no ponerlos aquí. Se irán mencionando más adelante durante la explicación del diseño e implementación del proyecto.

Capítulo 10

Arquitectura general

En primer lugar planteé el proyecto como un servicio REST, aunque con la evolución del desarrollo vi que algunos verbos no se iban a utilizar. Con esta condición, supe que la arquitectura iba a ser la de un servidor que accede a la base de datos, y la de un cliente con bastante funcionalidad desarrollada en Javascript.De este modo, empecé a desarrollar el servidor.

El servidor es un servidor sobre el framework Django escrito en Python como ya he comentado. Este framework nos permite diferenciar muy bien los ficheros de lógica del servicio de los de apariencia. Además, nos ofrece una gestión de la base de datos que nos permite no tener más que crear los modelos (lo que serán las tablas posteriormente) y Django creará las tablas añadiendo un identificador a cada registro. Con esto nos quitamos el problema de la base de datos que en este caso decidí que estuviera implentada sobre *sqlite3* dado que ya había trabajado con este tipo de bases de datos anteriormente.

Para hacernos mejor a la idea de cómo está montado el sistema, en la figura 10.1 tenemos una recreación de un escenario con tres clientes.

Como consecuencia del desarrollo del proyectos siguiendo el modelo REST, los métodos asociados a cada url están recogidos en la figura 10.2.

Como podemos ver se han utilizado dos de los cuatro verbos disponibles siguiendo este planteamiento:

- **GET**: Para la petición de datos al backend o carga de pantallas.
- **POST**: Creación de nuevos usuarios o nuevas partidas y actualización de datos en el backend.

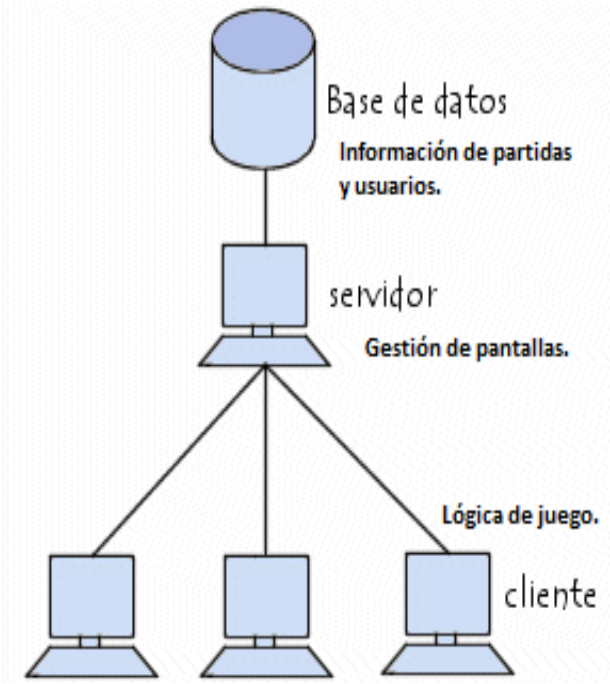


Figura 10.1: Esquema de la arquitectura general.

RECURSO	VERBO	ACCIÓN
home	GET	Devuelve la pantalla inicial.
Login	GET	Devuelve la pantalla de login
	POST	Intenta loguear al usuario en la aplicación
register	GET	Devuelve la pantalla de registro.
	POST	Registra un nuevo usuario en la base de datos.
newGame	GET	Devuelve la pantalla de creación de nuevas partidas.
	POST	Crea una nueva partida con los datos recibidos.
Logout	GET	Elimina los datos de sesión y devuelve al usuario a la pantalla inicial.
Games	GET	Devuelve el listado de partidas disponible para el usuario.
Actualgame	GET	Devuelve la partida seleccionada por el id.
Pregunta	GET	Dado un tipo de pregunta, devuelve una de las almacenadas en la base de datos.
cambiaTurno	POST	Cambia el turno de la partida solicitada.
Rombito	POST	Guarda en la base de datos que el usuario dado ha conseguido el rombito dado.
Fichas	GET	Pide el historial de rombitos conseguido por cada usuario para pintarlo.
Borrar	POST	Elimina la partida seleccionada.
Posición	POST	Actualiza la posición en la base de datos.
compruebaNombreDisponible	GET	Comprueba que el nombre de usuario esté disponible para el registro.
compruebaMailDisponible	GET	Comprueba que el email introducido no esté ya en uso por otro usuario.

Figura 10.2: Resumen de los recursos REST.

Capítulo 11

Pantallas

Hablaré ahora de los distintos escenarios que componen la plataforma definiendo su funcionamiento y las cosas que se pueden hacer en ellos. Además, comentaré qué métodos actúan y cómo se generan los distintos componentes de la misma.

11.1. Página principal

La imagen 11.1 es la que nos encontramos nada más acceder a la web. Está compuesta por un vídeo introductorio, de los botones para ir a las pantallas de registro y entrada en la plataforma, el *header* con la apariencia de todo el proyecto y un *footer* con la opción de contactar conmigo. Estos dos últimos componentes son comunes a toda la plataforma y se mantienen, con alguna modificación del *header*, en toda la web.

El vídeo situado a la izquierda de la página está incluido en la misma mediante la etiqueta Video de HTML5 sobre la que he hablado antes. Está codificado en *.mp4* y *webm* para reducir el tiempo de carga y permitir la visualización en Chrome y Firefox, y dispone de los controles por defecto que nos ofrece la etiqueta vídeo. Además, se reproduce de manera automática cuando cargamos la página.

El header y footer están definidos con sus correspondientes etiquetas de marcado con el mismo nombre. Usando CSS he colocado cada elemento en su sitio y con las plantillas de Django he podido centrarme solamente en el componente sobre el fondo pizarra dado que el otro componente está en la plantilla base por ser común a todas las pantallas.

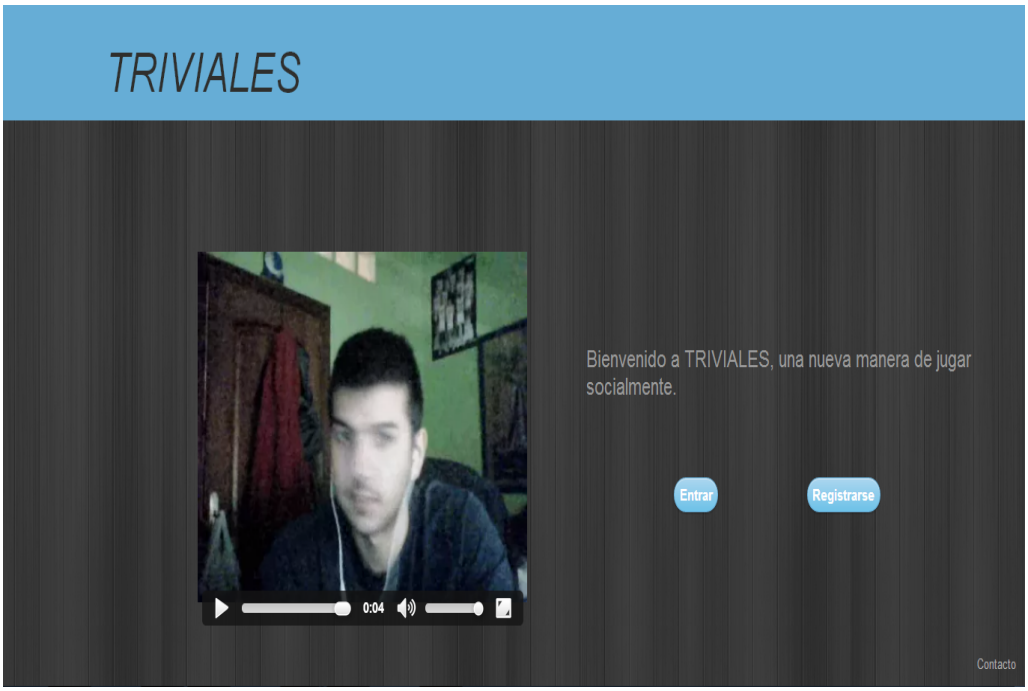


Figura 11.1: Apariencia de la página principal.

Cada uno de los botones tiene asociada una acción que llama al recurso *login* o bien al recurso *register* en función del botón que se haya pulsado, pero siempre con el verbo GET.

11.2. Login

En la figura 11.2 se encuentra la lógica de entrada en la aplicación. Tenemos un panel central que tiene una animación de rebote con CSS3 antes de quedarse estático a la espera de que el usuario introduzca sus datos para entrar en la aplicación.

Como se observa en la figura 11.2, hay dos campos para introducir nuestros datos: uno para el nombre de usuario elegido en el registro del tipo de datos name (predefinido de HTML5) y otro de tipo *password* también con especificaciones propias en HTML5. En este caso, el campo *password* automaticamente convierte los caracteres en puntos para dar privacidad a un dato secreto.

Podemos apreciar, además, dos botones en el recuadro, uno con una leyenda de volver y otro de entrar. Estos dos botones tienen la función de llamar a una función de javascript para retroceder en el historial una pantalla (caso del botón volver) y de comprobar que los datos son correctos con una petición AJAX al

Figura 11.2: Apariencia de la página de login.

recurso *login* con el verbo POST (caso del botón entrar).

En caso de que la validación de los datos haya sido incorrecta, aparecerá un mensaje sobre el *label* superior informando de que los datos no están almacenados en la base de datos como se muestra en la figura 11.3.

Esta actualización se hace desde el backend a la hora de procesar la petición. Si no es correcta, se incluye un mensaje de error en la plantilla antes de enviar el *HttpResponse* con la página de login otra vez.

11.3. Registro

Otra manera de acceder a la plataforma es mediante el registro de una nueva cuenta. Para esto son necesarios tres datos: nombre de usuario, email y contraseña que deberá introducirse dos veces para asegurar que se memoriza y no está equivocada. La apariencia está reflejada en la figura 11.4.

Al igual que en el caso del cuadro del login, se han utilizado tipos de datos específicos para los campos de recogida de datos. En este caso, se ha utilizado además de los mencionados anteriormente, el campo tipo *email*. Este campo lleva incluida una validación de forma nativa que evita que se introduzcan datos con

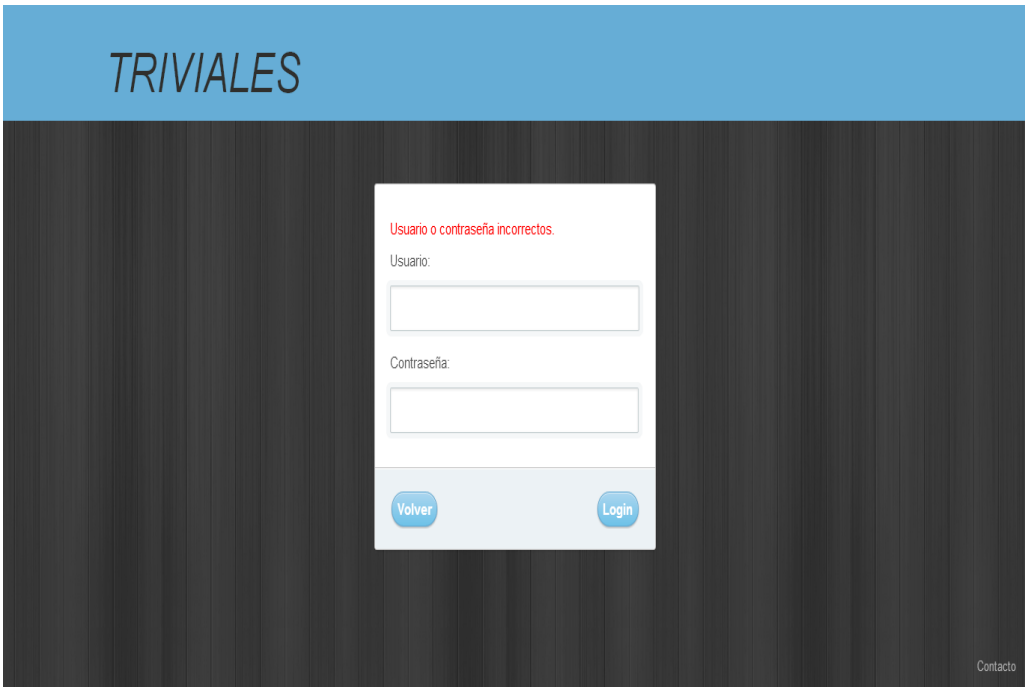


Figura 11.3: Apariencia de la página de login con mensaje de error.

formas que no correspondan a un email, es decir, el texto introducido tiene que contener una arroba, y acabar con un punto y dos o tres caracteres de extensión.

Cuando terminamos de escribir el nombre, se ejecuta una petición AJAX que consulta si ese nombre de usuario existe ya o no, dado que es una de las claves primarias de la tabla de la base de datos. Si está disponible, se pinta un tick verde y si está ocupado una equis roja como se ve en las figuras 11.5 y 11.6. Se hace lo mismo con el campo email, evitando así duplicidad de cuentas.

Una vez pulsamos el botón de registar, los datos introducidos se envían al servidor mediante un método POST y se crea el nuevo usuario, a continuación nos redirige a la pantalla de las partidas que tenemos actualmente.

11.4. Partidas

Representada en la figura 11.7, en esta vista obtendremos el listado de las partidas que tenemos en juego actualmente. Están divididas en partidas por jugar y partidas en las que tenemos que esperar a que juegue el otro jugador. Así mismo, al lado de cada partida hay un icono de una papelera para poder borrar la partida en todo momento.

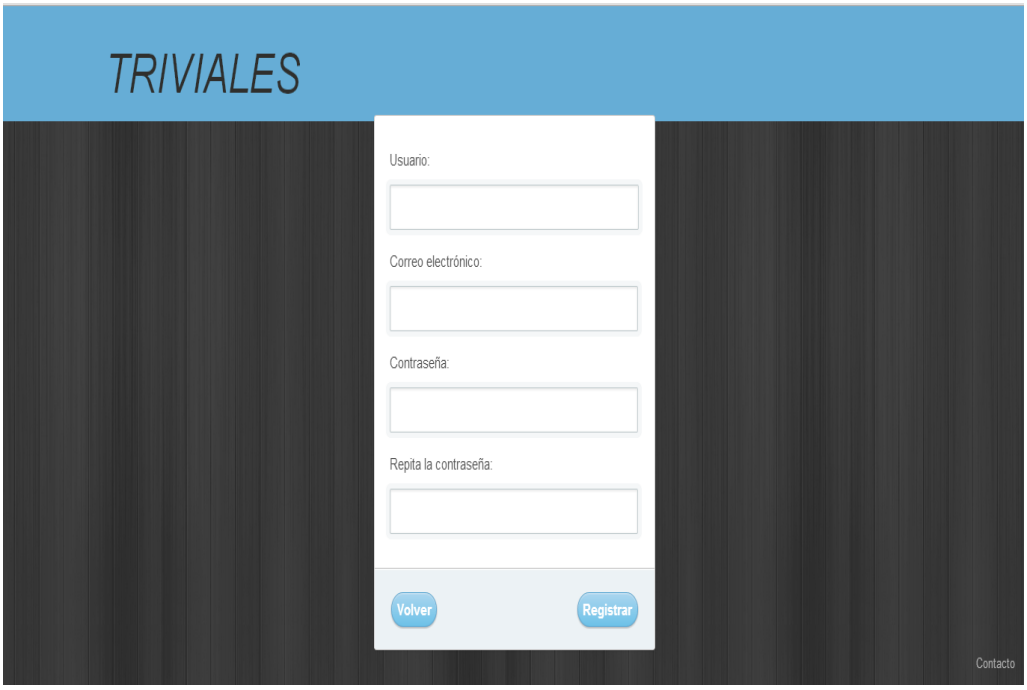


Figura 11.4: Apariencia de la página de registro.

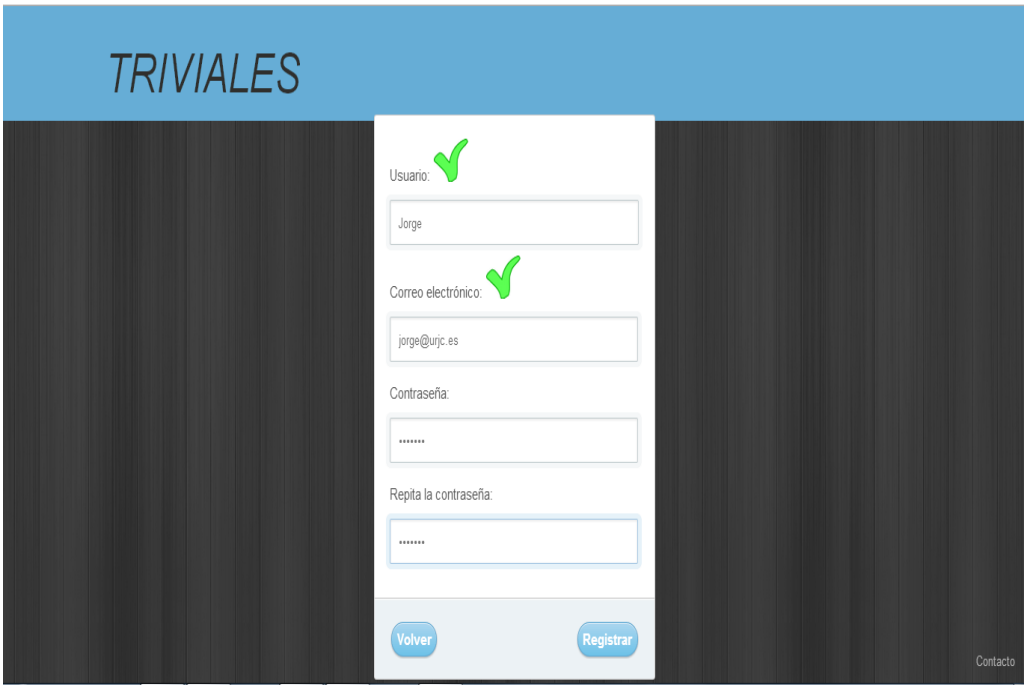


Figura 11.5: Página de registro válida.

Figura 11.6: Página de registro errónea.

Figura 11.7: Vista del listado de partidas.

Figura 11.8: Página de creación de partidas.

11.5. Crear partida

La apariencia de esta pantalla dedicada a la inicialización de partidas se muestra en la figura 11.8. Esta es la vista inicial, sin ninguna opción marcada. Como podemos apreciar, hay tres maneras de crear una nueva partida.

- **Encontrar por nombre de usuario**: Si se conoce el nombre de usuario del otro jugador, pinchando en este botón, se desplegará un cuadro editable donde se podrá escribir el mismo (figura 11.9) y se creará una partida con el otro jugador de manera inmediata.
- **Encontrar por correo electrónico**: Si queremos utilizar el correo electrónico, podemos emplearlo de la misma manera que en el caso anterior. Se ejecutará una consulta en base de datos usándose como clave el correo electrónico que está definido como atributo único de la tabla. En caso de no existir ese dato en ningún usuario, se mostrará un aviso en rojo (figura 11.10) de que no se ha podido crear la partida.
- **Partida con oponente aleatorio**: Esta opción creará una partida contra un usuario disponible en la base de datos elegido al azar mediante un proceso de generación de número aleatorio.

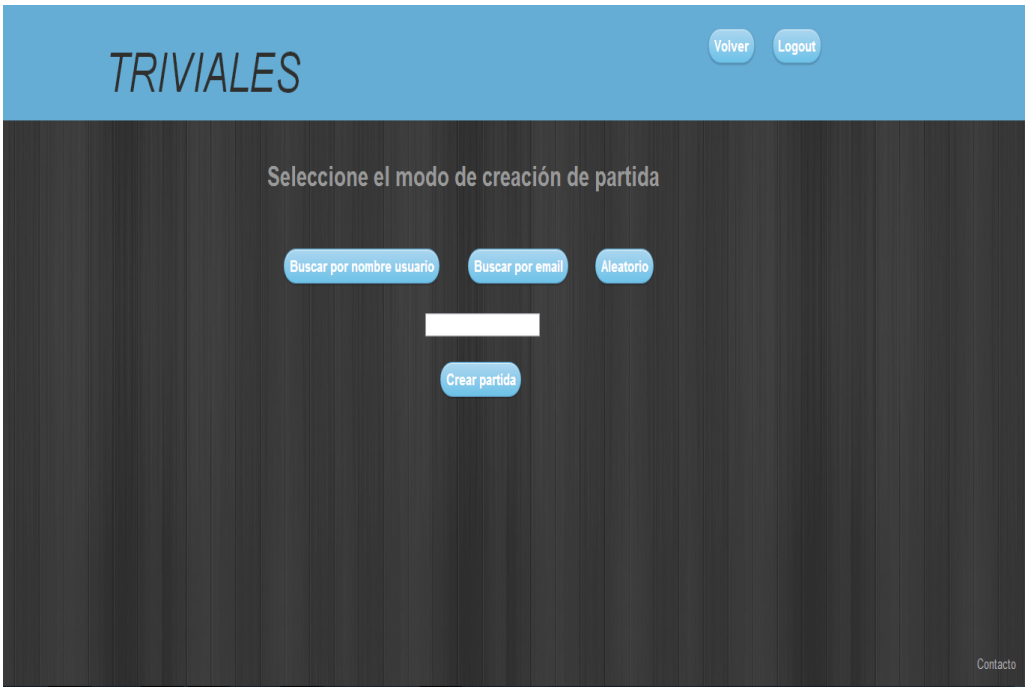


Figura 11.9: Creación de partida por nombre de usuario o email.

Figura 11.10: Error en la creación de partida por nombre de usuario o email.

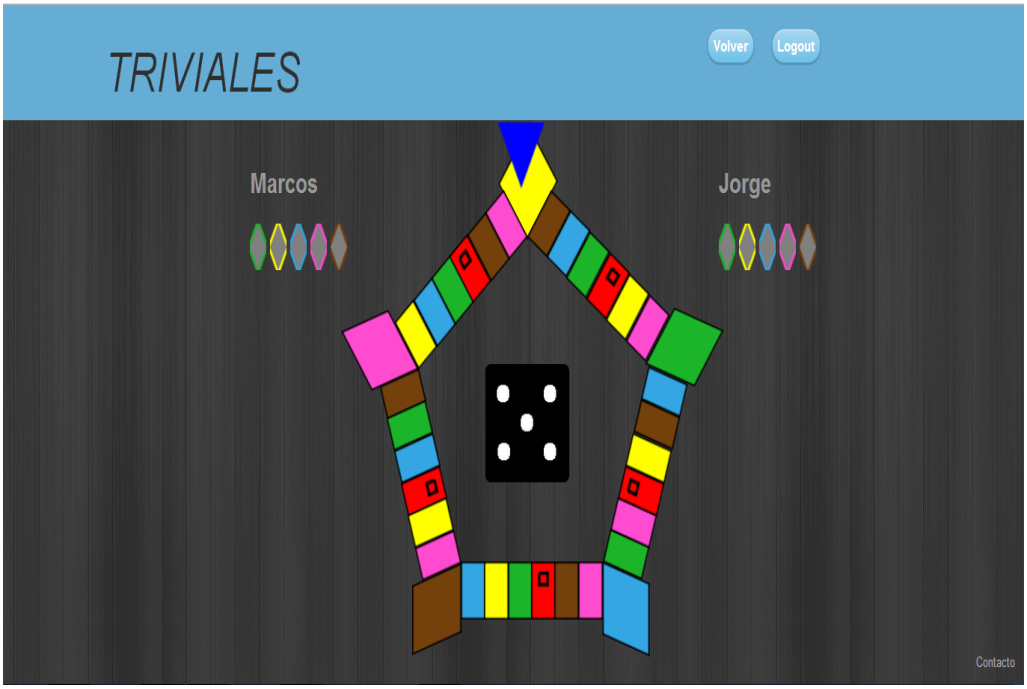


Figura 11.11: Visión de la partida con turno.

11.6. Pantalla de juego

En esta pantalla tenemos toda la lógica del juego en sí. Como se aprecia, si tenemos el turno aparecerá el dado central girando, que se parará cuando pulsemos sobre él. Como se puede apreciar en la figura 11.11, he decidido que el tablero sea pentagonal por varias razones, la principal es darle sencillez y rapidez al juego. Si tenemos muchos rombitos por conseguir, el juego puede llegar a ser pesado y nosotros podemos cansarnos de él.

Una vez que hacemos click en el dado, se pintan en blanco las posiciones posibles destino con el valor del dado que hemos obtenido como queda representado en la figura 11.12. Para movernos, hace falta otro click en una de las dos opciones disponibles marcadas en blanco.

Cuando hemos elegido la casilla a la que queremos desplazarnos, se ejecuta una petición AJAX a nuestro backend con el tipo de casilla que es, y se devuelve una pregunta de ese tipo que se muestra en pantalla como se observa en la figura 11.13.

Aquí cabe resaltar dos cosas. La primera es que el orden de las respuestas no será igual casi nunca, esto es debido a que he creado un algoritmo de aleatorización para la carga de las respuestas en la pantalla que evite que podamos

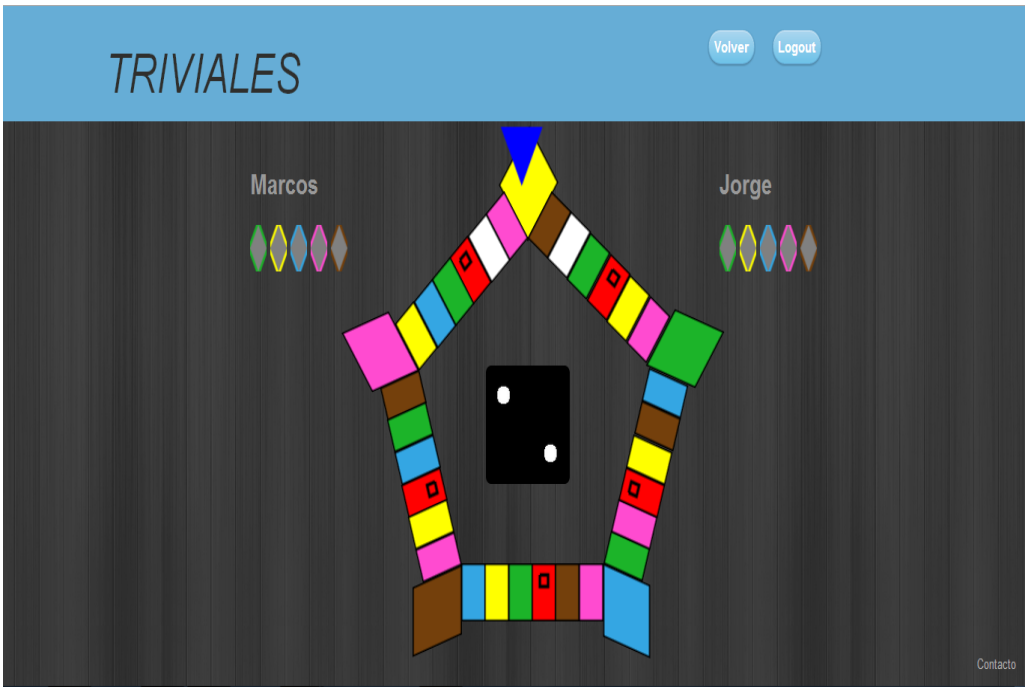


Figura 11.12: Visión de las casillas posibles.

Figura 11.13: Detalle de una posible pregunta.

Figura 11.14: Detalle de la alerta de felicitación por rombito.

aprendernos qué respuesta es la correcta sin saberlo realmente.

La otra cosa resaltable va en el mismo sentido. El botón de las respuesta se marcará en verde si es correcta y en rojo si, por el contrario, hemos fallado. En ningún momento se marcará la respuesta correcta en caso de fallo, esta decisión ha sido tomada así para evitar que nos la aprendamos en lugar de tenerla que buscar si queremos acertarla en la siguiente ocasión que nos encontremos con esta pregunta. Esta búsqueda nos dará lugar a una más que probable ampliación del conocimiento relacionado con el tema.

Si hemos acertado la pregunta, seguiremos jugando, pero si además hemos acertado una pregunta que nos da un rombito, nos aparecerá una alerta felicitándonos por el logro como el de la figura 11.14. Tendremos que aceptar esa alerta y podremos seguir jugando.

Si en cambio hemos respondido correctamente nuestro último rombito como ocurre en la figura 11.15, habremos ganado la partida y nos aparecerá otra alerta con una felicitación.

Tras esto, se enviará al backend una notificación que marcará la partida como terminada, impidiendo que se pueda jugar más en ella, y haciéndola desaparecer

Figura 11.15: Detalle de la alerta de felicitación por ganar.

Figura 11.16: Detalle de los rombitos vacíos.

del listado de partidas de los dos usuarios.

En esta pantalla además tenemos el recuento de los rombitos de cada usuario. Este recuento se mantiene mediante un canvas colocado debajo de su nombre de usuario. Las casillas inicialmente se pintan con un relleno gris, como se puede ver en la figura 11.16.

Para posteriormente ir llenándolas con el color correspondiente, manteniendo los rombitos que no se han conseguido aún con el fondo gris. La figura 11.17 nos muestra el estado de los rombitos en una partida avanzada.

Dentro de esta pantalla se encuentra toda la lógica del juego, desde los movimientos de las fichas hasta las peticiones de preguntas y actualizaciones de datos (posición, turno, rombitos) en la base de datos. La ficha del jugador local está animada y pintada en azul. Esto se ha hecho para dar claridad al desarrollo

Figura 11.17: Detalle de los rombitos con una partida avanzada.

del juego, ya que me pareció que una ficha saltando y apuntando al lugar que le corresponde en el tablero sería más sencilla de localizar que una ficha estática.

Para ello he utilizado una animación que simula los latidos del corazón, pero en este caso con un triángulo invertido para marcar mejor la posición. La animación es un mini-script de CSS3 en la que se juega con el tamaño de la imagen, desde su tamaño original hasta 1.4 veces el mismo y vuelta al inicio, en un periodo de medio segundo en cada transición.

Esta animación la he basado en [6], sustituyendo la figura usada para la animación y modificando una serie de parámetros que la hacían inoperativa. Estos parámetros son los relativos a cada navegador para incluir los prefijos a las funciones de CSS3.

Parte IV

Revisión y conclusiones

Capítulo 12

Resumen final

Este proyecto consitió desde un principio en una investigación con el objeto de aprender cómo funcionan las dos tecnologías mencionadas en el título del mismo: HTML5 y CSS3.

Una vez concluido el desarrollo del mismo de una manera exitosa, puesto que todos los objetivos se han cumplido, puedo asegurar que el principal objetivo de todos está también cumplido: he sido capaz de organizar un proyecto de desarrollo software desde sus inicios, con el diseño de toda la plataforma, y el desarrollo de la misma, lo cual, para mí, es un gran paso hacia el frente en lo que a mi actividad como ingeniero se refiere.

He conseguido crear una herramienta de aprendizaje basado en un juego conocido por todos y lo he hecho utilizando las dos tecnologías que marcarán y están marcando el desarrollo web en la actualidad, lo que me da una base muy amplia para seguir construyendo otros servicios más adelante. Posiblemente incluso en mi vida laboral.

Por otra parte, he trabajado con un lenguaje de programación que podríamos decir que ëstá de modaẗon un framework que se utiliza mucho a nivel empresarial para crear distintas soluciones. Todo ello basado en una base de datos relacional con distintas claves primarias, otro de los objetivos no explícitos del proyecto, pero que era un requisito intrínseco de la arquitectura que tenía en mente al diseñarlo.

Como colofón, quería destacar que casi todo el código Javascript (menos las peticiones AJAX), está hecho en Javascript nativo, permitiendo así un acceso

desde todos los navegadores más potentes, aunque por motivos de eficiencia sólo se ha probado para Firefox y Google Chrome.

Capítulo 13

Pruebas y comentarios

Después de poner el juego online, decidí someterlo a pruebas externas. En estas pruebas participó tanto gente que tiene conocimientos informáticos como mis compañeros de clase, como otras personas con menos conocimientos como puede ser mis familiares.
Para ello, les di acceso a la plataforma, ellos crearon sus usuarios y se pusieron a jugar y a hacerme comentarios acerca de la aplicación con fallos y cosas que no les convencían. Los he decidido separar en distintas categorías en función de a qué parte de la aplicación atañen. Empezando con los de apariencia podría destacar los siguientes:

- Por mejorar un poco el aspecto, mantener un poco la relación, no tantos espacios vacíos, no se lo típico que se usa en la web ahora.
- Hay veces que el triangulo [por la ficha] hace que alguna casilla no se pueda pulsar.
- En mi ordenador pequeño no puedo ver toda la página, no puedo jugar.
- Las casillas blancas no dejan ver de qué color es la casilla que elegimos. Pondría una leyenda para saber a qué categoría pertenece cada color.
- Animación del dado. Botón de papelera algo descuadrado con la apariencia general.

Hablando sobre la apariencia y funcionamiento de la ficha:

- Quizá la haría más cercana a la celda en la que te encuentras ya que a veces parece que estas en la de al lado y quizá permitiría la elección de la forma de la ficha estilo monopoly, estaría curioso.

- Darle movimiento al desplazar.
- Un círculo en vez de un triángulo, o un círculo además del triángulo.
- Que no provoque ataques epilépticos.

En cuanto a fallos de programación y problemas en el juego:

- Parece que no tienes una página de error, te salta todo el DEBUG cuando intentas acceder a páginas que no existen, creo que podrías poner algo aunque no fuera muy elaborado, o por lo menos redirección al home. Cuando te salta el error de Django, parece que la sesión se expira, teniendo que meter de nuevo user y pass.
- Se puede acceder a los datos de otros usuarios: partidas que tiene que jugar, partidas, partidas en espera, e incluso lo que aún es peor, poder jugar sus partidas.
- Al crear un usuario nuevo, cuando te pide confirmar la contraseña, no tiene en cuenta si son iguales y solo coge la primera que le metes.
- Una vez que has pulsado sobre una pregunta, o si has lanzado el dado, si reinicias la pantalla, vuelve a donde antes de tirar y no tiene en cuenta donde estabas.

Todos estos fallos se han intentado buscar explicación y solucionarlos. La gran mayoría de ellos se han corregido, pero el de la seguridad de la aplicación no lo veo necesario ya que es una investigación sobre HTML5 y CSS3 por lo que temas de autenticación y seguridad de datos no están entre los objetivos prioritarios del proyecto.

En otra parte de la encuesta, que se puede encontrar aquí [7], pregunté varias cosas más, esta vez los resultados no eran libres, si no que era una valoración de la plataforma. Estos son los resultados obtenidos:

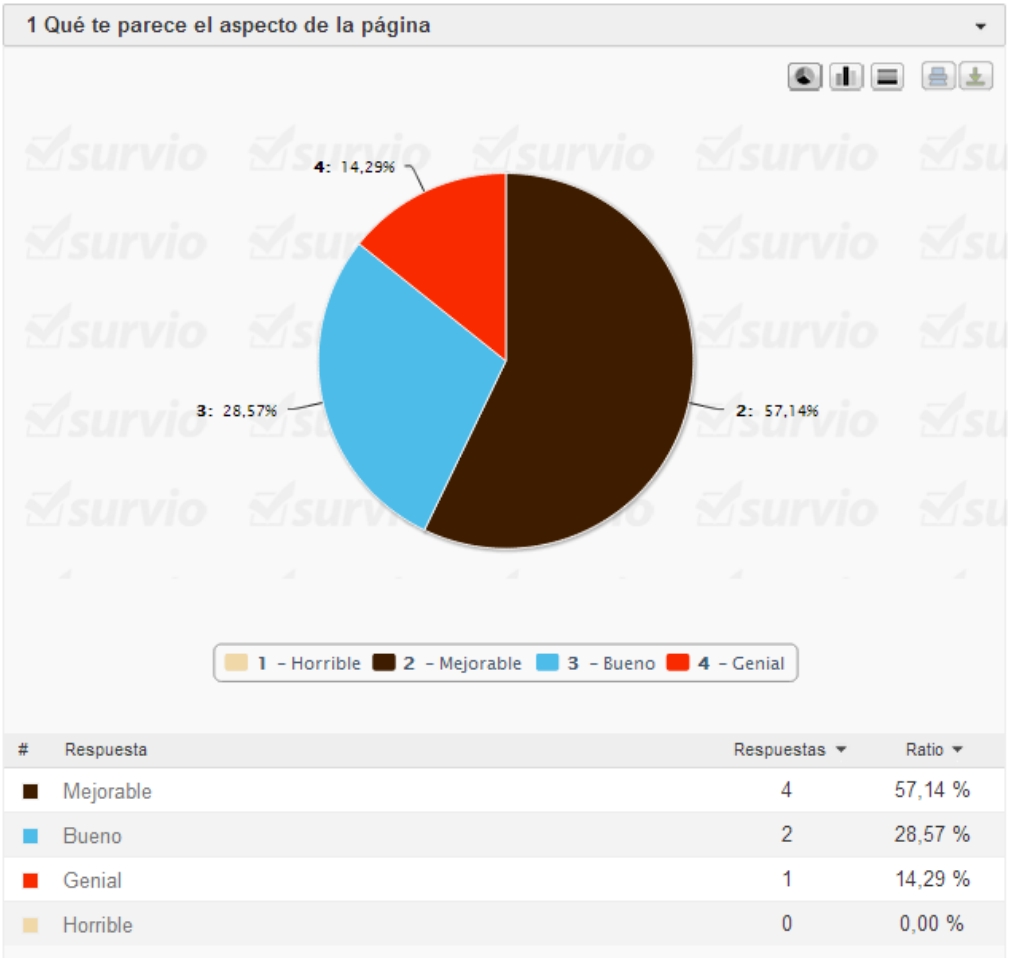
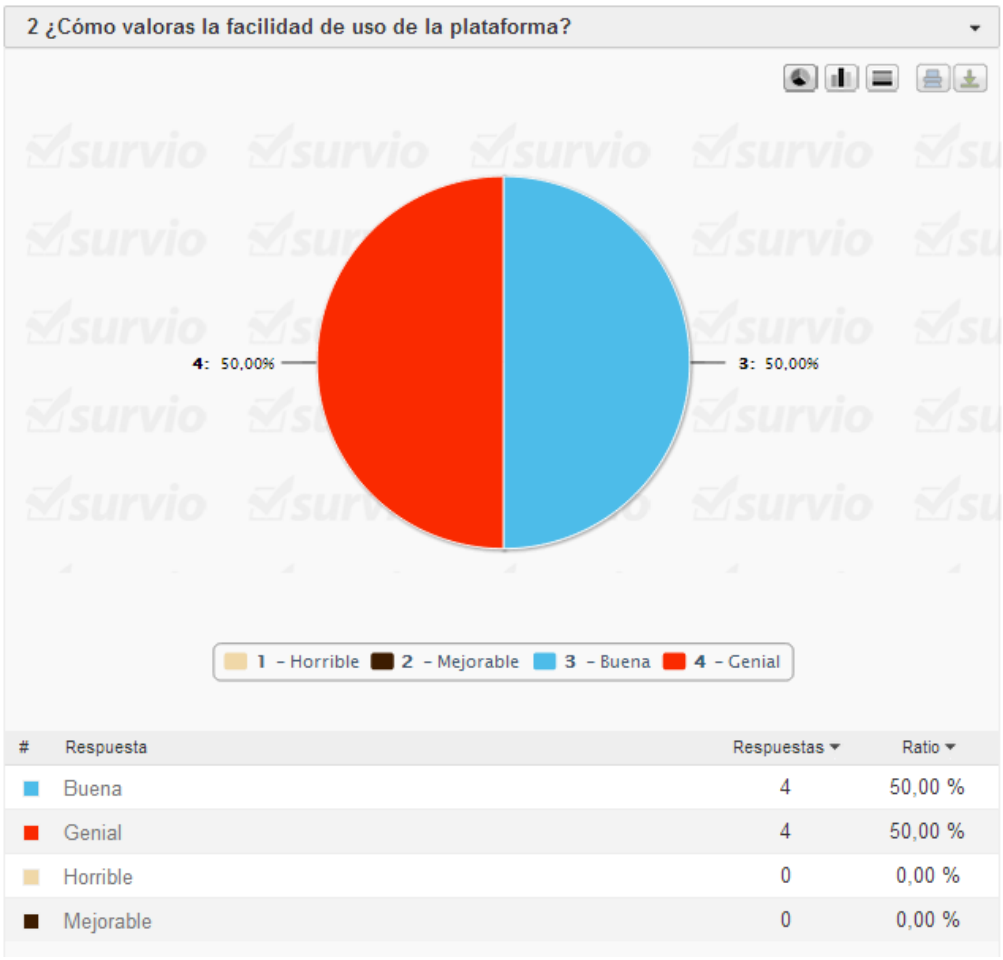
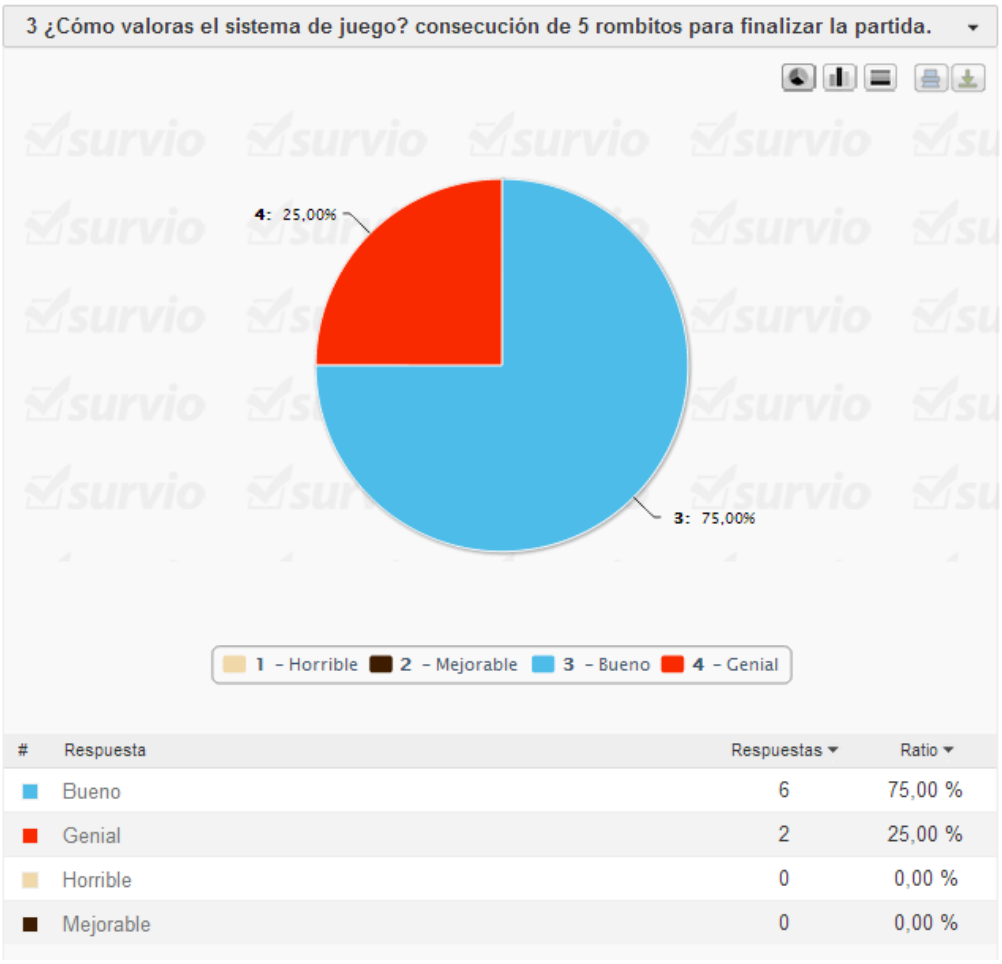


Figura 13.1: Pregunta 1 de la encuesta.



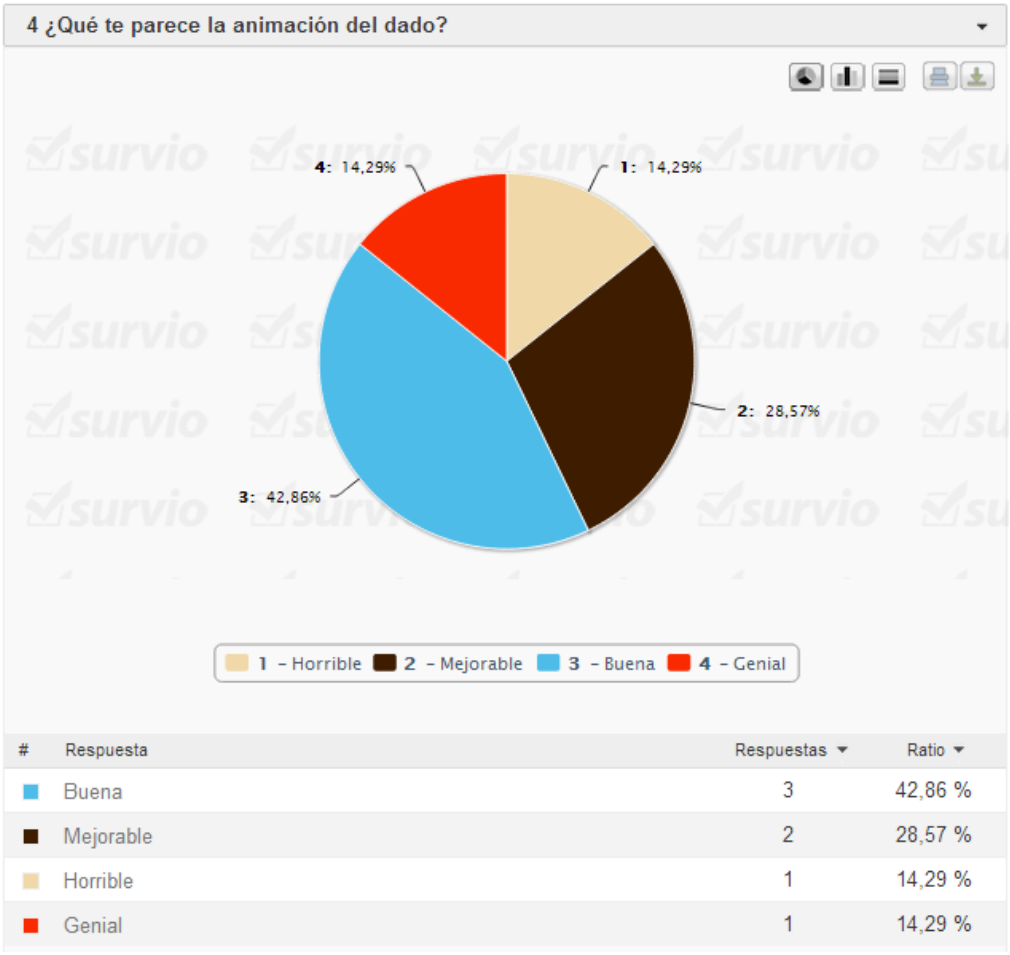
#	Respuesta	Respuestas	Ratio
	Buena	4	50,00 %
	Genial	4	50,00 %
	Horrible	0	0,00 %
	Mejorable	0	0,00 %

Figura 13.2: Pregunta 2 de la encuesta.



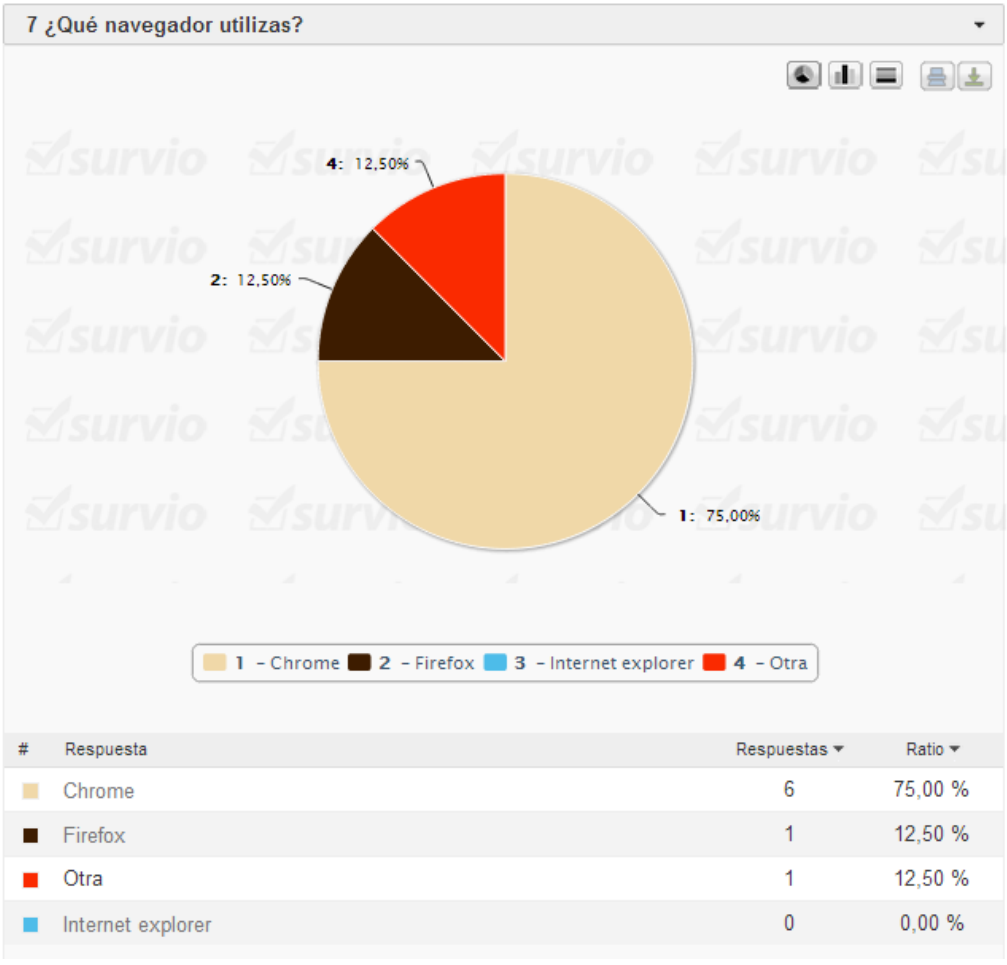
#	Respuesta	Respuestas	Ratio
	Bueno	6	75,00 %
	Genial	2	25,00 %
	Horrible	0	0,00 %
	Mejorable	0	0,00 %

Figura 13.3: Pregunta 3 de la encuesta.



#	Respuesta	Respuestas	Ratio
	Buena	3	42,86 %
	Mejorable	2	28,57 %
	Horrible	1	14,29 %
	Genial	1	14,29 %

Figura 13.4: Pregunta 4 de la encuesta.



#	Respuesta	Respuestas	Ratio
	Chrome	6	75,00 %
	Firefox	1	12,50 %
	Otra	1	12,50 %
	Internet explorer	0	0,00 %

Figura 13.5: Pregunta 7 de la encuesta.

Capítulo 14

Lecciones aprendidas

Durante los más de seis (intermitentes) meses que ha durado el desarrollo del proyecto, he logrado aprender muchas cosas. Algunas más importantes que otras, pero todas valiosas de cara al futuro.

Las más importantes que puedo destacar son estas:

- **El diseño de servicios web**: Este proyecto me ha permitido poner en práctica los conocimientos adquiridos en la asignatura de ingeniería del software sobre el diseño e implementación de un programa. Con ello he conseguido fijar dichos conceptos y algunas dudas que conservaba de la asignatura han quedado disueltas con la aproximación práctica de las mismas y una investigación un poco más exhaustiva.
- **Javascript**: Antes de comenzar el desarrollo del proyecto conocía un poco este lenguaje, pero durante el desarrollo del mismo me he dado cuenta de las capacidades que tiene y el potencial del mismo. Entiendo ahora que se considere el lenguaje del futuro, puesto que podemos evitar en gran medida las consultas al servidor, reduciendo así la cantidad de datos necesarios a la vez que el consumo del servidor.
- **HTML5**: Las nuevas herramientas disponibles para el desarrollo web era uno de los objetivos principales del proyecto. Durante el mismo me he peleado con el canvas para que pintase lo que quería, con las etiquetas y con las codificaciones de vídeo para que se viera en los distintos navegadores, pero al final he aprendido mucho y tengo unas cuantas notas mentales de cómo tienen que hacerse algunas cosas que no encontré muy claramente explicadas en Internet.

- **CSS3**: Hasta el momento de ponerme a trabajar con CSS3, todos mis servicios web habían sido bastante feos. Este proyecto quise que fuera un punto de inflexión en ese tema y he puesto especial ahinco en ello. Creo que he conseguido una interfaz bonita a la par que intuitiva, y que funciona perfectamente. Además quise meter animaciones directamente desde scripts CSS3, cosa que he conseguido hacer en el bote de la ficha de usuario y en la animación del recuadro del login y el registro.

- **Escritura en LaTeX**: Durante la carrera había usado siempre la herramienta de Microsoft Office Microsoft Word. De este modo he aprendido a utilizar LaTeX, el sistema de procesamiento de documentos empleado en la realización de esta memoria, y que seguramente usaré en todos los documentos oficiales de mis proyectos futuros.

Capítulo 15

Trabajos futuros

Una vez que el proyecto está acabado, se plantean varios posibles cambios de cara al futuro.
En primer lugar, sería una idea muy buena el intentar adaptar el juego para poderse usar como una manera de aprender por ejemplo un lenguaje de programación y, yendo aún más allá, hacerlo de tal manera que un dado administrador pudiera crear partidas a su antojo, eligiendo los tipos de preguntas y con la posibilidad de incluir preguntas de su propia gestión.

Otra de las posibles mejoras es la sustitución del dado, en el que actualmente se hace un cambio de imágenes mediante Javascript, por una animación CSS3 en la que fuese girando como haría un dado en la realidad. Para esto se requiere crear una animación muy compleja, dado que hay que definir el comportamiento de los puntos de cada cara y dada la limitada duración de este proyecto en paralelo con el estudio de las últimas asignaturas de la carrera, no me permitió hacerlo. Si que es una de las cosas que me habría gustado hacer, porque me parece que el efecto podía ser muy potente visualmente, pero he intentado hacerlo lo más bonito posible a la vez que con una implementación adecuada al tiempo disponible.

Otra cosa que podría ser intersante meter, sería la utilización de los websockets para la comunicación con el cliente desde el propio servidor. En este caso, la idea que tengo en mente es utilizarlo para enviar un mensaje al usuario de que ya está disponible una partida en la que le tocaba esperar. Me parece una cosa que puede enganchar a la gente a jugar ya que saben cuando les toca jugar y cuando no, por lo que sería muy útil para, en el caso de las estadísticas que comentaré ahora, tener más información útil sobre el juego.

Otro desarrollo futuro bastante potente, en mi opinión, sería la creación de un módulo de recogida de estadísticas. Podríamos saber qué usuarios son mejores, contestan más acertadamente a las preguntas, cuáles son las preguntas más difíciles, por ejemplo para utilizarlas en los rombitos dándole así una dificultad extra a estos, y cuanto tiempo pasan jugando los usuarios para modificar nuestra plataforma y hacerla más atractiva.

En otro orden de cosas, me habría gustado poder hacer el juego offline, es decir, utilizar la capacidad de *Local Storage* de HTML5 para guardar datos de las partidas en un directorio local, permitiendo así que se pudiera jugar sin necesidad de conexión a internet. El problema que vi para trabajar así, fue que las partidas son multijugador y por tanto existe implicitamente la necesidad de transmitir datos al servidor para que el otro usuario pueda jugar. Se podría valorar la opción de crear partidas de un solo jugador con otro objetivo de juego, por ejemplo el de batir su marca de tiempo en acabar una partida, o de número de aciertos en una misma partida, pero por motivos de tiempo no he podido valorarlo correctamente. Queda pues pendiente para una siguiente versión del software, y puede ser un buen punto de partida tomar lo que yo he hecho.

Por último, este diseño de la página ha sido creación propia con todo lo que ello conlleva. Estoy seguro de que un lavado de cara de la apariencia le haría más bien que mal, por lo que un trabajo exhaustivo de un equipo de diseño podría ser el paso final antes de considerar la aplicación totalmente finalizada.

Parte V

Apéndices

Apéndice A

Instalación en un servidor real

Después de hacer todo el desarrollo en un servidor local simulando las conexiones a internet llega la hora de ponerlo en circulación para que pueda probarlo la gente. En este capítulo, voy a hacer un breve tutorial de cómo he instalado la aplicación Django en un servidor Apache usando wsgi, una herramienta de enlace entre el Apache y Django.

A.1. Requisitos

Los requisitos mínimos necesarios son los siguientes:

- **Máquina servidor**: será donde estén todos los archivos almacenados.
- **Python**: necesario para la instación de django y para correr nuestra aplicación.
- **Servidor Apache**: encargado de servir las páginas, debe ir instalado en la máquina.
- **Framework Django**: instalado en la máquina, será el encargado de gestionar las páginas.
- **Mod_wsgi**: Plugin para enlazar django con el servidor apache.

A.2. Instalación del servidor Apache y mo_wsgi

Es el paso básico de la instalación de la aplicación. Este servidor lo he elegido porque había trabajado anteriormente con él y pensaba que me iba a resultar más fácil de tratar. Me equivoqué ya que no conseguía montar el wsgi, pero luego hablaré de ello.

Una vez instalada la máquina con Ubuntu, basta con ejecutar esta instrucción en consola:

```
> sudo apt-get install apache2 libapache2-mod-wsgi
```

Con ello se nos instalará el servidor apache, y la extensión de mod_wsgi que utilizaremos después.

A.3. Instalación de Python y Django

Ahora pasamos a la instalación del entorno de programación. Inicamos instalando python en la máquina servidora:

```
> sudo apt-get install python-setuptools
> sudo apt-get install python-pip
> sudo pip install django
```

A.4. Despliegue de la aplicación

Para el despliegue he seguido el tutorial de una página de internet [8]. En él se señalan las configuraciones que se deben meter en cada archivo y dónde hay que colocar cada archivo.

Por darle más claridad, voy a dividirlo en varios pasos.

A.4.1. Creación de archivo wsgi

Para que el servidor lea correctamente nuestra aplicación es necesario generar un archivo wsgi asociado a la misma. En este fichero se encontrarás las directrices para incluir el path de nuestro proyecto en el path del sistema, permitiendo de esta manera que nuestras settings se tengan en cuenta al lanzar la aplicación.

Con estos comandos crearemos el fichero asociado a nuestro proyecto (triviales es la carpeta que lo contiene), y lo abriremos para escribir en él las directrices necesarias.

```
> sudo mkdir /srv/www/triviales/apache
> sudo nano /srv/www/triviales/apache/django.wsgi

import os
import sys

sys.path.append('/srv/www')
sys.path.append('/srv/www/triviales')
sys.path.append('/srv/www/triviales/trivial')

os.environ['DJANGO_SETTINGS_MODULE'] = 'triviales.settings'

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

Tras esto, deberemos hacer saber al servidor Apache que existe esta aplicación. Para ello, deberemos crear un archivo dentro de la carpeta de sitios disponibles incluida en la instalación de Apache:

```
> sudo nano /etc/apache2/sites-available/triviales
```

Esta orden nos abrirá una vista del programa de edición de textos Nano donde deberemos escribir lo siguiente:

```
<VirtualHost *:80>
   ServerName pfc-lalonso.libresoft.es
   ServerAlias pfc-lalonso.libresoft.es
   ServerAdmin luisluso89@gmail.com

   DocumentRoot /srv/www/triviales

   WSGIScriptAlias / /srv/www/triviales/apache/django.wsgi

   Alias /favicon.ico /srv/www/triviales/images/favicon.ico
   Alias /static/ /srv/www/triviales/static/
        <Location "/static/">
            Options -Indexes
        </Location>
</VirtualHost>
```

Ahora necesitamos darle algunas directrices al servidor para que redirija y trate bien las peticiones. Esto se hace con las siguientes directrices escritas en el httpd.conf del servidor.

```
AliasMatch /([^/]*\.css) /sr/www/triviales/static/styles/$1

<Directory /usr/local/wsgi/static>
Order deny,allow
Allow from all
</Directory>

WSGIScriptAlias / /srv/www/triviales/apache/django.wsgi

<Directory /usr/local/wsgi/scripts>
Order allow,deny
Allow from all
</Directory>
```

Una vez guardado ese archivo, activamos el sitio web para que empiece a servirlo el Apache. Tras esto, hace falta recargar el servidor para que la nueva configuración surta efecto. Esto se debe a que Apache es un servidor estático, y no tiene ningún mecanismo propio de actualización de configuraciones.

Con estas instrucciones tendremos el sitio activado y el servidor se reiniciará:

```
> sudo a2ensite triviales
> sudo /etc/init.d/apache2 reload
```

Una vez ejecutado esto, ya podremos acceder a nuestro servicio mediante la url pfc-lalonso.libresoft.es y disfrutar del Triviales.

Apéndice B

Uso de la aplicación

A pesar de que esta aplicación se basa en un juego conocido por todos como es el Trivial Pursuit, tiene unas características diferentes y un modo de juego distinto al famoso juego de mesa. Para que se pueda probar correctamente la aplicación, voy a pasar a explicar el funcionamiento y algunas de las reglas de la misma.

B.1. Registro

El primer paso para poder jugar. En la pantalla de inicio, pulsaremos sobre el botón de registrar que nos redireccionará a de creación de cuentas. En esta, deberemos introducir un nombre de usuario, que se utilizará para entrar en la plataforma posteriormente, un email válido, que en la actualidad no se utiliza más que para evitar que un mismo usuario tenga dos cuentas con el mismo email, y una contraseña que deberá ser repetida dos veces para asegurarnos de que nos acordaremos.

B.2. Creación de partidas

Cuando hayamos finalizado el registro, seremos redirigidos a la pantalla resumen de las partidas que tenemos actualemente. Como acabamos de registrarnos, no tendremos ninguna y tendremos que crear una para poder jugar. Para ello, como ya se ha comentado antes, tenemos tres opciones:

- Buscar por nombre de usuario
- Buscar por correo electrónico
- Crear partida contra un oponente al azar.

Cuando la partida esté creada, entraremos directamente en la fase de juego, cuyas reglas explico en la siguiente sección.

B.3. Jugar a triviales

Para comenzar a jugar necesitamos tirar el dado. Una vez que obtengamos la puntuación, podremos elegir la casilla de destino entre los dos movimientos posibles, hacia la derecha y hacia la izquierda, para contestar nuestra primera pregunta. Los temas elegidos para esta versión de Triviales son:

- **Deportes**: Son las casillas de color azul. En ellas encontraremos preguntas de todo tipo relacionadas con cualquier deporte posible.
- **Historia**: Son las casillas amarillas. Todo lo que haya pasado en la historia del planeta puede entrar en esta categoría.
- **Ciencia**: Casillas de color verde. En ellas podemos encontrar preguntas de ciencia e ingeniería. Desde biología hasta relaciones de potencia. Si es científicamente probable, se encuentra aquí.

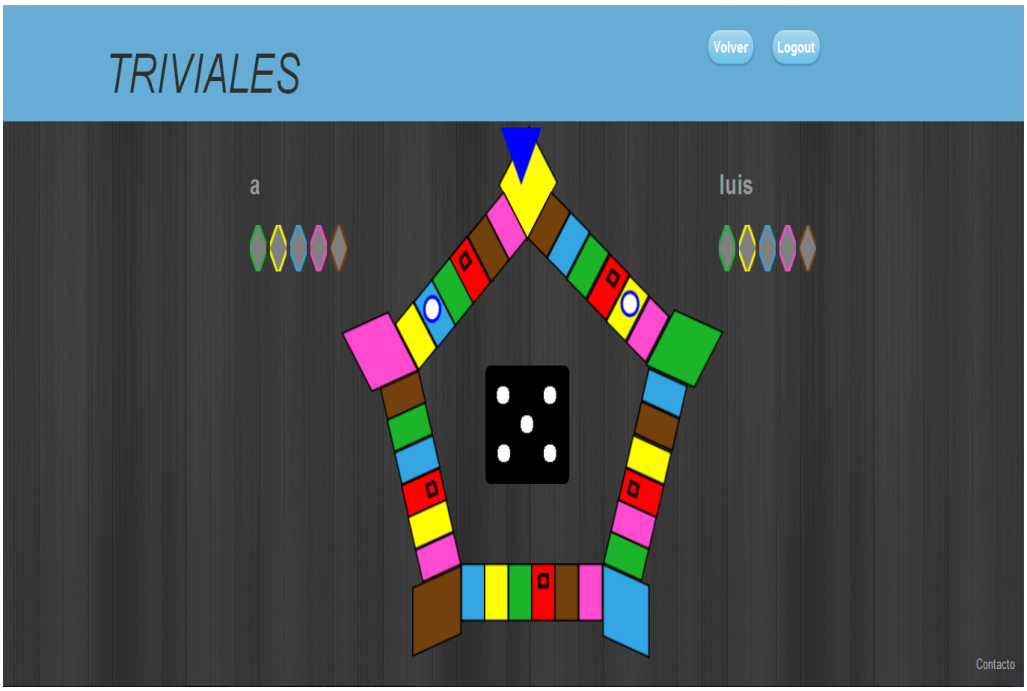


Figura B.1: Detalle de las casillas marcadas con un círculo.

- **Literatura**: Las casillas de color marrón son las casillas más literarias. En ellas podemos encontrar pregunta de temas relacionados con la literatura (como premios Nobel) o estadísticas sobre los libros más vendidos del planeta.
- **Espectáculos**: Casillas de color rosa. Tendremos que responder sobre temas como cine, teatro, música, etc. si queremos ganar este rombito.
- **Tira otra vez**: Casillas de color rojo con un cuadrado pintado. Estas casillas no tienen preguntas asociadas y nos permiten volver a tirar para continuar jugando.

Cuando tiramos el dado se marcan sobre el tablero las dos posiciones posibles con un círculo blanco. Esta decisión se ha tomado a raiz de las pruebas de concepto hechas con mis compañeros de clase y amigos. Dado que ha cambiado de la especificación inicial, en la figura B.1 podemos observar el estado final del tablero con las posiciones definidas.

Si contestamos correctamente a la pregunta que nos aparece, seguiremos jugando, mientras que si fallamos el turno pasará a nuestro rival y tendremos que esperar a que este falle para poder seguir jugando.

Si caemos en un rombito, que son las casillas más grandes, tendremos la opción de conseguirlo. Para ello deberemos contestar correctamente a la pregunta que

aparezca, y si conseguimos los cinco rombitos, antes de que el otro usuario lo consiga, habremos ganado la partida.

Bibliografía

[1] Canvas en w3schools.com. [Online]. Available: www.w3schools.com/html/html5_canvas.asp

[2] Draft oficial de CSS3. [Online]. Available: www.w3.org/Style/CSS/current-work

[3] The web framework for perfectionists with deadlines — django. [Online]. Available: https://www.djangoproject.com/

[4] Luis Alonso. Repositorio en GITHub del proyecto. [Online]. Available: https://github.com/luso11/triviales

[5] The Apache Software Foundation. [Online]. Available: www.apache.org

[6] Lea Verou. Animations with one keyframe. [Online]. Available: lea.verou.me/2012/12/animations-with-one-keyframe/

[7] Luis Alonso. Encuesta de pruebas de la aplicación. [Online]. Available: my.survio.com/Q1Y8K4N1T2F5J2H5U5E9/data/index

[8] Kevan Stannard. Installing Django with Apache and mod_wsgi on Ubuntu 10.04. [Online]. Available: blog.stannard.net.au/2010/12/11/installing-django-with-apache-and-mod_wsgi-on-ubuntu-10-04/

[9] Dive into HTML5. [Online]. Available: diveintohtml5.info

[10] HTML5 Rocks. [Online]. Available: www.html5rocks.com/es/

[11] W3C HTML5 página oficial. [Online]. Available: http://www.w3.org/html/

[12] HTML5 Demos. [Online]. Available: html5demos.com

[13] Tutorial de CSS3. [Online]. Available: www.w3schools.com/css3/

[14] CSS Transforms. [Online]. Available: css3.bradshawenterprises.com/transforms/

[15] Mary Lou. CSS3 LIGHTBOX. [Online]. Available: tympanus.net/codrops/2011/12/26/css3-lightbox/

[16] Javascript object notation. [Online]. Available: http://www.json.org/