UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Aplicación web para selección de imágenes utilizando tecnologías web en la nube

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

AUTOR/A: Pérez Lavarías, Diego

Tutor/a: Martínez Zaldívar, Francisco José

Cotutor/a: Igual García, Jorge

CURSO ACADÉMICO: 2021/2022

Resumen

Este proyecto ha sido realizado con la meta de desarrollar una aplicación web cuya utilidad sea la de implementar una herramienta para, de una manera lo más sencilla y transparente al usuario posible, obtener información acerca de sus gustos y preferencias en lo relativo a imágenes. Esta información obtenida ha de ser válida posteriormente para realizar una investigación en la que se pueda relacionar las preferencias del usuario con las características de las imágenes.

Para ello, se partió con la idea de enfrentar dos imágenes y solicitar al usuario que elija su preferida, con lo que, tras un cierto número de iteraciones con las diferentes imágenes incluidas en base de datos, se pueda realizar una clasificación y a partir de ella poder investigar si existe relación entre las elecciones realizadas por los usuarios y las características propias de la imagen.

Para llevar a cabo este proceso, se ha realizado una investigación acerca de las distintas partes implicadas (aplicaciones web, imágenes y sistemas de clasificación de elementos en base a enfrentamientos), para después desarrollar la aplicación y permitir su acceso a través de Internet de forma universal y sencilla.

Resum

Aquest projecte naix amb la idea de desenvolupar una aplicació web amb la utilitat d'implementar una ferramenta que, de la manera més senzilla i transparent amb l'usuari possible, obtinga informació rel·lacionada amb els seus gustos i preferències en l'àmbit de la fotografia. Aquesta informació ha de ser vàlida posteriorment per a dur a terme una investigació en la qual es puga rel·lacionar les preferències de l'usuari amb les característiques de les imatges.

Per a fer-ho possible, s'inicia el projecte amb la premissa d'enfrontar dues imatges i sol·licitar a l'usuari que trie la seua favorita i, després d'un indeterminat nombre d'iteracions amb les diferents imatges que es troben incloses en la base de dades, realitzar una classificació amb la qual investigar si existeix una rel·lació entre les eleccions realitzades pels usuaris i les característiques pròpies de les imatges.

Per a completar el procés, s'ha realitzat en primer lloc una investigació de les distintes parts implicades (com són les aplicacions web, les imatges, i els sistemes de classificació d'elements en base a enfrontaments directes), per a després desenvolupar la pròpia aplicació i permetre el seu accés a través d'Internet de forma universal i senzilla.

Abstract

This project has been carried out with the objective of developing a web application implementing a tool that gives us the possibility of retrieve information from users about their personal preferences in photografy field, always with maximum simpleness and transparency. This information has to be useful for us in order to investigate whether this personal preferences are related with images characteristics or not, and if we can make conclusions about it.

To make it happen, we started with the idea of confronting two images and asking the user to pick their favorite one and, after some iterations repeating this process with other images included in the database, create a classification that we can use to investigate the relation between that picks that the users made and the images' characteristics.

To complete the process, an investigation about the different involved sides has been made (web applications, photografy and classification systems based on direct clashes), to later on develop the app and create a system to allow to access via Internet to it.

Índice

Capítulo 1. Introducción

1.1 Motivación

La decisión de realizar este proyecto se ha fundamentado sobre diferentes motivos.

El principal es la unión de varios de mis intereses, como son las tecnologías modernas (el uso de Internet, los diferentes tipos de dispositivos desde los cuales se puede acceder, el acceso de gran parte de la población a ello lo cual aumenta el rango de acción en el que se puede llegar a causar cambios…) así como la fotografía (un gran interés de uno de mis tutores, Jorge).

El acceso a la fotografía así como a los diferentes tipos de cámaras es algo que ha estado muy presente en mi vida desde hace años, incluso en tiempos en los que los teléfonos aún no incluían módulos fotográficos y parecía que estaba más restringido, y se acabó convirtiendo en uno de los factores que influyeron en mi decisión de estudiar en el sector TIC.

El interés en las tecnologías y en Internet es también algo que me caracteriza desde siempre, ya que pertenezco a la generación que creció al mismo tiempo que se desarrollaba el mundo de la interconexión global y, como es muy habitual, no se entiende mi desarrollo personal y mi crecimiento sin este factor.

Es por ello que creo que este trabajo mezcla bien ambos factores, además de resultarme interesante debido a que está pensado para utilizar las opiniones de las personas para obtener resultados, es decir, no es un trabajo cerrado sino que incluye una retroalimentación con la sociedad lo cual para mí es vital a la hora de hablar de tecnologías, que no se entienden sin la participación de las personas que las usan en el día a día.

1.1.1 Personas y fotografía

El estudio de la relación entre las personas y la fotografía como arte, como muestra de gustos, o como estudio de toma de decisiones es el factor principal por el cual elegí este proyecto cuando lo vi en la oferta.

La imagen es un componente muy importante en la comunicación, y mucho más en las formas de comunicación que dominan en la actualidad, donde la facilidad de tomar fotografías y enviarlas es máxima, en muchas ocasiones mayor incluso que mensajes de texto o llamadas telefónicas. Esto, sumado a la influencia que estas imágenes tienen sobre las personas, nos puede dar una cantidad de información muy grande sobre las propias personas, así como conocer sus gustos, opiniones, o distintos factores de las imágenes que influyen en ellos.

Un buen ejemplo es el funcionamiento de la aplicación Instagram. En ella, se recibe en todo momento información del usuario: en qué imágenes se fija (desde el usuario que la sube hasta los colores o la composición), en qué temas se interesa, en qué anuncios entra… No es tan simple como subir fotografías y que los demás las vean, sino que tiene detrás un trasfondo que afecta al comportamiento personal y una cantidad de datos con la que se puede llegar a conocer a un conjunto de individuos de forma muy precisa.

Es importante saber que aplicaciones de este tipo son usadas por la mayor parte de la población mundial. Un estudio de We Are Social y Hootsuite a inicios de este 2022 refleja que alrededor de 1500 millones de personas utilizan de manera activa Instagram, mientras otras redes basadas en imágenes como Pinterest alcanzan los casi 500 millones de usuarios:

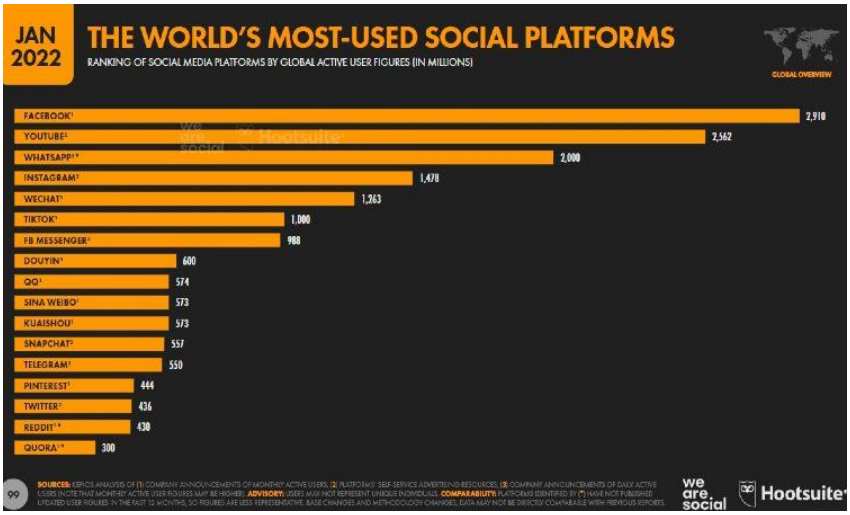

Figura 1. Estudio de uso de redes sociales, 2022

Por tanto, podemos llegar a la conclusión de que el uso de herramientas de este tipo está ampliamente integrado en nuestro día a día y que se ha convertido en algo tan cotidiano que realizar un buen uso de ello puede ser vital para establecerse en el mercado, así como para conocer a la sociedad de cara a cualquier tipo de objetivos tanto sociales como económicos.

1.1.2 Páginas web

Desde el establecimiento de los navegadores web como el principal modo de acceso a Internet (principalmente debido a su ligereza, facilidad de uso, posibilidad de uso en diferentes plataformas y/o sistemas operativos…) las páginas web se han convertido en la forma más popular de consumo de contenido en red por parte de los usuarios.

La simplicidad que aportan a la hora de ser mantenidas y actualizadas desde el lado del servidor también las ha convertido en algo muy deseable para los creadores de este contenido, con lo que cada vez la experiencia de uso ha ido aumentando y con ello el número de herramientas, la complejidad de las creaciones y la ayuda disponible de forma rápida y eficaz.

Estos dos motivos han sido los principales para decidir implementar la idea del proyecto mediante la creación de una aplicación web, con el añadido de poder tener una comunicación entre el usuario y nosotros, es decir, la información que se le aporta, mediante la interactividad constante que se permite.

Figura 2. Evolución páginas web (inicio-actual)

1.2 Objetivos

El principal objetivo de este proyecto es comenzar el trabajo con una aplicación web que nos permita obtener datos de este tipo de las personas que la utilicen. Esto es, mediante el uso de una web en apariencia simple, conocer la opinión de personas en temas que no se les preguntan directamente.

Por tanto, el objetivo es crear esta página con imágenes y, en base a una sucesión de enfrentamientos entre imágenes que se le presentan al usuario, poder obtener información relacionada con las imágenes que elija y que está implícita en las propias fotografías, como puede ser su composición de colores, el tema que presenta, la compresión de la imagen, etc.

Así, los diferentes objetivos que se presentan como parte de la creación de la página son:

- La investigación de cómo se pueden obtener datos a partir de las decisiones de los usuarios.
- La creación de algoritmos para la presentación dirigida de imágenes al usuario (mostrarle las fotografías que nos interesen a la hora de categorizar sus gustos).
- La creación de algoritmos de clasificación de imágenes (crear un orden en base a los gustos del usuario).
- Entender la base de todos estos algoritmos e investigaciones.

Otro objetivo del proyecto es el de realizar este trabajo de creación de la aplicación de la forma más moderna, simple y escalable posible. Esto es, estudiar el uso de herramientas de creación de aplicaciones en la nube que se nos ofrecen y que están muy presentes en el día a día del sector, ya que permiten realizar proyectos realmente avanzados con facilidad, además de aportar un amplio abanico de opciones que cubren la mayoría de necesidades que se pueden tener (bases de datos, *hosting* de la aplicación, autenticación/usuarios…). Este objetivo también incluye que la aplicación esté disponible en varias plataformas, ya que como se refleja en gran cantidad de estudios[cita], el uso de los navegadores web no es algo propio de un ordenador o de un portátil, y existen muchas maneras de hacerlo, siendo ya en la actualidad la preferente el uso de un *smartphone.*

Por tanto, nos marcamos como objetivo que la página web sea totalmente accesible y esté adaptada para su visualización tanto en un PC como en un dispositivo móvil (un tablet, un teléfono…).

1.3 Metodología

Se ha definido una estructura para realizar el proyecto de cara a dividirlo en distintas partes en las cuales se pueda avanzar y ordenar estos avances de forma clara. Este proceso o ciclo de vida es bastante habitual en el desarrollo de *software* y facilita el proceso a la vez que incrementa la calidad del mismo.

Para este proyecto, se ha decidido dividir el proceso en tres bloques. Estos bloques son el estudio e implementación de las herramientas necesarias para el almacenamiento, mantenimiento y funcionamiento de una página web, la creación de la propia página web y por último la investigación acerca de los algoritmos que controlan y orientan los resultados de la página web. Estos bloques fueron organizados de forma lineal en el orden en el que han sido indicados, con la salvedad de que los dos últimos han tenido algo de solapamiento debido a que las investigaciones de los algoritmos han aportado nuevas soluciones que en algunos casos incluían la necesidad de realizar cambios en la web de cara a mejorar o facilitar el uso por parte del usuario.

El primer paso fue el del estudio de las diferentes herramientas existentes cuya funcionalidad es la de facilitar o agrupar las distintas necesidades que una página web tiene y permitir el

mantenimiento de la misma en una sola interfaz o de forma intuitiva y lo más simple posible. Como se ha estudiado a lo largo de la carrera y el máster, las aplicaciones del tipo de la que se trata en este trabajo requieren de dos partes: la del servidor y la del cliente. En nuestro caso, el interés principal se encuentra en el lado del cliente, con la información que queremos obtener del uso de la misma aplicación, y por ello nos decidimos a estudiar las distintas alternativas que se nos presentaban a la hora de facilitar la programación y creación del *backend* o servidor, ya que nuestras necesidades en este lado no eran muy avanzadas y de este modo nos hemos podido dedicar en mayor medida al otro extremo de la balanza. Más adelante en esta memoria se incluye una comparativa de las distintas opciones y cuál ha sido la elegida.

El segundo paso es el de la propia creación de la aplicación de manera funcional. Esto es, la programación de las funcionalidades propias y necesarias de la misma (autenticación, presentación visual, funcionamiento sin errores…) siempre teniendo en cuenta el siguiente bloque y por tanto dejando abiertos algunos frentes que podrán ser modificados, pero de forma totalmente transparente para el usuario, que podría usar la aplicación de la misma manera en cualquier momento tras la finalización de este bloque sin importar que se haya llevado a cabo la creación de los algoritmos de investigación o no.

Por último tenemos dicho bloque, el del estudio de los diferentes algoritmos a implementar de cara a realizar la investigación. En este bloque podemos incluir dos tipos diferentes de algoritmos: por un lado tenemos los algoritmos de clasificación de las fotografías, esto es, cómo clasificar unas fotografías en relación a las otras, cúal está por encima y cuál por debajo según las votaciones de los usuarios, etc; y por otro lado tenemos los algoritmos de selección de las imágenes, que se ocuparán de elegir qué imágenes deben ser presentadas en cada momento a cada usuario, de manera que las votaciones que se realicen sean la mayor parte del tiempo interesantes para la investigación. Aquí se puede incluir desde algoritmos simples (evitar que se presente una imagen ante sí misma) o más complejos como la elección de imágenes con puntuaciones similares, composiciones similares u opuestas…

En el siguiente diagrama se observa el tiempo dedicado a cada módulo y el orden en el que se ha llevado a cabo tanto el desarrollo *software* como las investigaciones:

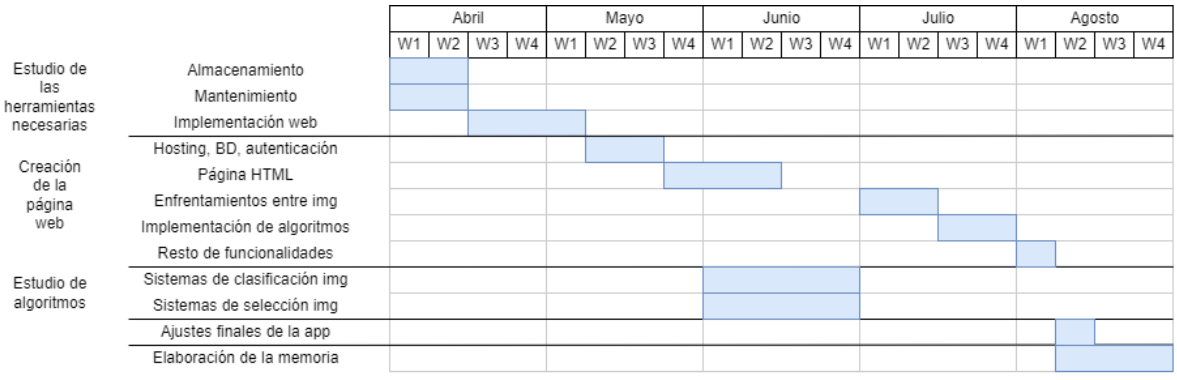

Figura 3. Diagrama de Gantt que explica los plazos temporales seguidos

Capítulo 2. Identificación y análisis de posibles soluciones

En este capítulo se llevará a cabo un análisis de las distintas opciones que se presentan para implementar el proyecto. El 100% del proyecto se realiza en *software*, por lo tanto todos los siguientes apartados tratan acerca de alternativas *software* para los distintos bloques de los que se compone el mismo.

2.1 Creación local de la página web (*front-end*)

En este subapartado se trata las alternativas que se presentan a la hora de crear la presentación de la página web y de dotarla de funcionalidades. Esto es, de qué manera se puede llevar a cabo la presentación de la información que va a ver el usuario y con la que va a interactuar.

En primer lugar, las herramientas que no pueden faltar son HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) y la programación JavaScript.

2.1.1 HTML

HTML[1] es el lenguaje de marcado utilizado para la creación y elaboración de páginas web. Se trata de un estándar que se encarga de mantener el Consorcio WWW, de la misma manera que hace con la gran mayoría de tecnologías relacionadas con la web.

Es el estándar debido a que es el lenguaje que se impuso en los inicios para la visualización de páginas web y es el que todos los navegadores utilizan. Existen alternativas como podrían ser lenguajes como XML y predecesores como SGML, pero dado que estamos hablando del estándar es el más usado y por tanto el que más herramientas proporciona, además de tener una enorme base de usuarios con lo que se facilita mucho el trabajo con la gran cantidad de opciones que se presentan y de ayudas para solucionar problemas que se pueden encontrar sin dificultad ninguna.

Este lenguaje se basa en etiquetas, que alteran el texto introducido en su interior dependiendo de las necesidades del usuario: por ejemplo, etiquetas de título, de imagen, de hipervínculos…

La estructura más básica de una página HTML es la siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <title>Hola HTML</title>
  </head>
  <body>
    <p>Hola Mundo!</p>
  </body>
</html>
```

Figura 4. Ejemplo básico de página HTML simple

En la figura podemos observar el uso de las etiquetas, comenzando por la que delimita el propio fichero <html>, pasando por la cabecera <head> y el cuerpo <body>. Todas estas etiquetas tienen su propio cierre.

[1] *Programación en Internet: Clientes Web* de Sergio Luján Mora.

2.1.2 CSS

Como se ha observado en el subapartado anterior, HTML nos aporta una gran flexibilidad y posibilidades de aumentar la complejidad en nuestro diseño y en la estructura de la página web, pero al estar basado en formato de texto plano, realmente necesita de añadidos externos que se integren en el mismo fichero y doten de mayor profundidad a la creación de la página.

Ese es precisamente el cometido de CSS[2]. Se trata de otro estándar mantenido por el Consorcio WWW y su función es la de ser un lenguaje de diseño gráfico que se acopla a HTML y define o describe la presentación de los elementos HTML dentro de una pantalla de navegador (o básicamente en cualquier lugar donde se pueda presentar una página web).

El funcionamiento de CSS se basa en archivos de texto (que pueden funcionar de forma externa a los HTML y ser importados o ir insertados directamente dentro de un fichero HTML) que modifican y añaden propiedades a las etiquetas creadas en la página web. Las propiedades van desde simples como el color del texto, el tamaño de la fuente, o similares, hasta más complejas como la forma que tendrá de comportarse esa etiqueta según se solape con otras, o cómo se debe alinear dentro de la representación en pantalla.

Como ejemplo de código CSS tenemos el siguiente:

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

Figura 5. Ejemplo básico de hoja de estilos CSS

En la figura podemos apreciar tres modificadores.

En primer lugar, se modifica el color del fondo de todo el cuerpo de la página web. En segundo lugar se le cambia el color al título (etiqueta h1) y se centra el texto en relación a la página, y finalmente se cambia la fuente y tamaño del texto de los bloques de texto que tengan la etiqueta p (de párrafo). El resultado final sería:

Figura 6. Resultado del ejemplo de la figura 4

Este lenguaje ayuda a resolver un gran problema: HTML no fue creado para describir el contenido de una web ni para añadirle un formato. Aun así, en versiones más modernas, se le añadieron algunas de estas capacidades, pero no resulta sencillo de usar ni es nada escalable, por lo que el uso de CSS se convierte en obligatorio a la hora de crear páginas complejas a la vez que aporta una gran capacidad de personalización y de posibilidades de mejorar la experiencia de uso.

[2] Información extraída de W3Schools

2.1.3 Lenguaje de programación (JavaScript)

En este apartado se van a comentar las posibilidades de las que se dispone a la hora de trabajar con el front-end. Esto es algo necesario dado que nuestra página no solo va a tratarse de una presentación al usuario, sino que va a reclamarle una participación al mismo y va a realizar procesos con esa información. Esto es posible simplemente usando HTML, pero no de forma muy avanzada, quedando en la simplicidad de obtener datos mediante un formulario, por ejemplo.

En cambio, para nuestra aplicación, tenemos la necesidad de obtener datos del usuario (cuál de las dos fotografías que se le presenten prefiere), realizar cálculos con ellos (aumentar/disminuir puntuaciones de las imágenes) y actuar en consecuencia (presentar otras dos imágenes al usuario en base a sus elecciones anteriores).

Para esto la principal herramienta a usar es JavaScript. No se trata de un estándar mantenido por un consorcio como los apartados anteriores de HTML y CSS, pero sí se puede decir que funciona como un estándar de facto, siendo la opción mayoritaria en este campo y la que mejor integrada está con las herramientas estandarizadas.

JavaScript o JS es un lenguaje de programación interpretado. Se define popularmente como "orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico". Es el lenguaje que más tipos de aplicaciones y usos puede abarcar en la actualidad, y es el que más se adecúa a nuestro trabajo. Es importante no confundirlo con Java, con el cual no comparte características más allá del nombre.

La parte que más nos interesa para este proyecto es la capacidad que aporta a las páginas HTML de ser más dinámicas e interactivas, ya que combina la presentación que dicho lenguaje genera junto con la creación de un programa con sus cálculos, funciones, y todo tipo de posibilidades como manipulación de imágenes, validación de formularios, etc.

Una vez comprendidas las necesidades básicas que tenemos, pasamos a estudiar las dos formas que se han considerado en cuanto a la programación del *front-end.*

2.1.4 Frameworks

Como alternativas, existen los llamados marcos o *frameworks*. Este tipo de herramientas, normalmente basadas también en JavaScript o Python, ayudan a reducir la complejidad a la hora de crear las instrucciones y el código que se quiera añadir en la aplicación, ofreciendo códigos preescritos (o librerías) a partir de los cuales el desarrollador puede construir su programa.

Como ejemplos, podemos encontrar como opciones más usadas *React o Angular*.

- React: se trata de una herramienta basada en JavaScript para construir interfaces de usuario, permitiendo el diseño de vistas simples para cada parte de la aplicación con la idea de desarrollar aplicaciones de una sola página. Hace que el código sea predecible y por tanto fácil de depurar mediante las llamadas vistas declarativas.
  Está basado en componentes encapsulados que manejan su propio estado y, al estar escritos en JavaScript se pueden trasladar los datos y cambios de manera sencilla a través de la propia aplicación.
  También permite el renderizado desde el servidor utilizando Node, y la construcción de páginas para móviles mediante React Native.
  Es de código abierto y se encuentra mantenido por Facebook y la comunidad de software libre, y algunas aplicaciones en las que se usa son Imgur o Airbnb.

- Angular: es otro ejemplo similar a React. En este caso está desarrollado en TypeScript (creado a partir de JavaScript y con múltiples añadidos), también de código abierto y mantenido por Google.

También se basa en componentes y sus principales características son la capacidad de desarrollo en múltiples plataformas, buen rendimiento, gran número de herramientas y muchos usuarios a nivel global.
Permite webs progresivas y dinámicas, posibilidad de usar cualquier sistema de back-end, plantillas con las que comenzar y capacidad de realizar todos los procesos incluidos en el desarrollo, como el testeo o las animaciones.

2.1.5 Librerías

De manera similar a los *frameworks*, el propio JavaScript permite la importación de librerías más simples que afecten al funcionamiento de CSS y HTML, modificando el aspecto de la interfaz y aportando la capacidad de realizar modificaciones simples. En este caso, no sería necesario nada más que la instalación de la librería a usar, lo cual añade nuevas instrucciones o comandos directamente al CSS que estamos usando y que enlazan con funciones propias como botones, formularios…

Algunos ejemplos son las librerías Materialize o Bootstrap.

- Bootstrap: se define como un '*toolkit'* o caja de herramientas de front-end. Fue desarrollado dentro de Twitter y su uso es muy destacado actualmente, siendo uno de los proyectos más usados de Github y es usado por la NASA, como ejemplo más famoso.
  Su uso es muy sencillo, no es necesario instalar nada, simplemente cargando como script el CDN de Bootstrap el navegador ya es capaz de funcionar con las instrucciones que aporta.
  A partir de dicho momento, tenemos disponibles los iconos, temas, incluso trozos de código algo más complejos (*snippets*) que añaden cabeceras de páginas, barras laterales o desplegables, etc. Se puede ver un gran número de ejemplos en su web[3].

- Materialize: se trata de otra librería muy similar a Bootstrap, pero con la diferencia de que es más moderno y fue creado posteriormente, y cuya mayor característica es la implementación del diseño Material de Google, un diseño limpio y apropiado para este tipo de aplicaciones.
  Más allá de eso, funciona de la misma manera que Bootstrap y añade funcionalidades análogas, como los botones, desplegables, etc.

2.2 Implementación web de la página (*back-end*)

En este subapartado se analizan las opciones disponibles para implementar las funcionalidades de servidor de la aplicación, de manera que el *front-end* sea totalmente funcional y el *back-end* sea transparente de cara al usuario a la vez que simple de mantener y escalable en el caso de que el número de usuarios crezca, además de proporcionar un buen rendimiento.

En este caso no se realiza de forma tan básica como el *front-end*, tenemos distintas alternativas pero ninguna es tan básica como el caso de HTML/CSS.

De esta manera, las opciones entre las que hay que elegir son *frameworks*, donde las diferencias se encuentran en el lenguaje en el que se basan, o incluso la creación de un servidor local en una máquina. Esta última opción es claramente peor, debido a que para cada herramienta que se necesite usar se debería crear prácticamente desde cero (existen facilidades y librerías para conseguir cualquier objetivo, pero de forma más compleja que usando un *framework*). Esta alternativa sería claramente superior si se deseara una mayor complejidad de la página y un mayor control de la base de datos y de la autenticación de usuarios.

Pasamos ahora a observar las alternativas que se han considerado para el *back-end.*

[3] https://getbootstrap.com

2.2.1 Firebase

La primera y principal alternativa estudiada ha sido Firebase de Google. Es una plataforma de desarrollo de aplicaciones que cuenta con millones de usuarios, convirtiéndose en la más usada para este tipo de proyectos. Fue lanzada en 2011 y adquirida por esta compañía en 2014.

La principal característica de esta plataforma es que está ubicada en la nube, concretamente en Google Cloud, por lo que todas las herramientas necesarias para crear y mantener el back-end de la aplicación están integradas en la plataforma y no han de ser instaladas por el usuario en su máquina.

Aporta las soluciones de compilación, lanzamiento y supervisión de la aplicación, con lo que el usuario que tenga en su poder el front-end creado solamente necesitará realizar lo que se conoce como un '*deployment*' para integrar su código en la web y ofrecer su aplicación de manera remota a los clientes o usuarios. Desde ese momento entra en juego el proceso de supervisión y mantenimiento de la app, para lo que también se ofrecen herramientas en Firebase.

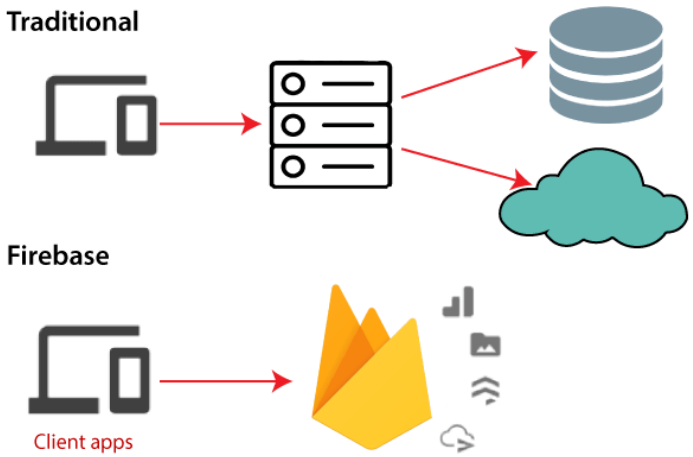

Figura 7. Diferencia entre los backend tradicionales y el servicio Firebase

Los puntos positivos que aporta, por lo tanto, son:

- Creación de proyecto sin necesidad de crear y mantener un servidor local propio. De esta manera se eliminan muchas barreras en un simple paso, ya que para ofrecer la aplicación sería necesario montar un servidor local, así como abrir la red privada local a Internet de manera que cualquier usuario pueda entrar, lo cual abre también la puerta a posibles ataques, además de tener que crear también soluciones como bases de datos, autenticación…
- Fácil sincronización de proyectos, así como implementación de un sistema de seguimiento de cambios y múltiples personas manteniendo el proyecto.
- Herramientas multiplataforma: desde navegadores sobremesa hasta sistemas operativos móviles.
- Uso de la infraestructura de Google, lo cual permite una gran escalabilidad.

Algunos de los productos ofrecidos son el Hosting, la Autenticación, las bases de datos Realtime y Firestore, almacenamiento en la nube, Google Analytics o monitorización del rendimiento de la aplicación.

Como ejemplos de casos que usan este servicio, tenemos la web del diario New York Times o la tienda Alibaba, entre otros.

2.2.2 AWS Amplify/Lambda

Como alternativa a Firebase aparecen los servicios web de Amazon o AWS, concretamente sus productos Amplify y Lambda.

El producto Lambda sería el equivalente a Firebase, soportando diferentes herramientas como Node.js o Python a la hora de crear el back-end. No es tan simple como la alternativa de Google, pero las posibilidades son mayores debido a este soporte. El propio sistema se define en su web como una solución para ejecutar código sin tener que pensar en los servidores o clústeres, esto es, nos permite crear toda la infraestructura necesaria para almacenar y presentar nuestra aplicación de manera remota en la nube de Amazon. Esto permite ahorrar muchos costes, ya que solamente se paga por el tiempo de uso y no por servicios que quizá no se van a usar o exceden nuestras necesidades.

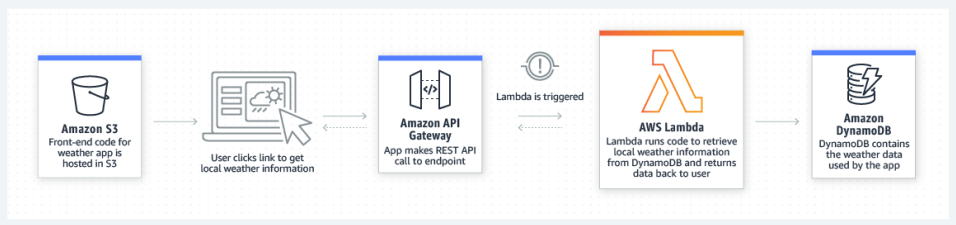

Figura 8. Servicios aportados por AWS Lambda

Sus casos de uso son el procesamiento de datos a escala, la ejecución de back-ends web/móvil interactivos (combinando Lambda con otros servicios) y la creación de aplicaciones basadas en eventos, facilitando la comunicación entre servicios que se encuentran desacoplados.

En cuanto a Amplify, va algo más allá, ya que combina ambas partes de la aplicación, tanto el front-end como el back-end, aportando sistemas de autenticación, almacenamiento y demás como ocurre en Firebase. Es el sistema que utilizan en la propia empresa para la web de su servicio de *streaming* de música, Amazon Music.

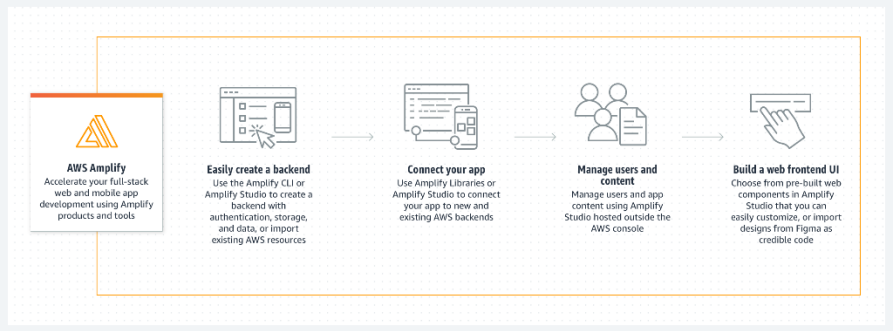

Figura 9. Servicios aportados por AWS Amplify

2.3 Sistemas de clasificación de las imágenes

En este punto de la memoria se investigan las diferentes posibilidades que se tienen de cara a implementar un sistema que clasifique las imágenes entre sí, para poder enlazar las elecciones que realice el usuario con las fotografías que se le han presentado previamente. El sistema ideal conseguiría ordenar las imágenes según las preferencias de los usuarios en el menor número de enfrentamientos posible para, a partir de este orden, realizar una investigación y obtener conclusiones.

Para la elección de un sistema de este tipo, hemos tratado de identificar cómo funcionan las clasificaciones en un gran número de aplicaciones existentes. En nuestra opinión, tenemos unas necesidades similares a las que se podría tener en una aplicación deportiva o competitiva

(clasificar el nivel de los diferentes competidores de cara a generar enfrentamientos interesantes y equilibrados).

Es por esto que se ha elegido entre las siguientes alternativas:

2.3.1 Sistemas ELO y similares

Los sistemas denominados ELO (por el profesor Arpad Elo, el inventor del primer sistema de este tipo) consisten en métodos basados en la estadística para calcular la habilidad o el nivel de un grupo de jugadores o de contrincantes en un juego. Se popularizaron debido a su uso en ajedrez, deporte para el cual fue ideado el primero de estos sistemas, el ELO puro.

Estos sistemas consisten en otorgar una puntuación a cada participante, que va variando al realizar enfrentamientos con otros usuarios, de manera que los resultados influyen en los puntos de cada uno de ellos y van subiendo a medida que el jugador va ganando combates o partidas. De esta manera, y como es lógico pensar, cuantas más partidas gane un jugador más puntos acumulará.

La peculiaridad de este tipo de sistemas es que el cálculo de cuántos puntos gana o pierde cada participante se basa no solo en ganar o perder el enfrentamiento, sino en lo que la estadística opina del combate, es decir, las probabilidades que cada uno tenía de ganar o perder. De esta manera, la diferencia de puntuación influye en la ganancia y pérdida de puntos por lo que no es igual obtener la victoria en un combate equilibrado que en uno en el que se parte a priori con una gran ventaja.

Se ha considerado el uso de este tipo de sistemas ya que se encuentran similaridades entre un enfrentamiento de imágenes y una partida de ajedrez, siendo un uno-contra-uno, y contando con un puntaje cada imagen que se ha ido desarrollando con diferentes enfrentamientos en el tiempo.

También es importante indicar que este sistema tiende a estabilizarse con el tiempo y por tanto a colocar a cada participante en su lugar, y que su uso en gran cantidad de deportes y juegos de competición nos dice que suele ser fiable.

Algunos ejemplos más allá del sistema ELO clásico son el Glicko (usado también en ajedrez y go) o Trueskill (implementado por Microsoft para el sistema de videojuegos de Xbox).

2.3.2 Sistema round-robin o victoria/derrota

Otro tipo de sistema que se ha considerado es el más típico en los deportes tradicionales, como el fútbol o el baloncesto. Consiste en realizar enfrentamientos entre todos los participantes a modo de competición, de manera que funciona como una liga: una victoria te suma puntos y una derrota te resta o te suma 0. Así, una vez todos los contrincantes se han enfrentado entre ellos (una o varias veces), se realiza la clasificación por orden de puntuación.

Tiene algunas ventajas respecto al sistema ELO, ya que es mucho más simple de implementar al ser solo una suma después de cada enfrentamiento, y un orden al acabar con todos. También se necesita un calendario simple que indique los enfrentamientos que se han realizado y los que faltan, ya que coger los resultados sin que todos los participantes hayan tenido la oportunidad de enfrentarse a todos los demás daría resultados engañosos.

Pero eso también lo convierte en una desventaja, ya que nuestra idea de proyecto incluye la facilidad de coger los resultados en cualquier momento y que el usuario pueda dedicarle el tiempo que él quiera y no tener que realizarse todos los enfrentamientos. Otra desventaja es la gran posibilidad de empates que puede aparecer, en especial si el número de contrincantes es bajo y que puede darse un círculo de enfrentamientos en el que uno le gane a otro y así sucesivamente hasta llegar nuevamente al primero.

2.4 Descripción de la solución adoptada

En este apartado se procede a explicar qué alternativas se han escogido de entre las planteadas en el punto anterior, y el porqué de dichas elecciones.

2.4.1 Front-end

Para realizar el front-end de la aplicación se ha decidido, aparte de las herramientas estándar HTML, CSS y JavaScript, que no era necesario el uso de un *framework* avanzado como serían Angular o React, y que simplemente se necesita alguna librería para adaptar la presentación de la página a nuestras necesidades, pero que dado que no se requiere de una gran complejidad se puede cubrir los requisitos de esta manera.

La mayor complejidad que tiene la web es el comportamiento dinámico, ya que solamente se dispone de un cuadro o página pero no debe mostrar la misma información en el caso de estar registrado y tras haber hecho *login* que en el caso contrario. Esto se puede llevar a cabo sin problemas mediante el uso de Materialize, como se explica en apartados posteriores, y es mucho más simple para el desarrollador ya que tiene el código más concentrado y no es necesario llevar el mismo orden, además de no necesitar aprender a usar una nueva herramienta como sería un *framework.*

Concretamente, se ha decidido usar las distintas opciones que Materialize ofrece en cuanto a apariencia general de la web, así como botones, clases de ventanas estilo *popup* y formularios.

Como se ha explicado anteriormente, esta librería implementa el Material Design creado por Google, lo cual se ha considerado un punto muy positivo dado que es un diseño que se ha usado muchísimo en la última década desde que fuera introducido en el año 2014 y al que la mayoría de la gente está acostumbrada. Utiliza diseños simples, basados en cuadrículas, animaciones ligeras, algunos efectos como son luces o sombras, y que sobre todo aporta una sensación de ligereza a las aplicaciones que lo utilizan.

Además, tanto la librería Materialize como el diseño Material están totalmente pensados y adaptados para su uso en terminales móviles, uno de los grandes objetivos del proyecto.

2.4.2 Back-end

Para el back-end se ha tomado la decisión de utilizar Firebase de Google. Esta plataforma facilita el soporte y mantenimiento de una página web, ya que agrupa todas las soluciones necesarias para ello: desde el *hosting* de la propia página (el almacenamiento de la web en los servidores de Google de forma que pueda accederse remotamente) hasta la seguridad, la autenticación de usuarios, la base de datos…

El uso de Firebase ha requerido de un pequeño proceso de aprendizaje, pero no ha conllevado ningún problema gracias a la cuantiosa documentación de que dispone en su página y de las facilidades que aporta, como código de ejemplo que con pequeñas modificaciones se adapta a cualquier proyecto.

También se ha valorado como positivo el sistema de versiones que incorpora, además de la posibilidad de añadir colaboradores al proyecto, en mi caso mis tutores, por lo que ha sido sencillo compartir información con ellos en los casos en los que aparecían dudas.

Más adelante en esta memoria se explica en detalle los servicios de Firebase que se han utilizado, el proceso seguido desde el inicio para tener acceso a todo, y los problemas que han aparecido y cómo han sido subsanados.

Por último, otro motivo para elegirlo ha sido el precio de uso. Como aparece en su web, este producto tiene dos planes de precios, y nos viene muy bien dado que uno de ellos se adapta precisamente a nuestro caso de uso: proyectos a pequeña escala y en el ámbito de la educación.

El llamado Plan Spark o sin costo permite el uso de los sistemas de autenticación (hasta 50.000 usuarios al mes), hosting (10 GB, con dominio personalizado), base de datos (hasta 100 conexiones simultáneas y 1 GB de almacenamiento) y almacenamiento en la nube (hasta 5 GB). Nuestras necesidades están más que cubiertas con estos números, con lo que el uso que realicemos de Firebase será siempre gratuito. También se encuentra positivo que el uso del otro plan es prepago, por lo que la aplicación siempre va a ser escalable, y en el caso de que se aumenten las necesidades el precio a pagar será por las herramientas a usar y no existe la restricción que tienen otros servicios de tener que contratar servicios que no se van a usar.

2.4.3 Sistema de clasificación

El sistema de clasificación de imágenes que se ha decidido utilizar es una mezcla entre los sistemas ELO clásico y el más moderno y actual Glicko-2. El motivo principal por el que se ha decidido es que entendemos el objetivo del proyecto como crear enfrentamientos entre dos imágenes (a pesar de que existe un modo con un número más elevado de fotografías, no es el principal) y se encuentran muchas similaridades con un juego como el ajedrez o incluso con juegos o competiciones actuales, ya que este sistema se utiliza por ejemplo en muchos videojuegos multijugador en los que se intenta adjudicar o asignar un nivel de capacidad a cada jugador de cara a crear los enfrentamientos más igualados o interesantes.

También se elige para darle al usuario la posibilidad de estar el tiempo que desee realizando enfrentamientos, ya que este sistema, si bien es más exacto cuantos más partidas se realicen, permite que un enfrentamiento sea totalmente independiente del resto y a la vez tenga en cuenta la historia de cada contrincante.

Otro motivo por el cual se ha elegido estas alternativas es que son sistemas que se encuentran programados en la actualidad en una gran cantidad de lenguajes, por ejemplo en Node.js, con lo que adaptarlos a nuestra página es sencillo y su uso también. Es la mejor manera de evitar errores a la hora de crear código o incluso a la hora de usarlo, ya que en los repositorios en los cuales se puede obtener estos sistemas se ofrece ayuda y es posible que exista solución en el caso de que surja algún problema en cualquiera de los procesos.

Capítulo 3. Fundamentos teóricos de la comparación entre imágenes

Dentro de las elecciones que se han realizado y de las características del propio proyecto, se ha tenido que realizar un estudio de cara a conocer los entresijos de los objetivos que se quieren cumplir. Es por esto que se han identificado dos puntos principales de los cuales se considera que hay que realizar explicaciones más detalladas acerca de su funcionamiento y de la teoría que se esconde detrás del mismo.

3.1 Sistemas ELO

En este apartado se va a explicar la base teórica de los sistemas ELO, así como su origen, funcionamiento, y en especial cómo se ha utilizado en este proyecto y lo que se ha conseguido con su uso.

En primer lugar, y para añadir alguna explicación a las razones expuestas anteriormente de por qué usar este tipo de sistemas, la principal es porque se ha encontrado una gran similaridad del objetivo de nuestro proyecto con juegos como el ajedrez, y este parecido es que se ha identificado como un juego de suma cero. Esto significa que en la teoría de juegos (también sirve para situaciones de economía) es un juego entre dos lados enfrentados y que el resultado es una ganancia para un lado y una pérdida equivalente para el otro, de forma que lo que uno gana lo pierde el otro. De esta manera, el beneficio neto entre ambos es cero, en un sistema ideal el número de puntos totales entre todos los contrincantes será constante.

Así, podemos pasar a explicar el sistema ELO original. Ideado por el profesor de física Arpad Elo[4] en la década de los 50, con el objetivo de aplicarlo al ajedrez, juego del cual era fanático, y por tanto mejorar los sistemas que se utilizaban entonces, no empezó a usarse de forma generalizada por la Federación Internacional (FIDE) hasta 1970. Desde entonces el sistema ha sufrido de modificaciones y ha sido adoptado por la gran mayoría de federaciones y asociaciones de este deporte hasta el punto de que sería impensable una competición sin usar un sistema de clasificación de este tipo.

Pero no solo se ha utilizado para el ajedrez, sino que se ha implementado en otros deportes y juegos como el fútbol americano, baloncesto, tenis de mesa y en especial juegos de mesa y deportes electrónicos (que en la actualidad pasan por un gran momento y en ciertos países incluso compiten con los deportes tradicionales en número de participantes y espectadores[cita]).

Para esta explicación nos vamos a centrar en el sistema ELO original, ya que el resto de los existentes se basan en adaptaciones a partir de éste y en añadidos posteriores a su uso generalizado, por lo que su entendimiento es simple después de comprender el original.

3.1.1 Funcionamiento

En el tiempo en el que Elo participaba en torneos de ajedrez de la Asociación Americana, se utilizaba un sistema numérico ideado por Harkness que, si bien se consideraba más o menos justo, en ciertas circunstancias daba resultados extraños. De esta manera Elo decidió que se necesitaba un sistema que tuviera una base estadística más adecuada. Por ejemplo, algunos sistemas de *rating* aportan un valor a cada torneo o partida de forma arbitraria, como ocurre en tenis con los diferentes torneos, y no varía según la dificultad del recorrido o de los rivales encontrados, con lo que la puntuación del competidor es la misma sin importar los rivales.

[4] The Rating of Chessplayers, Past and Present, 1978

En cambio, con un sistema basado en la estadística, se puede tener en cuenta la habilidad de los jugadores a la hora de determinar su nivel tras un partido, de forma que no es lo mismo ganar ante un rival de tu nivel que a uno muy superior o inferior.

Siguiendo con Elo y sus estudios, determinó que el nivel de los jugadores seguía la misma forma que una distribución normal. Es decir, que su nivel se encuentra en una media, y que de vez en cuando podía desplazarse un poco hacia arriba o hacia abajo, pero que los cambios de una partida a otra no serían demasiado grandes (algo que sí podía ocurrir en el sistema anterior) y que para que los cambios sean notables el jugador debe estar por encima de su media de manera sostenida en el tiempo.

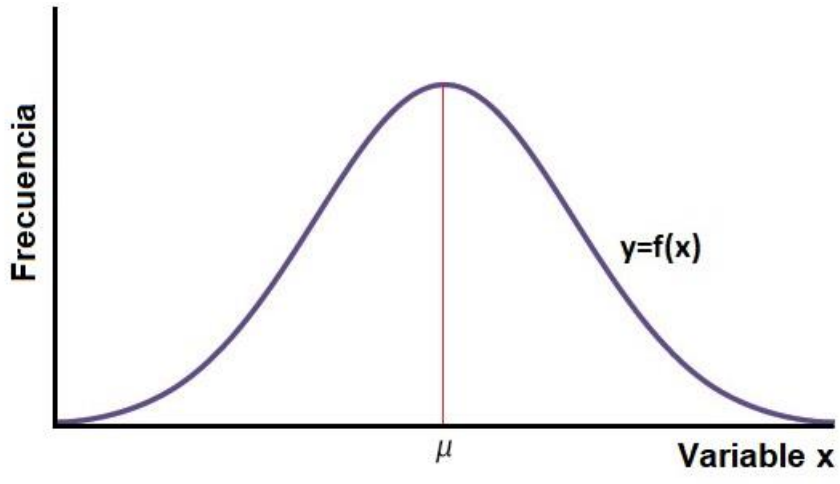

Figura 10. Gráfica correspondiente a una distribución normal

Así, Elo asumió que cuando un jugador ganaba a otro, había rendido a un nivel algo superior a su media, y cuando perdía igual pero a un nivel inferior. La magnitud de esta diferencia entre el nivel medio y el desempeño inmediato, entonces, debe venir dada por la diferencia de niveles entre un jugador y su rival.

Aquí es donde aparece la faceta estadística del sistema. De manera previa a una partida o enfrentamiento, los *ratings* de uno y otro actúan como predictores de cómo debe ser el resultado final y cómo de igualado debe estar. Por eso, el sistema entiende que si, por ejemplo, el jugador con menor puntuación gana el encuentro, es que ha rendido bastante por encima de su nivel medio, y por tanto los puntos que gane serán más que los que ganaría el a priori favorito.

Para calcular la probabilidad de victoria de un jugador A ante otro B, el sistema utiliza la siguiente fórmula:

$$E = \frac{10^{R_A/400}}{10^{R_A/400} + 10^{R_B/400}},$$

Figura 11. Fórmula de probabilidad E utilizada en el sistema Elo Rating

Donde R_A y R_B son los *ratings* de cada uno de los jugadores.

Por ejemplo, para unos niveles de 1500 y 1700, la probabilidad E nos dice que hay en torno a un 24% de posibilidades de que A gane la partida.

Entonces, una vez calculadas estas probabilidades, podemos pasar al cálculo, o mejor dicho, a la estimación posterior. Para ello, en nuestro caso, hemos tenido en cuenta tanto el resultado del

enfrentamiento como lo que se había previsto que pudiera pasar, así como un factor externo que nos ayuda a obtener resultados más grandes o pequeños si así lo deseamos. Así, la fórmula implementada en nuestro código es:

$$rating\ change = k * (resultado - E)$$

Donde k es el factor externo, por defecto 20 y en nuestro caso 50, resultado es 1 en caso de victoria y 0 en caso de derrota, y E es la probabilidad de victoria calculada en el paso anterior. Esta cifra será luego aplicada a cada uno de los lados del enfrentamiento y sumada a la puntuación previa al mismo.

En este caso podemos identificar una gran diferencia con la fórmula habitual para el ajedrez, ya que en ese caso no solamente se tiene en cuenta el enfrentamiento actual y la puntuación previa al mismo, también se tiene un factor que sopesa y añade más importancia a torneos recientes con respecto a resultados más antiguos, ya que en el caso de personas se tiene en cuenta que el nivel actual es más representativo que aquel que pudiera haber dado hace un tiempo. En nuestro caso pensamos que no es así y que todas las opiniones de los usuarios se han de tener en cuenta por igual para asignar una puntuación.

La asignación de 50 para el factor k ha sido debida a que nos interesa que las puntuaciones fluctúen algo más de lo que lo hacían por defecto, ya que el número de iteraciones, es decir, de enfrentamientos que se requerían para llevar a cabo cambios significativos se convertían en algo demasiado grande, pero en estudios posteriores se puede probar a disminuir este número e intentar llegar a un número de usuarios superior. Números más bajos para el factor k pueden ayudar a que cada elección tenga un menor peso y por tanto evitar elecciones fallidas o no representativas.

3.1.2 Aplicación del sistema al proyecto

Para aplicar esto al proyecto, y dado que tenemos varias restricciones y necesidades, se ha decidido implementar dos sistemas diferentes.

En primer lugar, para el modo básico de uso, que es el enfrentamiento entre dos imágenes, se ha implementado el sistema clásico *elo-rating*. Funciona de la manera que se ha explicado en el apartado anterior, tomando la puntuación de cada una de las imágenes incluidas, llamando a la función que controla la aplicación del sistema, que devolverá directamente los *ratings* actualizados.

Como ejemplo, con puntuaciones de A = 1516 y B = 1418, se obtiene un valor E = 0.637, es decir, se asigna una probabilidad de victoria del 63.7% para A y 36.3% para B. Sabiendo esto, y aplicando la siguiente fórmula, obtenemos que A ganará 18 puntos mientras que B los perderá, y por tanto, el resultado final será A = 1534 y B = 1400. Resultados de este tipo son los que nos resultan interesantes, ya que la variación de puntos es suficiente como para ir estableciéndose en torno a una puntuación media en la que debería estar cada imagen en un estado de equilibrio final ideal, pero no tanto como para que dos malos resultados consecutivos provoquen un gran descenso.

En segundo lugar, para el modo alternativo que se ha ideado (9 imágenes simultáneas, donde solo gana una) con el objetivo de tener resultados más rápidos (por supuesto menos exactos) se ha decidido aplicar un sistema Glicko-2, el cual da la posibilidad de realizar enfrentamientos numerosos. No es el modo principal de uso ni es tan interesante para el proyecto, pero puede servir en casos en los que se tiene un número de candidatos muy elevado y se quiere descartar muchas fotografías rápidamente. De este modo, con imágenes que bajen de cierto umbral, se les puede considerar menos interesantes y por tanto nos ahorramos una gran cantidad de tiempo.

En este caso, el sistema es similar al ELO, pero con la salvedad de que los enfrentamientos ahora permiten que haya una clasificación final en lugar de una victoria/derrota. En nuestro caso, lo que nos interesa es 1 ganador y el resto empatados en segundo lugar, con lo que entre ellos no habría intercambio de puntuaciones y el ganador resultaría en un mayor aumento de puntos a costa del resto de perdedores.

3.1.3 Selección de imágenes

Otro apartado para el que interesa el uso del sistema de clasificación de imágenes es para la selección previa. Esto es, la elección de las imágenes que se van a enfrentar de manera que los enfrentamientos no sean aleatorios, y que los resultados sean importantes. Por ejemplo, si la selección es aleatoria, puede darse el caso de que se enfrenten dos fotografías que se encuentren muy diferenciadas en puntuación. En este punto, la previsión está demasiado desbalanceada, con lo que el cambio en puntos no sería notorio y estaríamos perdiendo el tiempo.

Para llevar a cabo dicha tarea, se ha pensado en un sistema de categorías o agrupaciones. De esta manera, cada fotografía estará incluida en una categoría en base a su puntuación, y solamente se podrían enfrentar imágenes de la misma categoría entre sí, o como mucho con una categoría de diferencia en el caso de que haya pocas imágenes en una o ya se hayan enfrentado todas entre sí para un mismo usuario. El objetivo es que los enfrentamientos estén siempre nivelados y balanceados, con lo que los resultados serán interesantes siempre.

Capítulo 4. Desarrollo de la aplicación

En este capítulo se va a describir el desarrollo de la aplicación en la que se ha implementado nuestro proyecto. Se mostrará un pequeño diagrama de bloques en el que nos hemos basado, siguiendo con el diseño de la propia aplicación y de la base de datos para cumplir con dicho diagrama y, finalmente, una descripción del código.

4.1 Diagrama de bloques del proyecto

En primer lugar, se va a explicar la idea inicial de la aplicación. Para ello, se ha diseñado el siguiente diagrama o esquema que muestra el funcionamiento visible de la misma:

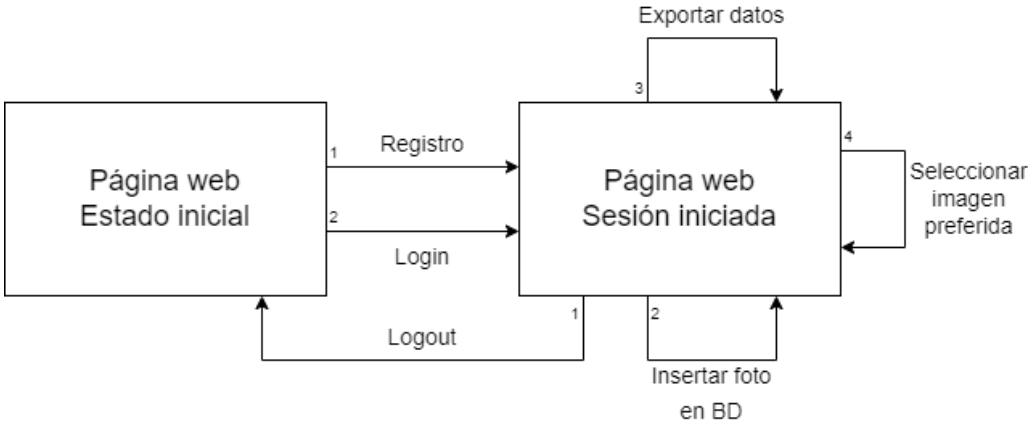

Figura 12. Diagrama de bloques del funcionamiento de la aplicación web

En él, vemos como nuestra idea es la de tener solamente una página que muestre distinta información según si el usuario ha iniciado sesión o no.

En el primer caso, se le indicará que debe iniciar sesión o registrarse, y por tanto aparecerán los botones correspondientes. En el segundo caso, al estar ya dentro de una sesión, aparece directamente el diseño final, con las imágenes para elegir, además de los botones que incorporan las nuevas funcionalidades, como cerrar sesión, exportar los resultados o cambiar entre los modos de 2 y 9 imágenes representadas.

Para implementar esto, se ha llevado a cabo la siguiente estructura:

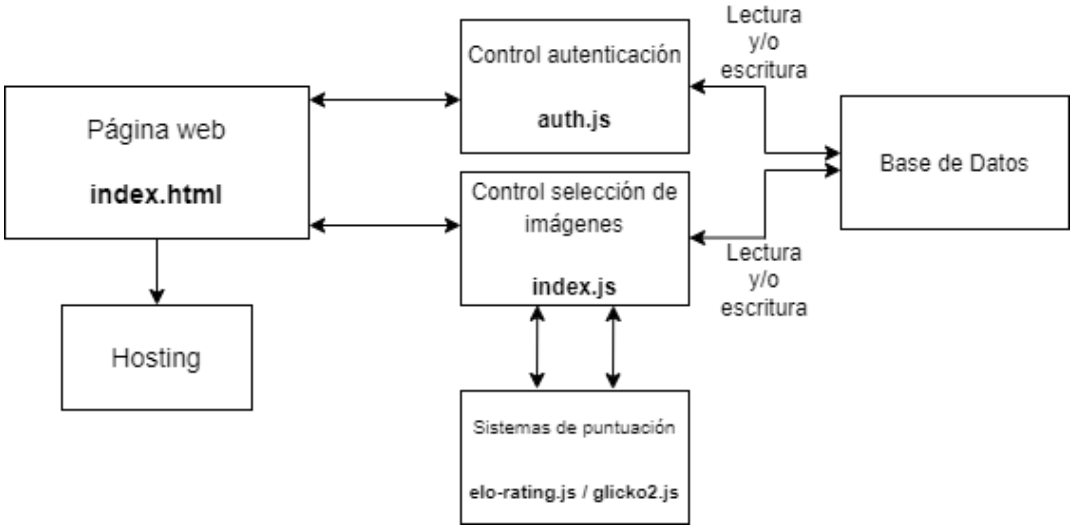

Figura 13. Estructura seguida en el desarrollo de la aplicación web

Como se puede observar, como página solamente se dispone del archivo index.html, en el que se realizan modificaciones en base al estado de la sesión, mientras que todo el código y las funcionalidades ocurren en los archivos js. También se añade el *hosting* y la base de datos mediante Firebase, a los cuales es necesario acceder para aplicar los cambios y obtener información que se utiliza en la página (URL para representar las imágenes, puntuaciones de cada una...).

En los siguientes apartados se procederá a entrar en detalle en todos estos aspectos.

4.2 Configuraciones de Firebase

Como se ha explicado en apartados anteriores, se ha utilizado la herramienta Firebase de Google para realizar el back-end de nuestra aplicación y actuar de apoyo a la misma. Esto ha incluido, en nuestro caso, el *hosting* de la página web para garantizar el acceso a ella desde Internet, la base de datos donde se almacena la información que se ha usado para la página (usuarios, imágenes, puntuaciones, enfrentamientos) y la autenticación de los usuarios.

Para ello, se ha tenido que realizar ciertos procesos en la consola de la herramienta web.

En primer lugar, la creación de un proyecto de Firebase adaptado a lo que ellos llaman Plan Spark (gratuito) y que permite el uso de las soluciones necesarias para cubrir nuestras necesidades. Esto se ha realizado de manera simple entrando en su web, iniciando sesión con un usuario de Google y agregando un nuevo proyecto:

Figura 14. Selección de proyectos en Firebase

También existe la posibilidad de utilizar proyectos de prueba donde todas estas soluciones ya están implementadas y es más sencillo el desarrollo, ya que solamente se ha de modificar los archivos plantilla.

Una vez realizado este paso, es necesario integrar Firebase con el directorio local en el que se va a llevar a cabo el desarrollo del front-end y configurar las carpetas de forma que la herramienta de Google pueda tomar los archivos correctos y aplicarlos al *hosting* y a otras soluciones si es necesario.

Para ello, se ha seguido la documentación de Firebase que indica los pasos a seguir, que son los siguientes:

- Usar Firebase CLI, o interfaz de línea de comandos. Este programa se añade a la consola de comandos de Windows y permite ejecutar instrucciones.
- Configurar el direcorio de proyecto. Esto se realiza abriendo una consola de comandos en la carpeta donde se quiere desarrollar la página y ejecutando la instrucción `firebase init`. Posteriormente se ha de seguir los pasos para enlazar el directorio al proyecto previamente creado y que esté todo identificado.
- Realizar las modificaciones en el front-end.
- Implementar los cambios en el sitio. Se realiza mediante el comando `firebase deploy`. Pasados unos minutos ya se pueden visualizar los cambios en la web.

Por supuesto, entre todos estos casos aparece la configuración del sitio web, como el nombre (en nuestro caso es https://tfm-test-1-220321.web.app).

También se ha de tener en cuenta la estructura de los archivos. En nuestro caso se ha seguido la siguiente:

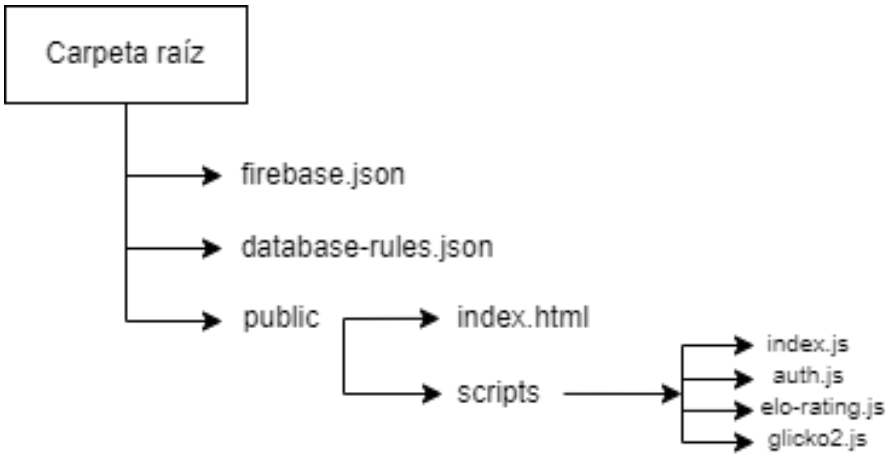

Figura 15. Estructura de archivos en local

Como se aprecia en el esquema, se tiene una carpeta llamada *public* en la raíz del repositorio Firebase, dentro de la cual se encuentra la página html llamada index, que sirve de base y que es la que se representa en la web. De forma paralela en esta página tenemos la carpeta donde guardamos los scripts y donde se puede acceder desde html de manera normal. Estos scripts los carga el navegador al cargar la página y ejecuta su código.

En la misma raíz donde se encuentra la carpeta *public* se sitúan los archivos que utiliza Firebase para llevar a cabo el hosting de la aplicación. El más importante es el fichero *firebase.json* en el cual se le indica a la herramienta la carpeta que ha de subir al servidor para representar la página, en nuestro caso la carpeta *public*, así como las reglas a seguir para la base de datos.

De forma paralela al *hosting*, se ha utilizado también la base de datos Firestore Database y la Autenticación interna de Firebase.

Ambas herramientas se encuentran enlazadas, pero la autenticación no se ha realizado en el propio código y almacenando las contraseñas en base de datos (como podría haberse hecho), sino que se ha usado la aplicación propia de Firebase que facilita el proceso y guarda las contraseñas de forma segura y respetando la privacidad de los usuarios.

La Autenticación se controla mediante la siguiente herramienta:

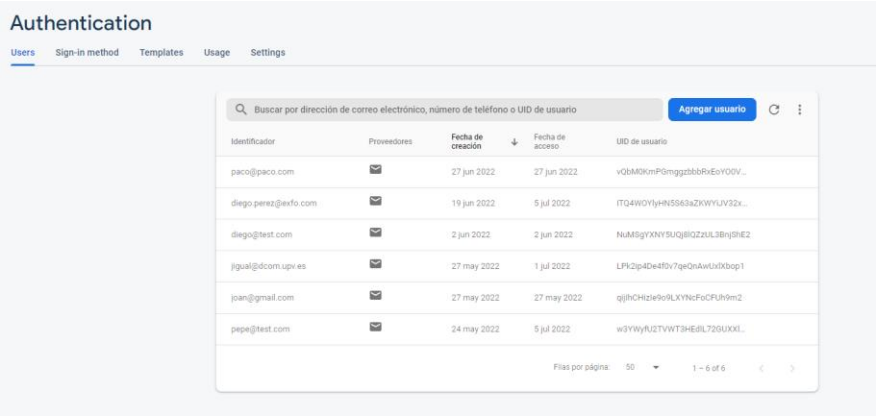

Figura 16. Página de gestión de autenticación manual en Firebase

Se observa que el panel de control de la misma permite observar los usuarios creados con su correo electrónico, fechas de creación/acceso y el identificador de usuario que se añade a la base de datos. Desde este panel se permite realizar mantenimientos tales como eliminar cuentas o restablecer contraseñas.

Cabe destacar las distintas opciones que se permiten a la hora de registrarse, en este caso solamente se permite el uso del correo electrónico pero también es posible usar proveedores tales como Google o Github, así como el número de teléfono. Estos sistemas son bastante usados en la actualidad y es interesante de cara a mejorar el proyecto.

4.3 Base de datos

Como se ha expuesto en el apartado anterior, se ha utilizado la base de datos de Firestore con el objetivo de almacenar la información necesaria de las imágenes, usuarios y enfrentamientos con el fin de implementar la aplicación.

El primer objetivo a cumplir en lo relativo a la base de datos era el propio diseño, es decir, tomar los requisitos que teníamos y crear un esquema de base de datos para poder seguirlo y realizar esta creación en la web de Firebase. El esquema ideado es el siguiente:

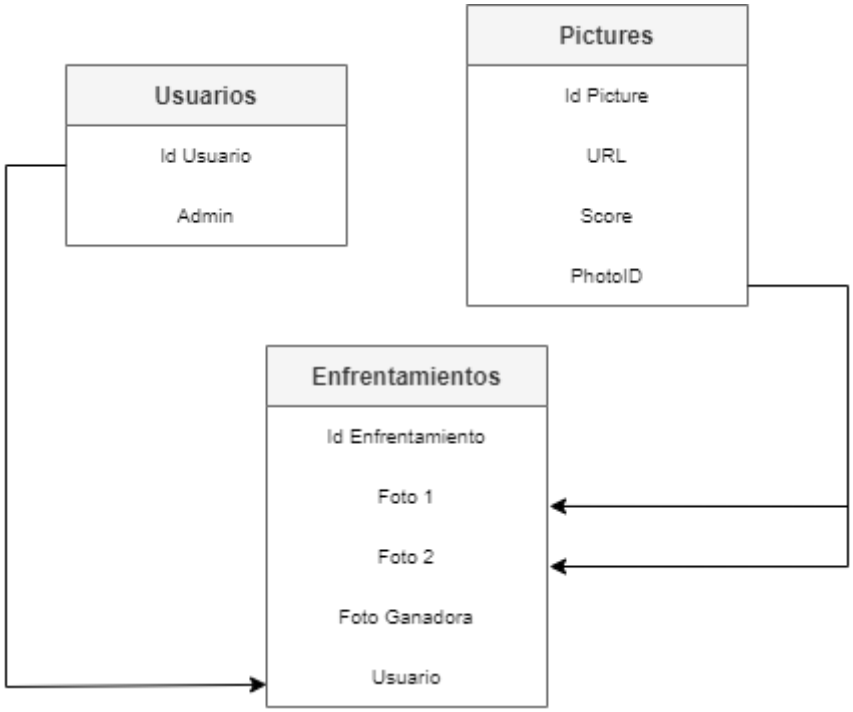

Figura 17. Estructura de la base de datos y relación entre las tablas

Como se puede observar en el esquema, la base de datos consta de tres tablas:

- Usuarios: se crea una entrada por cada usuario que se registra en la aplicación. El único añadido en este caso es el campo *admin* que controla si el usuario puede insertar fotografías o no. Este campo solamente puede cambiarse desde la herramienta Firestore Database, con la idea de que esté protegido y solo la persona al cargo pueda controlarlo.
- Pictures: es la tabla que incluye los datos de las imágenes. Cada imagen tiene tres campos: URL (donde se guarda la dirección en la que se puede encontrar la imagen en la red), score (puntuación que tiene la imagen dentro del proceso, como ya se ha comentado se comienza en 1500 y varía según si la imagen gana o pierde enfrentamientos) y photoId (es otro identificador que se le asigna a cada imagen y el cual ayuda a la hora de realizar

la selección, ya que los identificadores normales son útiles en el caso de ser únicos pero no ayuda su formato, alfanumérico y de un gran número de caracteres).

- Enfrentamientos: esta última tabla incluye los datos de enfrentamientos y se crea una nueva entrada cada vez que el usuario selecciona una imagen u otra. Los campos que incluye son las dos fotos que participaban en el encuentro y el id del usuario que ha elegido una u otra. La funcionalidad de esta tabla es la de controlar que un mismo usuario no repita enfrentamiento, es decir, que no se le pueda presentar la misma pareja de imágenes a un usuario más de una vez.

Para crear esta base de datos solamente se han tenido que seguir los pasos indicados en la documentación de Firebase, con la salvedad de haber tenido que modificar las reglas de uso para poder acceder a las tablas desde el navegador por parte de los *scripts*. Esto ocurre porque el caso por defecto, para lograr una mayor seguridad, restringe los accesos a la base de datos por parte del código base.

En nuestro caso, el código añadido a las reglas de uso ha sido el siguiente:

```
// match logged in user doc in users collection
match /users/{userId} {
  allow create: if request.auth.uid != null;
  allow read: if request.auth.uid == userId;
}

match /pictures/{pictureId} {
  allow read, write: if request.auth.uid != null; // para que tengan que estar autenticados
}

match /enfrentamientos/{enfrentamientoId} {
  allow read, write: if request.auth.uid != null; // para que tengan que estar autenticados
}
```

Figura 18. Reglas de uso de la base de datos de Firebase

El código añadido permite que solamente si existe una sesión iniciada se puedan realizar consultas y escrituras en la base de datos. Esto es útil en especial para evitar ataques externos e instrucciones no deseadas, permitiendo solamente ejecutar el código que nosotros hemos programado.

En el siguiente apartado se explicará también cómo se accede a la base de datos desde el código y los distintos usos que se le han dado a las tablas aquí expuestas.

4.4 Desarrollo software

En este apartado se va a explicar, paso a paso, cómo se ha llevado a cabo el desarrollo software de la aplicación. Para ello, se ha decidido que la mejor manera es dividir la explicación en los diferentes archivos de los que consta la misma.

4.4.1 Index.html

En primer lugar tenemos la propia página, en formato html. La idea inicial y que se ha tratado de seguir en todo momento es la de tener una página simple, con el mínimo de elementos posibles para mantener un buen rendimiento y la vista poco cargada, sin muchos recuadros ni colores.

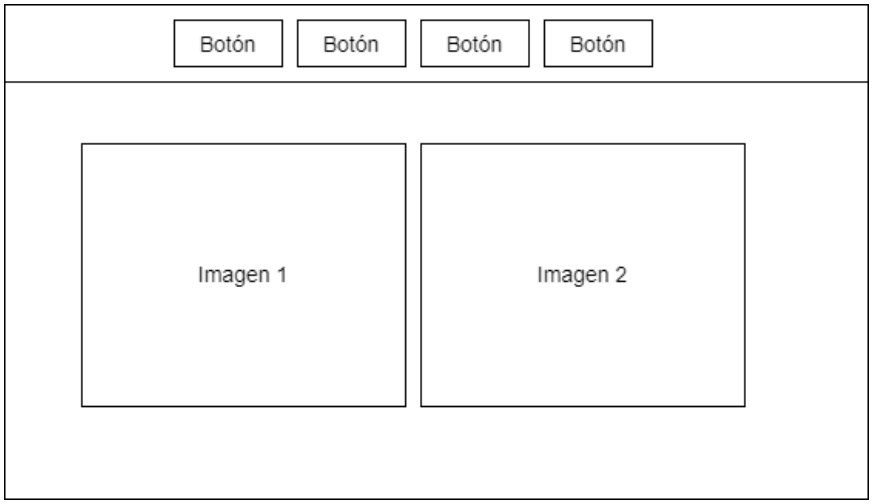

Figura 19. Boceto inicial de la página web

Para ello, se creó este esquema que se muestra en la figura y mediante el cual nos hemos podido centrar en crear 3 diferentes estructuras o contenedores, uno para la cabecera (donde poner el logotipo de la Universidad, así como algún título si era necesario), otro para los botones que controlarán la funcionalidad de la aplicación (y que van dentro de la misma cabecera a su vez), y finalmente un contenedor para el propio cuerpo donde irán las imágenes que participen en el enfrentamiento. En el esquema se muestran solamente dos imágenes pero el funcionamiento es el mismo para el caso de 9 fotografías.

Como cabecera del propio fichero html tenemos el siguiente código:

```
<html lang="es">
<head>
  <meta charset="UTF-8">
  <!-- Compiled and minified CSS -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
  <title>TFM: Diego</title>
</head>
```

Figura 20. Cabecera del fichero index.html

Se puede observar que es bastante simple, ya que indicamos el idioma mediante la propiedad ‘lang’, el set de caracteres que se pretende utilizar (UTF-8, es simple y nos aporta los caracteres necesarios para escribir todo el contenido de la página en castellano), una llamada a la librería CSS Materialize, necesario para poder tener acceso a todas sus características (como quedó explicado en apartados anteriores) y finalmente el título de la página mediante la etiqueta ‘title’. Éste es el título que aparece en la parte superior de la ventana del navegador.

Pasando ya al diseño del contenido de la propia página, se ha incluido todo en el elemento ‘body’. Este elemento es único en cualquier documento html y contiene toda la información.

```
<body class="grey darken-4">

  <!-- Cabecera -->
  <nav class="z-depth-0 grey lighten-2">
    <div class="nav-wrapper container">
      <a href="#" class="brand-logo">
        <img src="img/logo.svg" style="width: 180px; margin-top: 2.5px;">
      </a>
```

Figura 21. Barra superior de la página web, fichero index.html

En esta figura vemos la parte del código que abre el elemento <body>, así como el contenedor de la cabecera. Lo primero a destacar sería la clase asociada al elemento <body>, que quizás llama la atención. Esto no es algo estándar de html sino que nos lo aporta la librería CSS importada (Materialize).

Una forma de aplicarla es aplicarle clases a los distintos elementos, como en este caso el color que se le da al fondo de la página y que corresponde con el "*grey darken-4"*. De la misma manera funciona el color de la cabecera, que es bastante más claro y consiste en el "*grey lighten-2"*, además de la instrucción "*z-depth-0*" mediante la cual se añade una ligera sombra.

Además se puede observar el añadido del logo de la Universidad mediante una etiqueta simple de imagen o <img>, mientras que usando CSS básico se le da un tamaño de 180 píxeles (de forma que pueda verse siempre aun usando un dispositivo móvil) y cierto margen al límite superior. También cabe destacar que se le añade un enlace con una almohadilla, lo cual consigue que si se clica en la imagen se redirige a la misma página al usuario, es decir, se recarga.

El resultado conseguido con este trozo de código es el siguiente:

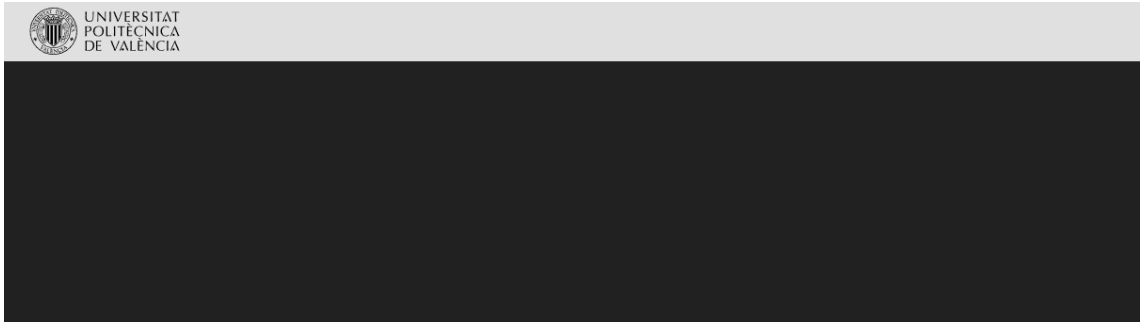

Figura 22. Aspecto de la página web solo con cabecera y fondo vacío

En esta figura se observa que de momento se cumple con lo decidido en el esquema inicial, pero nos faltan dos elementos más: los botones y las imágenes.

Para los botones tenemos que crear una lista en el mismo contenedor que incluye la cabecera, ya que queremos situarlos en esa zona de color gris claro.

```
<ul id="nav-mobile" class="">
    <ul style="display: flex; justify-content: center;">
      <li class="logged-in" style="display:none">
        <a href="#" class="waves-effect waves-light btn" id="logout" style="margin-right: 10px; margin-top: 10%;">Logout</a>
      </li>
      <li class="logged-in" style="display:none">
        <a href="#" class="waves-effect waves-light btn hide-on-med-and-down" id="more_pictures" style="margin-left: 10px; margin-top: 10%; margin-right: 10px;">NºFotos</a>
      </li>
      <li class="logged-in" style="display:none">
        <a href="#" class="waves-effect waves-light btn hide-on-med-and-down" id="export" style="margin-left: 10px; margin-top: 10%; margin-right: 10px;">Exportar</a>
      </li>
      <li class="logged-in" style="display:none">
        <a href="#" class="waves-effect waves-light btn modal-trigger" data-target="modal-insert" id="insert_pictures" style="margin-left: 10px; margin-top: 10%;">Insertar
      </li>
      <li class="logged-out" style="display:none">
        <a href="#" class="waves-effect waves-light btn modal-trigger" data-target="modal-login" style="margin-right: 10px; margin-top: 10%;">Login</a>
      </li>
      <li class="logged-out" style="display:none">
        <a href="#" class="waves-effect waves-light btn modal-trigger" data-target="modal-signup" style="margin-left: 10px; margin-top: 10%;">Sign Up</a>
      </li>
    </ul>
```

Figura 23. Lista de botones a situar en la barra superior de la web, fichero index.html

En esta figura se indica la parte de código con el que buscamos dicho objetivo. Pero hemos de tener en cuenta que tenemos una sola página tanto para el caso en el que el usuario ha iniciado sesión como para el que no lo ha hecho, así que tenemos que generar los botones y mostrar solamente los que deban estar presentes según esta situación.

Para ello se ha creado una lista no ordenada (elemento <ul> de html) a la que se le han añadido un total de 6 elementos de lista o <li>, cuatro de los cuales nos servirán cuando el usuario esté dentro de la sesión y dos cuando no. Con vistas a esto, lo cual solo se podrá conseguir mediante el uso de un script de JavaScript, se han creado estos elementos con dos clases distintas, *"logged-in"* y *"logged-out"*, de forma que quede claro qué botones mostrar en cada situación.

Aparte de ello, se le ha colocado a cada botón un identificador, para poder seleccionarlos desde los scripts a la hora de aportarles una funcionalidad, y ciertas clases que, de la misma manera que los colores de la cabecera y fondo de la página, utilizan Materialize CSS para crear un diseño simple. En este caso, se utilizan los botones "*waves-effect waves-light btn*", y mediante los estilos se colocan de forma equidistante entre ellos y centrados en la parte superior de la página. El resultado es el siguiente:

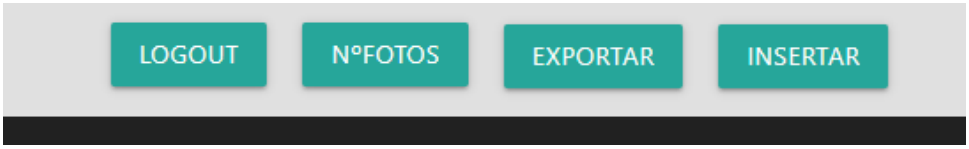

Figura 24. Botones presentados con sesión iniciada

Figura 25. Botones presentados sin sesión iniciada

Una vez conseguido esto, la última parte necesaria para cumplir con el esquema inicial es el contenedor de las imágenes. Es también la parte más simple del código HTML ya que la mayor parte del trabajo se realiza en los scripts.

```
<!-- Lista de imágenes -->
<div id="imagen" class="container" style="margin-top:30px; color: white; object-fit: cover;">

</div>
```

Figura 26. Contenedor html para imágenes, fichero index.html

Consta solamente de un elemento (al cual se le pueden añadir subelementos desde los scripts) y ciertos parámetros de estilo que nos servirán para el texto alternativo, es decir, cuando no haya imágenes que presentar (debido a que la sesión de usuario no esté iniciada y por tanto se encuentre fuera del sistema). El resultado es el siguiente:

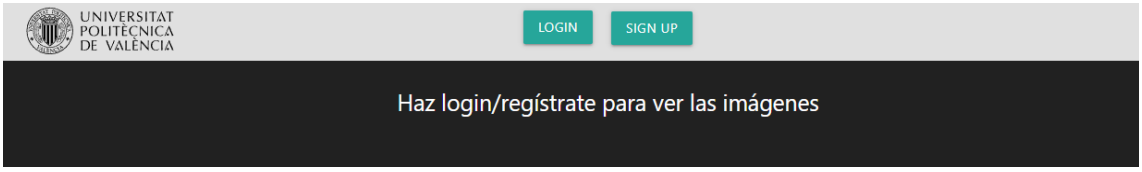

Figura 27. Presentación de botones y texto previa a iniciar sesión

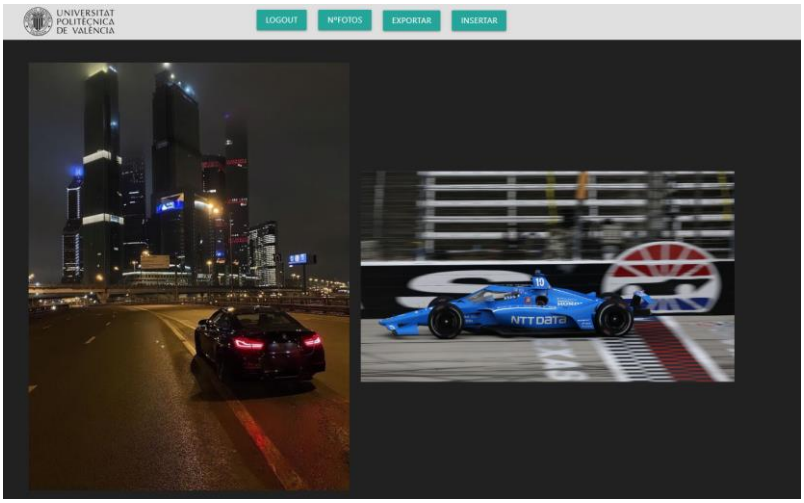

Figura 28. Presentación de botones e imágenes tras iniciar sesión

Para finalizar con este fichero, además de los elementos considerados en el esquema inicial, también es necesario crear otra forma de interactuar con la página, ya que determinadas funcionalidades así lo requieren. Concretamente, registrar un usuario, iniciar sesión e insertar fotos. En los tres casos se requiere de un formulario bastante similar (solamente cambia el número de campos necesarios) y se crea de la siguiente manera:

```
<!-- Formulario de Registrarse -->
<div id="modal-signup" class="modal">
  <div class="modal-content">
    <h4>Sign up</h4><br/>
    <form id="signup-form">
      <div class="input-field">
        <input type="email" id="signup-email" required />
        <label for="signup-email">Email address</label>
      </div>
      <div class="input-field">
        <input type="password" id="signup-password" required />
        <label for="signup-password">Choose password</label>
      </div>
      <button class="btn yellow darken-2 z-depth-0">Sign up</button>
      <p class="error pink-text center-align"></p>
    </form>
  </div>
</div>
```

Figura 29. Implementación en html del formulario de registro, fichero index.html

Se utiliza la clase modal (también proveniente de Materialize) que crea una ventana popup o de diálogo que aparece encima de la página, a la que se le añade un formulario (nativo de html) con dos campos: usuario y contraseña. La mayor peculiaridad es que ambos campos tienen su propia clase ya que necesitan que, en el caso del usuario ha de introducirse un correo electrónico (si el formato no concuerda salta error) y en el de la contraseña ha de tener más de 6 caracteres y no se muestra el valor sino que se oculta.

Ambos valores son requeridos por el formulario y el resultado es el siguiente:

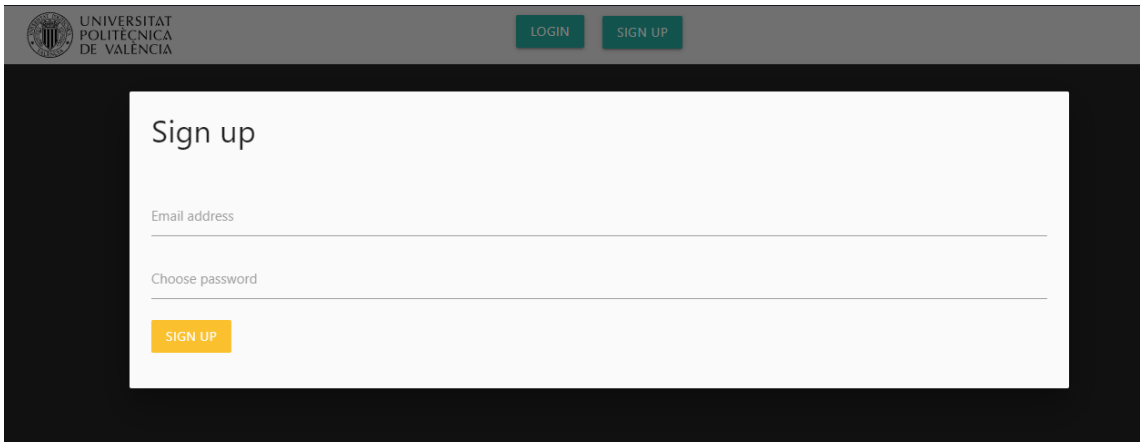

Figura 30. Formulario de registro en página web

Para el caso del formulario de iniciar sesión el procedimiento es exactamente el mismo, solamente cambiando las clases para poder identificarlo en los scripts a la hora de implementar las funcionalidades que veremos más adelante:

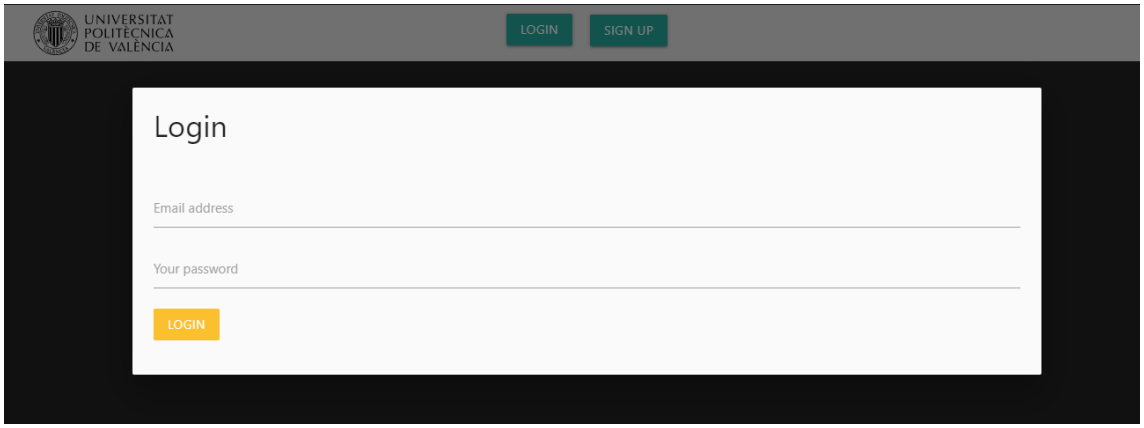

Figura 31. Formulario de inicio de sesión en página web

En cuanto al último caso, es aún más simple, ya que solamente requiere de un campo de texto simple, por lo que funciona del mismo modo pero sin necesidad de campos especiales ni requerimientos extra como en estos dos casos.

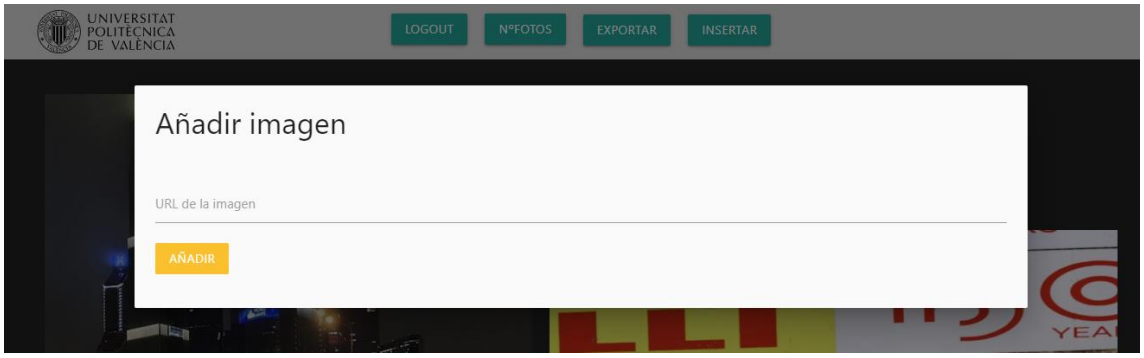

Figura 32. Formulario de inserción de imagen en página web

Para finalizar con el código HTML, es necesario un último paso. Todos estos contenedores y formularios que se han explicado en este apartado no implementan una funcionalidad por sí misma, se han nombrado varias veces los famosos scripts. Para que los scripts tengan acceso a los elementos html y puedan modificarlos, se han de ejecutar en el navegador, y para ello se usa la etiqueta <script> de html. En nuestro caso tenemos dos partes:

```
<script src="https://www.gstatic.com/firebasejs/5.6.0/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/5.6.0/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/5.6.0/firebase-firestore.js"></script>
<script>
  // Your web app's Firebase configuration
  // Initialize Firebase
  var firebaseConfig = {
    apiKey: "AIzaSyAC8-MxXevDBnKlIqBHwjqxV_dvjZ3mN4k",
    authDomain: "tfm-test-1-220321.firebaseapp.com",
    databaseURL: "https://tfm-test-1-220321-default-rtdb.europe-west1.firebasedatabase.app",
    projectId: "tfm-test-1-220321",
  };
  firebase.initializeApp(firebaseConfig);

  // make auth and firestore references
  const auth = firebase.auth();
  const db = firebase.firestore();


  // update firestore settings
  db.settings({ timestampsInSnapshots: true });

</script>
```

Figura 33. Llamada en html de los scripts propios de Firebase, fichero index.html

En primer lugar, la configuración y scripts relacionados con Firebase. Mediante esta parte del código el navegador es capaz de gestionar tanto la autenticación como la base de datos de la herramienta de Google, y se crean los objetos auth y db que se utilizarán más adelante. Se puede observar que se utilizan los scripts relacionados con la versión 5.6.0 y añadimos nuestras claves de forma que autenticamos a nuestra página dentro de nuestro proyecto.

Esta parte ha sido directamente extraída de la documentación de Firebase, y es igual para todos los proyectos de este tipo, el único cambio posible sería la adición de otros objetos en el caso de usar más herramientas de las que ofrece la página.

En segundo lugar, añadimos los scripts propios donde hemos implementado el resto del código y que son los que alterarán la página html con las funcionalidades.

```
<!-- Scripts propios JavaScript -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
<script src="scripts/auth.js"></script>
<script src="scripts/index.js"></script>
<script src="scripts/elo-rating.js"></script>
<script src="scripts/glicko2.js"></script>
```

Figura 34. Llamada en html del resto de scripts, fichero index.html

Aquí se encuentran todos los scripts relacionados con la autenticación (y que también controla lo que se muestra en pantalla según el estado de la misma), con las funcionalidades de las imágenes y con los sistemas de clasificación que utilizamos para las mismas.

En los siguientes apartados se explican cada uno de ellos en profundidad.

4.4.2 Auth.js

En este script es donde se han implementado las funciones relacionadas con la propia autenticación, como el nombre del fichero indica. Esto incluye registrar el usuario, iniciar/cerrar sesión y también lanzar los cambios correspondientes cuando la autenticación cambia, es decir, siempre que se detecta un usuario iniciando/cerrando sesión.

Esto último es importante dado que es lo primero que se ejecutará al entrar en la página.

```
// Cambios a realizar cada vez que se detecta un cambio en la autenticación (login/logout)
auth.onAuthStateChanged(user => {
    if (user) {
        // Obtener datos de la BD e inicializar la página
        db.collection('pictures').onSnapshot(snapshot => {          //get().then() habria que recargar. onSnapshot actualiza la info él solo
            img = 2;        // Para mostrar siempre 2 fotos como inicio al hacer login - Valor por defecto
            setupPictures(snapshot.docs, img);
            setupUI(user);
        }, err => {
            console.log(err.message)
        });
    } else {
        setupUI();
        setupPictures([]);
    }
});
```

Figura 35. Función onAuthStateChanged, fichero auth.js

Esta imagen corresponde precisamente con ese caso. Siempre que la página detecte un cambio en el estado de autenticación del usuario, ya sea por abrir la página, iniciar/cerrar sesión, o registrarse, se ejecutará esta función.

Las instrucciones aquí escritas consisten en llamar a la base de datos de imágenes y obtener lo que se llama un *snapshot* de la colección. Con esta *snapshot* se podrá, más adelante, realizar consultas y peticiones a la base de datos, así que éste es el primer paso que hay que llevar a cabo.

Una vez obtenido este *snapshot*, y considerando que todas las operaciones de llamada a base de datos realizadas en JavaScript corresponden a operaciones asíncronas, una vez obtenido llamamos a las funciones correspondientes a preparar el *layout* de la página, en este caso 'setupPictures' y

'setupUI'. Se puede apreciar la diferencia en la forma de la llamada en base a si existe un usuario autenticado o no. En el primer caso se envía a las funciones los datos como argumentos, de forma que se presente la página con las imágenes deseadas, y en el segundo caso se envía sin argumentos.

La siguiente función implementada es la de registrar un usuario:

```
// Registrar un usuario
const signupForm = document.querySelector('#signup-form');
signupForm.addEventListener('submit', (e) => {
    e.preventDefault();

    // Obtenemos info del formulario
    const email = signupForm['signup-email'].value;
    const password = signupForm['signup-password'].value;

    // Registramos el usuario en Firebase
    auth.createUserWithEmailAndPassword(email, password).then(cred => {
        return db.collection('users').doc(cred.user.uid).set({
            admin: false
        });
    }).then(() => {
        // Vaciar el formulario (por si se vuelve a abrir) y cerrarlo
        const modal = document.querySelector('#modal-signup');
        M.Modal.getInstance(modal).close();
        signupForm.reset();
        signupForm.querySelector('.error').innerHTML = '';
    }).catch(err => {
        signupForm.querySelector('.error').innerHTML = err.message;
    });
});
```

Figura 36. Función de registro de usuario, fichero auth.js

Esta función es bastante diferente a la anterior por distintos motivos.

El primero de ellos es que en este caso hay involucrado un elemento html. Para utilizarlo, tenemos que escribir la primera instrucción que aparece en la figura, y que se ocupa de identificar este elemento mediante su clase y guardarlo en una constante de JavaScript que puede utilizarse después.

Para utilizarlo, le añadimos un *Listener*, que no es otra cosa que un objeto que provoca la ejecución de cierta función cuando ocurre el evento que está esperando.

Se puede observar que el primer comando escrito dentro de este *Listener* es 'preventDefault'. Su utilidad es la de cancelar cualquier tipo de evento que el elemento pueda tener asociado, y de esta manera evitar que realice algún cambio inesperado para nosotros, ya que vamos a programar en las siguientes líneas nuestro código y es importante tenerlo controlado.

El resto es simple, ya que tomamos los datos introducidos por el usuario en el formulario de registro y utilizamos las instrucciones propias de la autenticación que aporta Firebase en su objeto 'auth'. En este caso '*createUserWithEmailandPassword'*, la cual añade los datos a la base de datos directamente y controla la autenticación en la propia página.

Esta función también permite realizar más pasos una vez concluido el registro, y nosotros lo utilizamos para añadir otro campo al usuario, el campo de 'admin' que podemos modificar después a mano en la propia base de datos, y finalmente vaciar el formulario y cerrar el popup para mantenernos preparados en el caso de que se cierre la sesión y se quiera utilizar el formulario otra vez. Si no se añadieran estos pasos, y entendiendo que no se refresca la página en ningún momento, al abrir otra vez el formulario aparecerían los datos previamente escritos.

En el caso de iniciar sesión funciona exactamente de la misma manera:

```
// Iniciar sesión de usuario
const loginForm = document.querySelector('#login-form');
loginForm.addEventListener('submit', (e) => {
    e.preventDefault();

    // Obtener info del formulario
    const email = loginForm['login-email'].value;
    const password = loginForm['login-password'].value;

    // Iniciar sesión en Firebase
    auth.signInWithEmailAndPassword(email, password).then(cred => {
        // Vaciar el formulario (por si se vuelve a abrir) y cerrarlo
        const modal = document.querySelector('#modal-login');
        M.Modal.getInstance(modal).close();
        loginForm.reset();
        loginForm.querySelector('.error').innerHTML = '';
    }).catch(err => {
        loginForm.querySelector('.error').innerHTML = err.message;
    });
});
```

Figura 37. Función de inicio de sesión, fichero auth.js

El único cambio es el uso de la función '*signInWithEmailAndPassword'* en este caso.

Algo similar ocurre en el caso de cerrar sesión, salvo porque en este caso no se necesita ningún argumento ni realizar ninguna acción después, solamente cerrar la sesión con la función nativa '*signOut'*:

```
// Cerrar sesión de usuario
const logout = document.querySelector('#logout');
logout.addEventListener('click', (e) => {
    e.preventDefault();
    auth.signOut();
});
```

Figura 38. Función de cierre de sesión, fichero auth.js

4.4.3 Index.js

En este apartado se explica el contenido del fichero index.js, que realmente es donde se encuentra el grueso de la programación del proyecto, ya que controla básicamente todo el funcionamiento de la página y de sus procesos, desde la presentación de la misma hasta las acciones realizadas por los botones.

En primer lugar tenemos las funciones que preparan la página y que se llaman desde la autenticación, como se ha visto en el apartado anterior.

- setupUI:

```
const setupUI = (user) => {
  if(user) {
    db.collection('users').doc(user.uid).get().then(doc => {

      // Obtener estatus de administrador del usuario
      var admin = doc.data().admin;

      // Sacamos botones correspondientes a usuario logeado
      loggedInLinks.forEach(item => item.style.display = 'block');
      loggedOutLinks.forEach(item => item.style.display = 'none');
      // En caso de ser administrador le damos la opción de insertar imágenes
      if (admin){
        loggedInAdminLinks.forEach(item => item.style.display = 'block');
      } else {
        loggedInAdminLinks.forEach(item => item.style.display = 'none');
      }
    })

  } else {
    // Sacamos botones correspondientes a usuario no logeado
    loggedInLinks.forEach(item => item.style.display = 'none');
    loggedInAdminLinks.forEach(item => item.style.display = 'none');
    loggedOutLinks.forEach(item => item.style.display = 'block');
  }
}
```

Figura 39. Función de presentación de la interfaz de usuario, fichero index.js

Como se aprecia en la captura, se realiza un acceso a la base de datos para comprobar si el usuario que se nos pasa en el argumento de la función existe (para saber si se ha iniciado sesión o no) y le colocamos a cada elemento (en este caso botones superiores) el estilo de 'block' o 'none' según si se ha de mostrar o no.

También es importante el punto en el que si el usuario es administrador (definido manualmente en la base de datos) se le muestra un botón más que al resto, que es el que permite insertar nuevas imágenes en la BD.

- setupPictures: se encarga de preparar la presentación de las imágenes en pantalla, el enfrentamiento en sí.

Es una de las funciones más importantes de la aplicación y se ocupa de realizar muchas cosas. Lo primero que se realiza es comprobar si los datos recibidos en la llamada existen o están vacíos, lo cual nos daría pie a saber si hay que presentar imágenes o pedir al usuario que inicie sesión, respectivamente.

Una vez realizada esta comprobación, y en el caso de que la sesión esté iniciada, pasamos a la presentación. Para ello, tendremos que calcular el número de filas y de imágenes por fila que tendremos, y es lo primero que se realiza. Tenemos dos opciones: 2 imágenes o 9. En el primer caso, tendremos solo una fila con dos imágenes, y en el segundo tendremos tres filas de tres imágenes.

```
db.collection('pictures').orderBy("photoId","desc").limit(1).get().then(function(querySnapshot) {
  querySnapshot.forEach(function(doc) {
      id = doc.data();
      numFotos = id.photoId;
      rnd1 = selectPictures(img, numFotos);
  })
});
```

Figura 40. Inicio de la función de presentación de imágenes en pantalla, fichero index.js

El primer paso, como se muestra en la captura, es realizar la selección de las imágenes que se van a mostrar. Para ello tenemos que hacer un primer acceso a base de datos, de donde obtenemos el identificador más alto, el cual actúa en este caso de contador de imágenes dentro de la BD, ya que lo necesitamos para seleccionar las imágenes.

Una vez realizado esto pasamos a la función que las selecciona:

```
// Funcion para obtener los indices de las fotos a sacar. La primera imagen es random y la segunda no.
function selectPictures(size, num) {
  if (size == 9){
    let ind = Array.from({length: size}, () => Math.floor(Math.random() * num));
    while (ind.some(noEsPrimero)){
      ind = Array.from({length: size}, () => Math.floor(Math.random() * num));
    }
  } else {
    let ind = [];
    ind[0] = Math.floor(Math.random() * num);
    db.collection('pictures').where("photoId", "==", ind[0]).get().then(function(querySnapshot) {
      querySnapshot.forEach(function(doc) {
        let score = doc.data().score;
        var grupo = 0;
        switch (true) {
          case score <= 1200:
            grupo = 1;
            break;
          case score > 1200 && score <= 1500:
            grupo = 2;
            break;
          case score > 1500 && score <= 1800:
            grupo = 3;
            break;
          case score > 1800 && score <= 2100:
            grupo = 4;
            break;
          default:      // Caso superior a 2100
            grupo = 5;
            break;
        }
        // Sacamos todas las fotos correspondientes de ese grupo
        let arrayImgs = [];
        arrayImgs = obtenerImgsGrupo(grupo);
        // Obtenemos imagen aleatoria de ese array
        var rand = Math.floor(Math.random()*arrayImgs.length);
        ind[1] = arrayImgs[rand];
      })
    });
  }
  return ind;
```

Figura 41. Función de selección de imágenes de BD, fichero index.js

En esta parte es donde se añade la complejidad, ya que se ha de cumplir que la imagen no se repita y que el enfrentamiento sea interesante, equilibrado. Esto se ha implementado mediante la idea de los grupos, donde cada imagen se introduce en un grupo según su rating, y una vez elegida una imagen (la primera es aleatoria siempre), la segunda ha de pertenecer al mismo grupo.

Es importante explicar que en el caso de 9 imágenes, la selección de las mismas siempre es aleatoria, ya que sería muy costoso y complejo gestionar a la vez la presentación de nueve imágenes y que sean del mismo grupo.

Se observa en la captura cómo se toma la primera imagen de forma aleatoria, se accede a su entrada en la BD para conocer su puntuación, mediante la cual la asignamos a un grupo o a otro. Finalmente llamamos a la función correspondiente que obtiene un array con las imágenes que pertenecen a dicho grupo, de entre las cuales seleccionaremos aleatoriamente una:

```
function obtenerImgsGrupo(grupo){
  let limites = [];   // limites[0] será el valor inferior y limites[1] el superior
  switch (grupo) {
    case 1:
      limites[0] = 0;
      limites[1] = 1200;
      break;
    case 2:
      limites[0] = 1201;
      limites[1] = 1500;
      break;
    case 3:
      limites[0] = 1501;
      limites[1] = 1800;
      break;
    case 4:
      limites[0] = 1801;
      limites[1] = 2100;
      break;
    case 5:
      limites[0] = 2101;
      limites[1] = 100000;     // Valor muy alto para que no sea alcanzado
      break;
  }

  let arrayImgs = [];

  // Accedemos a base de datos y obtenemos los id de las imágenes que cumplan con la restricción de grupo
  db.collection('pictures').where("score", ">=", limites[0]).where("score", "<=", limites[1]).get().then(function(querySnapshot) {
    querySnapshot.forEach(function(doc) {
      let id = doc.data().photoId;
      arrayImgs.push(id);
    })
    return arrayImgs;
  })
}
```

Figura 42. Obtención de todas las imágenes que pertenecen a un grupo, fichero index.js

- checkEnfrentamientos: es la función que se ocupa de que las dos fotos a presentar (solamente se ejecuta en ese caso, ya que con 9 imágenes es muy complicado que dos fotos no se encuentren) no se hayan enfrentado previamente.

Se ejecuta posteriormente a la anterior, cuando ya tenemos los datos de las imágenes que han sido seleccionadas y el usuario al que se le va a presentar el enfrentamiento:

```
// funcion evitar enfrentamientos repetidos
function checkEnfrentamiento(fotos){

  let flag = false;

  // Buscamos enfrentamientos para ver si se repite el que ha salido
  // Si se repite, devolvemos true, si no false
  db.collection('enfrentamientos').where("foto1", "==", fotos[0]).get().then(function(querySnapshot) {
    querySnapshot.forEach(function(doc) {
        if (doc.data().foto2 == fotos[1]){
          flag = true
        }
    })
  })
  db.collection('enfrentamientos').where("foto1", "==", fotos[1]).get().then(function(querySnapshot) {
    querySnapshot.forEach(function(doc) {
      console.log("foto1 = "+doc.data().foto1 + ", foto2 = "+doc.data().foto2);
        if (doc.data().foto2 == fotos[0]){
          flag = true
        }
    })
  })
  return flag;
}
```

Figura 43. Comprobación de si el enfrentamiento seleccionado ya ha existido, fichero index.js

Realiza accesos a la tabla de enfrentamientos y comprueba que el actual no se encuentre ya registrado para dicho usuario.

Una vez elegidas las imágenes se trata de presentar dichas imágenes en la página, lo que se realiza modificando el código HTML desde el script con los siguientes bucles *for* anidados:

```
for (let step1 = 0; step1 < rows; step1++) {

  const ul1 = `
    <ul class="pictures" style="border: none; display: flex; justify-content: center; color: white;">
  `
  html += ul1;

  for (let step2 = 0; step2 < size; step2++) {
    let step = (3*step1)+step2;
    const picture = data[rnd1[step]].data();            // Aquí elegimos la imagen de la BD con los ID elegidos arriba
    const picId = data[rnd1[step]].id;
    // obtenemos score de fotos mostradas y almacenamos en array
    scoreArray[step] = picture.score;
    // presentamos fotos por pantalla
    let li = ``;
    // Creamos el html con las imágenes
    if (rows == 1){
      li = `
        <li  style="margin: 10px;">
          <img id="image${step}" src="${picture.url}" alt="${picId}" style="width:100%; height:100%; max-width:70vh; max-height:40vw; object-fit:scale-down;"/>
        </li>
      `;
    } else {
      li = `
        <li  style="margin: 10px; ">
          <img id="image${step}" src="${picture.url}" alt="${picId}" style="width:100%; height:100%; max-width:60vh; max-height:12.5vw; object-fit:scale-down;"/>
        </li>
      `;
    }
    html += li;
  }
  const ul2 = `
    </ul>
  `
  html += ul2;
}

pictureBox.innerHTML = html;        // Añadimos el html de las fotos al elemento de la página correspondiente
```

Figura 44. Creación de los elementos <img> en html, fichero index.js

El primer bucle recorre las filas y el segundo las imágenes dentro de cada fila. Por cada fila necesitaremos un elemento del tipo <ul> por lo que lo creamos en el primer bucle. Para cada imagen dentro de una fila usamos un elemento <li> con la información de la imagen.

De cada imagen nos interesan los siguientes campos: identificador, URL y rating o score. Por eso accedemos a base de datos y los obtenemos, para luego añadir el score al array correspondiente (que se utilizará más adelante para calcular los ratings en los enfrentamientos) y para escribir la URL y el identificador en los elementos <li> que presentaremos en el código HTML. Finalmente creamos el código HTML y lo concatenamos en la variable que luego se introduce en index.html.

Algo importante y que ha sido necesario modificar y testear bastante ha sido el ancho y alto de las imágenes, de forma que la presentación fuera correcta. Para ello se ha utilizado las propiedades *max-width* y *max-height* de CSS. Se ha llegado a la conclusión de que cuando se tienen dos imágenes el ancho de las mismas es el aspecto restrictivo, y en el caso de nueve imágenes el alto es el aspecto restrictivo. Con estas propiedades conseguimos dicho efecto, por lo que cuando tengamos dos imágenes ambas serán igual de anchas (manteniendo siempre la relación de aspecto, por lo que es posible que una sea más alta que la otra) y cuando se tengan nueve imágenes todas serán igual de altas.

Una vez presentadas las imágenes, el punto más importante es la ejecución cuando el usuario selecciona una u otra. No es complejo de captar este click, ya que, como se ha explicado con anterioridad, cualquier elemento HTML permiten asignar una función que “escuche” cuando esto ocurre:

```
// al clicar en imagen
img0.addEventListener('click', (e) => {
  e.preventDefault();

  // Actualizamos scores
  onPictureSelection(img0.alt, img1.alt, scoreArray, 0)

  // cargamos enfrentamiento con resultado a la BD
  updateEnfrentamientos(rnd1, userId);

});
```

Figura 45. Función activada al clicar en imagen, fichero index.js

Esto se tiene que incorporar para cada elemento (img0/img1 en el caso de 2 imágenes, el resto si hay 9). Resulta simple ya que consta de una función que calcula las puntuaciones según ganadora y perdedora y otra que coge este enfrentamiento y lo almacena en base de datos para aumentar la información y saber que no se puede repetir en posteriores selecciones.

- onPictureSelection(),

```
// Funcion para cuando cliques
function onPictureSelection(imgW, imgL, scoreArray, posW){
  if(scoreArray.length == 2){
    for (let step = 0; step < scoreArray.length; step++){
      if (step != posW) {
        posL = step;
      }
    }
    // actualizamos scores
    scoreW = scoreArray[posW];
    scoreL = scoreArray[posL];
    updateScore2(imgW, scoreW, imgL, scoreL);
  } else{
    // lo mismo pero para 9 imágenes
    updateScore9(imgW, scoreArray, posW);
  }
}
```

Figura 46. Actualización de puntuaciones de imágenes, fichero index.js

Como se aprecia en la figura, tenemos las dos opciones según el número de imágenes. En el primer caso, necesitaríamos el identificador y score de cada una de ellas (que es simple de obtener) mientras en el segundo caso trabajamos directamente con array de scores ya que hay hasta nueve imágenes implicadas.

Las funciones que se encargan por lo tanto de realizar la actualización del score mediante el sistema correspondiente y almacenar los nuevos valores en BD son las siguientes:

```
// Funcion para actualizar el score cuando hay 2 fotos
function updateScore2(id0, score0, id1, score1) {

  // Calculamos scores resultantes del enfrentamiento
  [score0_res, score1_res] = EloRating.calculate(score0, score1);

  // aumentamos score ganador
  db.collection('pictures').doc(id0).update({
    score: score0_res
  });

  // disminuimos score perdedor
  db.collection('pictures').doc(id1).update({
    score: score1_res
  });
}
```

Figura 47. Uso de Elo Rating y cambio en BD de los scores para 2 imágenes, fichero index.js

Para dos imágenes, simplemente le aplicamos el sistema elo-rating (explicado en el apartado siguiente) y tomamos los resultados para guardarlos en base de datos.

```
function updateScore9(idW, scoreArray2, idArray, posW){
  var scoreW = scoreArray2[posW];
  scoreArray2.splice(posW,1); // eliminamos ese valor del array
  idArray.splice(idW,1); // eliminamos también el id

  // Creamos los participantes con su score
  var winner = ranking.makePlayer(scoreW);

  // Ahora los no ganadores mediante un bucle for
  var losers = [];
  for (let step = 0; step < scoreArray2.length; step++){
    losers[step] = ranking.makePlayer(scoreArray2[step]);
  }
  var race = glicko.makeRace(
    [
      [winner],
      [loser[0], loser[1], loser[2], loser[3], loser[4], loser[5], loser[6], loser[7]]
    ]
  );

  ranking.updateRatings(race);

  // Actualizamos BD con los nuevos ratings
  db.collection('pictures').doc(idW).update({
    score: winner.getRating()
  });

  for (let step = 0; step < scoreArray2.length; step++){
    db.collection('pictures').doc(idArray[step]).update({
      score: losers[step].getRating()
    });
  }

}
```

Figura 48. Uso de Glicko y cambio en BD de los scores para 9 imágenes, fichero index.js

En cambio, para nueve imágenes tiene algo más de complejidad al trabajar con arrays y con el sistema Glicko 2, cuyo funcionamiento es menos intuitivo al añadir nuevas funciones.

Para trabajar con arrays primero tomamos el valor ganador y después lo eliminamos del mismo array, pasando a tener todos los perdedores en este último. Entonces ya seguimos los pasos correspondientes para preparar la función de Glicko (explicado en el apartado 4.4.5 de la memoria) y ejecutamos el enfrentamiento. Finalmente solo queda actualizar la base de datos con los valores resultantes.

- updateEnfrentamientos:

```
function updateEnfrentamientos(rnd, Id, posW){
  if(img == 2){
    db.collection('enfrentamientos').add({
      foto1: rnd[0],
      foto2: rnd[1],
      winner: rnd[posW],
      user: Id,
    }).catch(err => {
      console.log(err.message);
    });
  }
}
```

Figura 49. Adición de enfrentamiento nuevo a BD, fichero index.js

Esta función también resulta poco compleja ya que se trata solamente de un acceso a base de datos, donde se añade una nueva entrada con esos tres campos.

Otra de las opciones permitidas es la de insertar una nueva imagen en la base de datos directamente desde la interfaz, solamente para los usuarios marcados como administradores. Aporta comodidad ya que el sistema de Firebase es algo engorroso y evita confusiones al trabajar con identificadores aleatorios con caracteres alfanuméricos.

La implementación es la siguiente:

```
// insertar nueva imagen
const insertForm = document.querySelector('#insert-form');
insertForm.addEventListener('submit', (e) => {
    e.preventDefault();

    // obtener el photoId máximo para asignar el siguiente
    db.collection('pictures').orderBy("photoId","desc").limit(1).get().then(function(querySnapshot) {
        querySnapshot.forEach(function(doc) {
            id = doc.data();
            lastId = id.photoId;
            const nextId = lastId + 1;

            // añadir foto a BD
            db.collection('pictures').add({
                url: insertForm['url-img'].value,
                score: 1500,
                photoId: nextId
            }).then(() => {
                // Cerramos el formulario y lo vaciamos por si se vuelve a usar
                const modal = document.querySelector('#modal-insert');
                M.Modal.getInstance(modal).close();
                insertForm.reset();
            }).catch(err => {
                console.log(err.message);
            });

        })
    })
});
```

Figura 50. Inserción de nueva imagen en BD, fichero index.js

Consiste en un formulario (como se ha visto en apartados anteriores) en el cual introducimos la URL de la imagen en cuestión y que, al clicar en el botón de enviar, realiza concretamente dos llamadas a base de datos seguidas: primero una para obtener el valor máximo de identificador que tenemos (ya que tendremos que tomar este valor y sumarle 1 para insertarlo en la nueva imagen) y después otro acceso para crear la nueva entrada en la base de datos con los valores correspondientes, como se aprecia en la captura (la URL insertada, el score por defecto a 1500 y el identificador de imagen).

Finalmente se cierra el formulario y se vacía para el caso en el que se quiera volver a usar.

- Exportar resultados:

```
// al clickar el boton de export
const exportarResults = document.getElementById('export');
exportarResults.addEventListener('click', (e) => {
    // arrays para meter los resultados
    var resultadosURL = [];
    var resultadosScore = [];
    // numero de resultados configurable
    var numResultados = 10;

    db.collection('pictures').orderBy("score","desc").limit(numResultados).get().then(function(querySnapshot) {
        querySnapshot.forEach(function(doc) {
            // pasar datos a fichero y al acabar exportarlo
            id = doc.data();
            //console.log(id.photoId);
            resultadosURL.push(id.url);
            resultadosScore.push(id.score);
        })
    // funcion para crear fichero csv y exportarlo
    descargarCSV(resultadosURL, resultadosScore);
    })
})
```

Figura 51. Exportación de resultados a CSV, fichero index.js

Como se observa en la figura, consiste en ejecutar una acción al clicar el botón correspondiente. Una vez se clica, básicamente se realiza una lectura en base de datos, de donde sacamos los datos que nos interesan (en este caso la URL y el score de las 10 primeras fotos ordenadas precisamente por score, para obtener las que en teoría son las diez mejores) y se van introduciendo una a una en un array. Este array se pasa como argumento a la siguiente función:

```
// Función para crear el fichero CSV y descargarlo
function descargarCSV(arrayURL, arrayScore){
    var rows = arrayURL.length;
    let file_text = `url;score;`;

    for (let i = 0; i < rows; i++) {
        const line = `\n${arrayURL[i]};${arrayScore[i]};`
        file_text += line;
    }
    // descargar csv
    download(file_text);

}
```

Figura 52. Preparación del CSV para descargarlo, fichero index.js

Esta función, a su vez, se encarga de preparar la información y situarla en forma de archivo CSV, dándole el formato correcto, con la cabecera y con las distintas filas, separando cada columna con punto y coma para obtener un archivo CSV (separado por comas). Finalmente se introduce este texto formateado en la última función, que descarga el archivo creado:

```
// Función a la que le pasamos el CSV y lo descarga del navegador
const download = function(data) {

    const blob = new Blob ([data], { type: 'text/csv' });
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');

    a.setAttribute('hidden', '');
    a.setAttribute('href', url);
    a.setAttribute('download', 'resultados_pictures.csv');

    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
}
```

Figura 53. Descarga del fichero previamente creado, fichero index.js

- Cambiar layout entre 2 y 9 imágenes

Para finalizar, tenemos la función que controla el layout de dos o nueve fotografías en pantalla. Resulta simple de implementar, ya que solamente cambia una variable y realiza una llamada a recargar la página ejecutando otra vez la función principal que hemos visto en este mismo apartado:

```
// Cambios a realizar al pulsar el botón de número de fotos (2 -> 9 o viceversa)
const layoutAlt = document.getElementById('more_pictures');
layoutAlt.addEventListener('click', (e) => {
    if (img == 2) {
        img = 9;
    } else {
        img = 2;
    }
    // Obtener datos de la BD e inicializar la página
    db.collection('pictures').onSnapshot(snapshot => {
        setupPictures(snapshot.docs, img);
    }, err => {
        console.log(err.message)
    });
})
```

Figura 54. Cambio de layout entre 2 y 9 imágenes alternativamente, fichero index.js

Dicha función se vuelve a ejecutar, lo que cambia la presentación de la página entre los layouts de 2 y 9 fotos alternativamente.

4.4.4 Elo-rating.js[5]

La implementación de este sistema resulta simple. Como se ha visto en el apartado anterior, se le realiza una llamada directa al objeto 'EloRating' junto con una de sus funciones, la cual calcula las variaciones de *rating* en función de las puntuaciones de la imagen ganadora y de la perdedora. La implementación de dichas funciones es la siguiente:

[5] Extraído de https://www.npmjs.com/package/elo-rating

```
calculate(playerRating, opponentRating, playerWin = true, k = 50) {
    const playerExpected = this.expected(playerRating, opponentRating);
    const ratingChange = parseInt(k * (!!playerWin - playerExpected), 10);

    playerRating = playerRating + ratingChange;

    opponentRating = opponentRating + ratingChange * -1;

    return [playerRating, opponentRating];

}
```

Figura 55. Cálculo de puntuaciones, fichero elo-rating.js

Esta es la función 'calculate', que es la que se llama desde el script 'index.js' y por tanto la principal del sistema elo-rating. Observamos cómo funciona: primero calcula el factor E o '*expected*' del enfrentamiento, para luego aplicar la función 'parseInt' con el factor k a la resta entre la victoria del jugador (valor numérico 1 normalmente) y la probabilidad que se le había calculado al jugador. Esta función realiza un simple cálculo, restando 1 – la probabilidad y multiplicando ese valor por el factor K. El 10 que aparece es la base numérica a la que se está transformando el valor *true* de la victoria del jugador o imagen.

Posteriormente, cambia los valores de *rating* y los devuelve como resultado de la función. Cabe recordar que esta función implementa un sistema de suma cero, y por tanto los puntos que gane una imagen serán iguales a los que pierda la otra.

Para realizar estos cálculos se efectúa una llamada a la función 'expected':

```
expected(playerRating, opponentRating) {
    return 1 / (1 + Math.pow(10, this.ratingDifference(opponentRating, playerRating) / 400));
},
```

Figura 56. Cálculo de probabilidad de victoria en enfrentamiento, fichero elo-rating.js

Esta función consiste en la implementación de la fórmula del Elo Rating en su versión para JavaScript. Para ello se calcula la diferencia de *rating* entre ambas imágenes y se aplica la fórmula de la figura. El factor 400 se explica debido a que es el límite que se utiliza en el cálculo de la diferencia de *ratings*, ya que las probabilidades a partir de dicho número no resultan interesantes para los cálculos a realizar.

```
ratingDifference(playerRating, opponentRating) {
    return Math.max(Math.min(playerRating - opponentRating, 400), -400);
},
```

Figura 57. Cálculo de diferencia de puntuaciones entre competidores, fichero elo-rating.js

4.4.5 Glicko2.js[6]

En este caso, y como se ha explicado a lo largo de los distintos apartados, este sistema resulta bastante más complejo que el caso anterior. Esto es debido a que su funcionamiento no es tan directo y dependiente de una fórmula como el elo-rating, sino que implementa diversos algoritmos y factores al sistema de clasificación utilizado.

Como no se ha implementado manualmente, sino que se trata de un script importado, simplemente se va a proceder a explicar en este apartado el funcionamiento general y de las funciones que se han utilizado directamente.

[6] Extraído de https://www.npmjs.com/package/glicko2

Al contrario del sistema anterior, en este caso la idea del sistema se basa en la creación de objetos (players) desde los cuales se puede realizar llamadas a las distintas funciones que implementa el script. De esta manera, y siguiendo los pasos explicados en el apartado 4.4.3, hemos utilizado las siguientes funciones:

- makePlayer(score):

```
Glicko2.prototype.makePlayer = function (rating, rd , vol) {
    //We do not expose directly createInternalPlayer in order to prevent the assignation of a custom player id whose uniqueness could not be guaranteed
    return this._createInternalPlayer(rating, rd, vol);
};

Glicko2.prototype._createInternalPlayer = function (rating, rd , vol, id){
    if (id === undefined){
        id = this.players_index;
        this.players_index = this.players_index + 1;
    } else {
        //We check if the player has already been created
        var candidate = this.players[id];
        if (candidate !== undefined){
            return candidate;
        }
    }
    var player = new Player(rating || this._default_rating, rd || this._default_rd, vol || this._default_vol,
                            this._tau, this._default_rating, this._volatility_algorithm, id);

    this.players[id] = player;
    return player;
};
```

Figura 58. Creación de objeto Player, fichero glicko2.js

Como su nombre indica, se utiliza para crear el objeto Player. Esta creación del objeto se realiza cada vez que se va a utilizar las funciones, es decir, en cada enfrentamiento, y se asigna a cada jugador su puntuación, y los valores por defecto de 'rating_deviation' (factor que pondera la confianza en la precisión del rating) y de la volatilidad (factor de fluctuación del rating, que suele ser bajo). No ha habido motivo para realizar cambios en estos valores en nuestro proyecto.

- makeRace(players): crea un objeto de tipo Race (carrera/enfrentamiento). En nuestro caso, le pasamos como primera posición a la imagen ganadora y el resto empatada en segunda posición, por lo que entre las perdedoras no hay intercambio, y solamente le aportan todas puntos a la ganadora (dependiendo de su puntuación cada una).

```
Race.prototype.computeMatches = function(results){
    var players = [];
    var position = 0;

    results.forEach(function (rank) {
        position += 1;
        rank.forEach(function (player) {
            players.push({"player": player, "position": position});
        })
    })

    function computeMatches(players){
        if (players.length === 0) return [];

        var player1 = players.shift()
        var player1_results  = players.map(function(player2){
            return [player1.player, player2.player, (player1.position < player2.position) ? 1 : 0.5];
        });

        return player1_results.concat(computeMatches(players));
    }

    return computeMatches(players)
}
```

Figura 59. Creación de objeto Enfrentamiento, fichero glicko2.js

Como se observa en la figura, este tipo de objeto es una concatenación de los distintos objetos Player creados anteriormente.

- updateRatings(Race): esta función realiza, a su vez, una llamada a la función 'update_rank', la cual se encarga de ejecutar el enfrentamiento y calcular los puntos que cada player ha de sumar o perder.

```
Player.prototype.update_rank = function(){
    if (!this.hasPlayed()){
        this._preRatingRD();
        return;
    }

    var v = this._variance();

    var delta = this._delta(v);

    this.__vol = this.volatility_algorithm(v, delta);

    this._preRatingRD();

    this.__rd = 1 / Math.sqrt((1 / Math.pow(this.__rd, 2)) + (1 / v));

    var tempSum = 0;
    for (var i=0,len = this.adv_ranks.length;i< len;i++){
        tempSum += this._g(this.adv_rds[i]) * (this.outcomes[i] - this._E(this.adv_ranks[i], this.adv_rds[i]));
    }
    this.__rating += Math.pow(this.__rd, 2) * tempSum;
};
```

Figura 60. Actualización de scores, fichero glicko2.js

Esta función toma los jugadores implicados y sus variables asociadas y realiza los cálculos implicados en el sistema Glicko. La peculiaridad de este script es que permite realizar estos cálculos en enfrentamientos múltiples, en nuestro caso con 9 jugadores implicados.

La fórmula utilizada es la siguiente:

$$\phi' = 1/\sqrt{\frac{1}{\phi^{*2}} + \frac{1}{v}}$$

$$\mu' = \mu + \phi'^2 \sum_{j=1}^{m} g(\phi_j)\{s_j - \mathrm{E}(\mu, \mu_j, \phi_j)\}$$

Figura 61. Fórmulas que implementa el sistema Glicko[7]

Se puede observar la complejidad de la misma por lo que no va a ser explicada en esta memoria. En nuestro proyecto ha sido utilizada como herramienta y no ha sido modificada ni implementada de otra manera, por lo que no resulta de interés profundizar en la complejidad matemática de la misma. De igual modo, es interesante saber la magnitud de los cálculos que se realizan en este script.

- getRating(Player): esta última función simplemente accede al objeto Player creado anteriormente y obtiene su puntuación. Es importante de nombrar ya que en todos los casos en los que se crea un Player o se accede a su Rating, se realiza un cambio de formato, ya que la fórmula de la figura anterior necesita que este valor esté dentro de un rango y los ratings normales (en nuestro caso alrededor de 1500-2000 puntos) no permitirían realizar esos cálculos.

4.5 Resultados obtenidos: Aplicación Web

Siguiendo todos los pasos recientemente explicados, se ha obtenido el siguiente resultado. La web se puede visitar en el siguiente enlace: https://tfm-test-1-220321.web.app

[7] Extraído del paper 'A Comprehensive Guide To Chess Ratings' del profesor Mark E. Glickman, creador del sistema Glicko.

4.5.1 Inicio de la aplicación

El aspecto inicial de la aplicación en la primera ocasión en la que se accede es el siguiente:

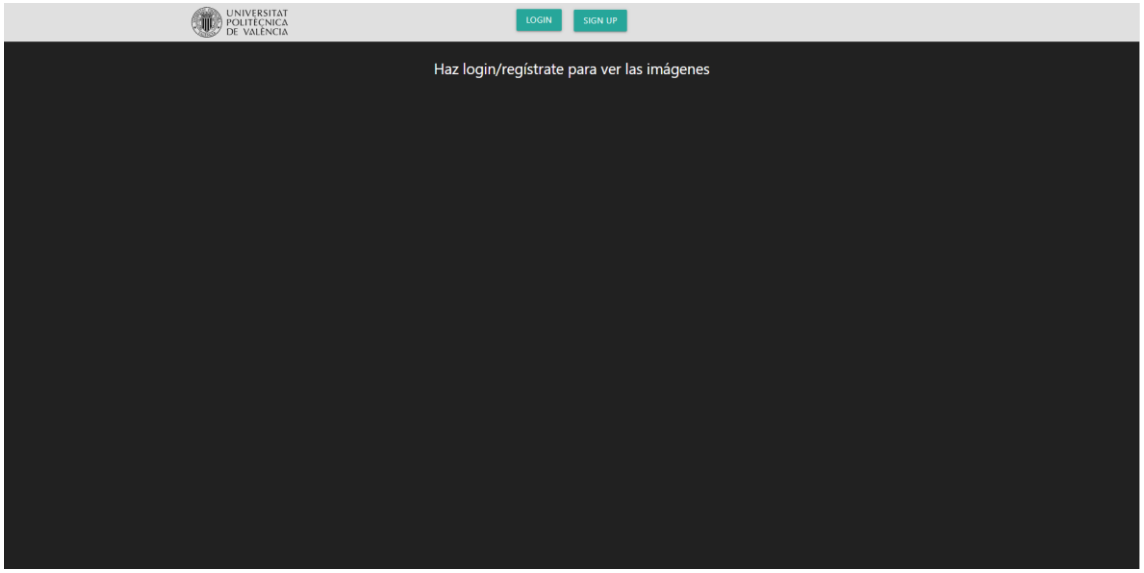

Figura 62. Inicio de la aplicación sin sesión iniciada

Podemos observar que no hay ningún tipo de información disponible, solamente el mensaje de aviso al usuario de que necesita iniciar sesión o registrarse para poder continuar.

Para realizar una de estas dos operaciones, se presentan los botones correspondientes, que dan lugar a la aparición de formularios donde introducir la información y poder comparar con los datos almacenados en Firebase y que controlan la autenticación, como se ha explicado en el apartado 4.2 y 4.4.

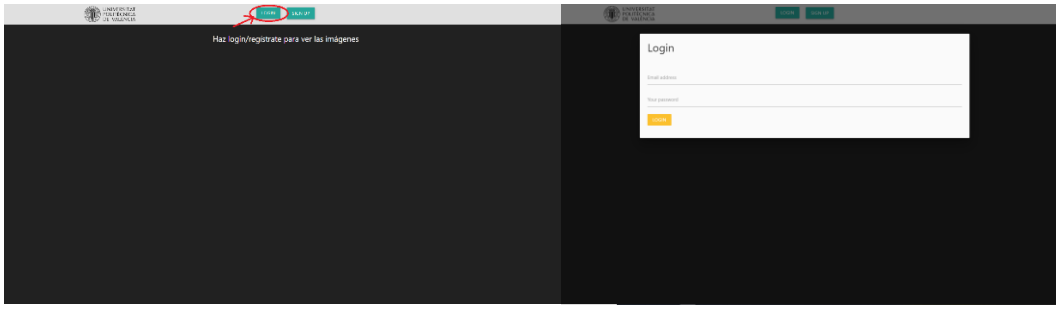

Figura 63. Acceso al formulario de inicio de sesión

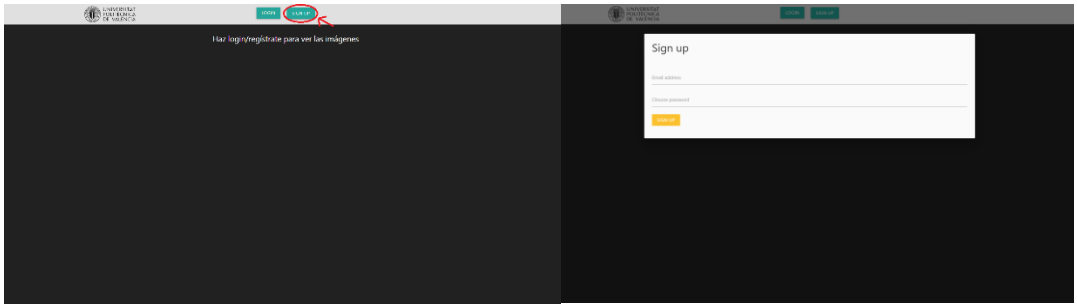

Figura 64. Acceso al formulario de registro de usuario

4.5.2 Aplicación con sesión iniciada

Al iniciar sesión se muestra la página con toda la información disponible y en su modo por defecto de 2 imágenes.

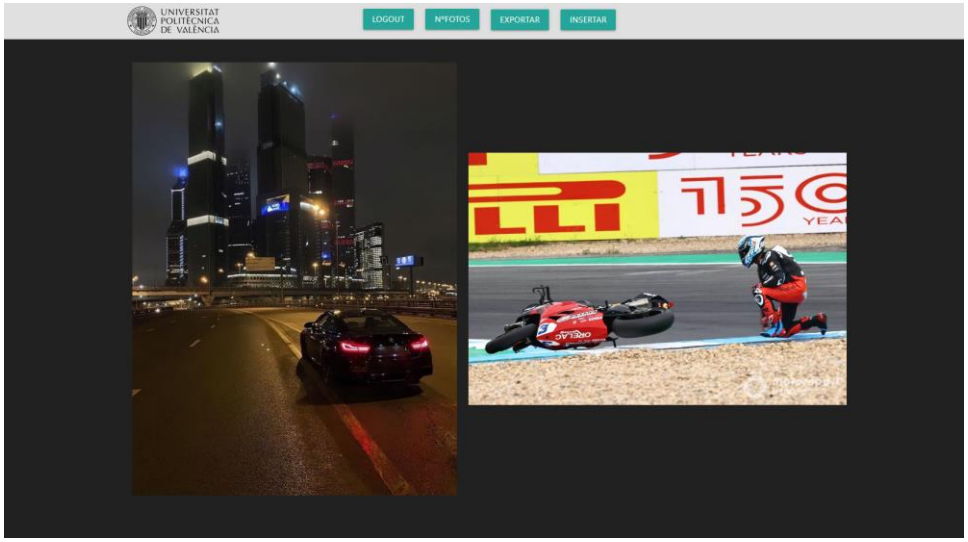

Figura 65. Aplicación con sesión iniciada y derechos de administrador

Aparecen las dos imágenes en el centro, una a cada lado, y son ambas clickables. En el caso de clicar una u otra, se lleva a cabo el proceso de registrar los cambios y se refresca la página, mostrando otra vez dos imágenes.

Ahora aparecen más botones, con las funcionalidades de cerrar sesión (volver a la página anterior), cambiar el modo entre 2 y 9 fotos, exportar resultados (descargar el archivo, no cambia el aspecto de la página) e insertar una nueva imagen en base de datos (aparece un formulario).

El modo de 9 fotos se ha ideado solamente para su representación en sistemas de escritorio, ya que en un teléfono móvil, al tener la pantalla más estrecha, no se tiene una buena visibilidad. Es por eso que, como vemos en las siguientes capturas, el botón no aparece en dichos casos.

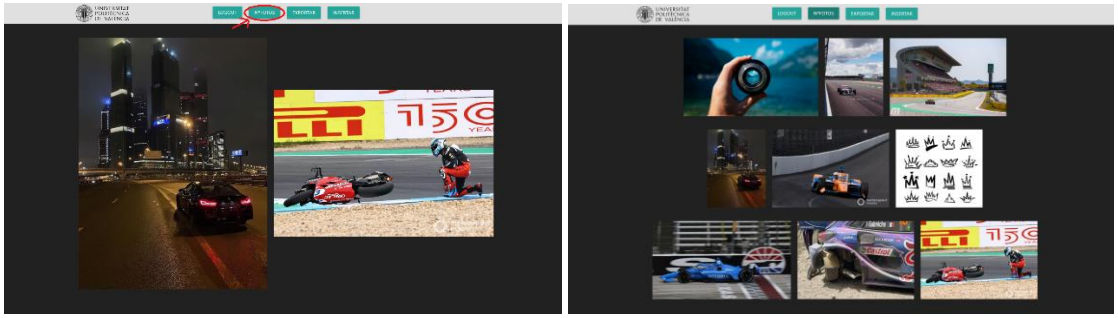

Figura 66. Layouts de dos (izquierda) y nueve imágenes (derecha)

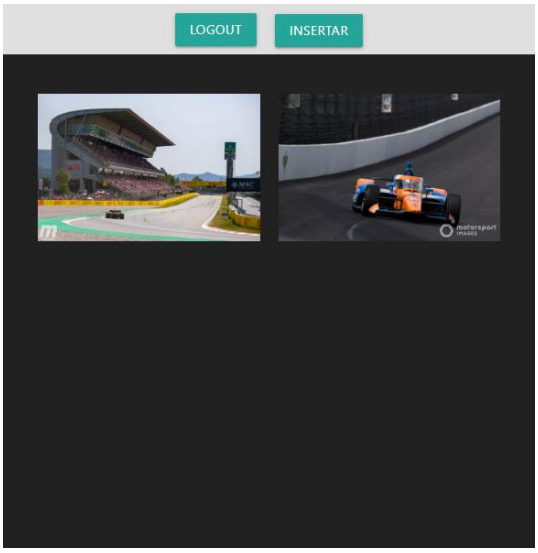

Figura 67. Presentación de la app en navegador móvil (modo vertical)

En cuanto a la opción de insertar una imagen en base de datos, aparece el siguiente formulario en el que se solicita la dirección URL de la misma y clicar en el botón de añadir para realizar el proceso.

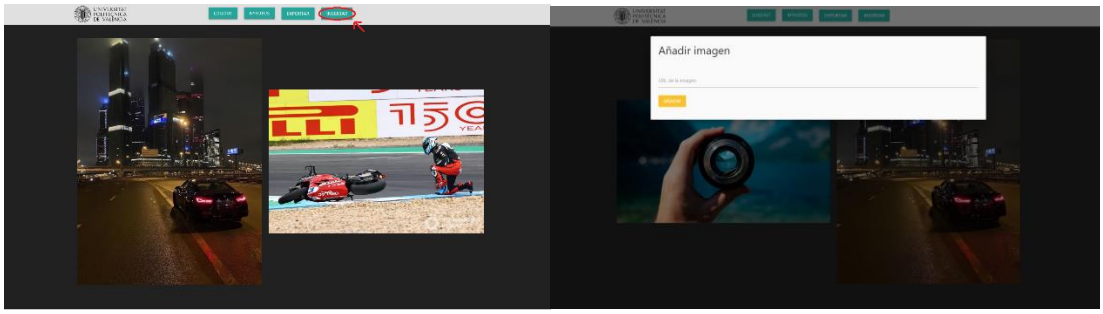

Figura 68. Acceso al formulario de inserción de imágenes en BD

4.5.3 Formato de los archivos exportados

Como el objetivo de los datos obtenidos es el de realizar análisis con ellos, se ha creído oportuno tener la posibilidad de extraerlos directamente desde la aplicación, sin necesidad de realizar operaciones en Firebase con la base de datos. Por tanto, el único requisito era que este archivo exportado sea fácilmente operable, y se ha decidido que el mejor modo de obtener esto era mediante un archivo CSV, o de valores separados por comas.

Este tipo de archivos pueden abrirse fácilmente con Excel o Matlab, de manera que se puede leer el archivo y traspasar sus valores directamente a un *array* con el cual realizar operaciones matemáticas, ordenarlos, modificarlos para adaptarlos a otra escala…

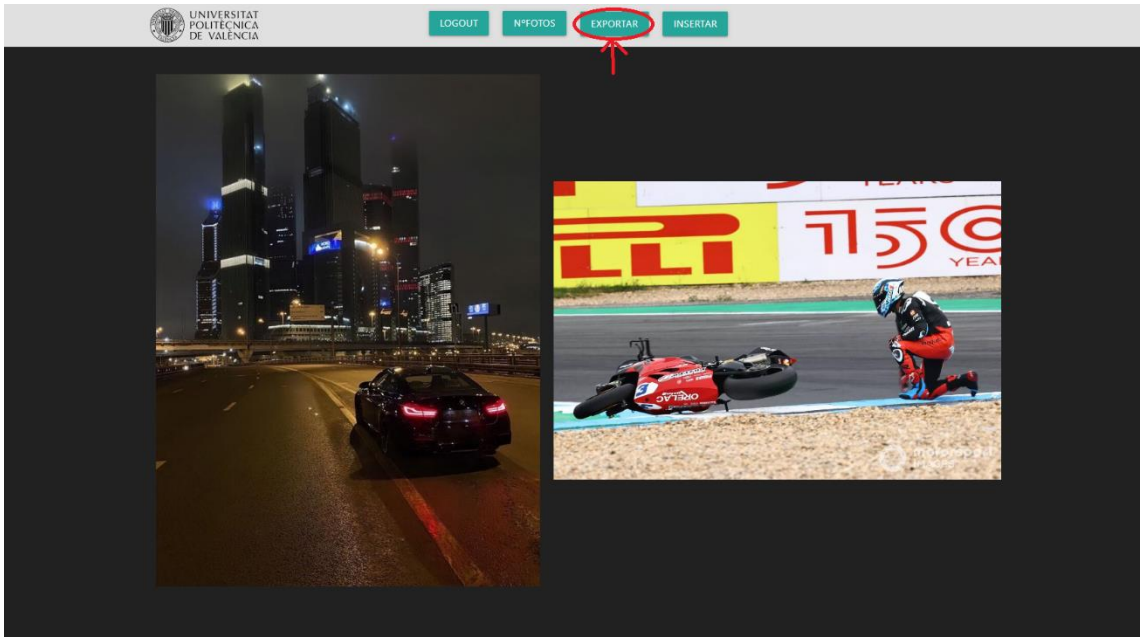

Figura 69. Opción de exportar resultados de BD

	A	B	C
1	url	score	
2	https://cdn-1.motorsport.	1596	
3	https://i.redd.it/d3d8vtcr	1582	
4	https://i.pinimg.com/564:	1559	
5	https://pbs.twimg.com/m	1534	
6	https://cdn-1.motorsport.	1524	
7	https://cdn-1.motorsport.	1500	
8	https://i.pinimg.com/564:	1500	
9	https://cdn-1.motorsport.	1500	
10	https://cdn-1.motorsport.	1500	
11	https://d500.epimg.net/c	1500	
12			
13			

Figura 70. Ejemplo de fichero descargado con datos de BD

Capítulo 5. Conclusiones y propuesta de trabajo futuro

5.1 Conclusiones del proyecto

Para hablar de las conclusiones, se va a revisar los objetivos indicados en el apartado introductorio de esta memoria.

Como objetivo principal, se propuso la creación de una aplicación web que sirviera como herramienta para conocer datos acerca de los usuarios sin la necesidad de preguntarles directamente. Para ello, se pensó en una página en la que se presenten dos fotos y preguntar cuál es la mejor a ojos del usuario, y con dichos resultados poder realizar un análisis acerca de qué preferencias tiene. Este objetivo ha sido cumplido, ya que la página ha sido creada y con ella la base de datos que nos indica qué imágenes son las preferidas de los usuarios, ordenando estas imágenes con lo que de un vistazo se puede realizar un análisis rápido.

Esta aplicación se puede convertir en una base para realizar otro tipo de análisis más concienzudos, pero realmente del mismo tipo, sin grandes necesidades, más allá del uso de herramientas disponibles online.

En cuanto a los objetivos secundarios planteados en el apartado 1.2, también han sido cumplidos los siguientes:

- Se llevó a cabo la investigación de cómo obtener estos datos, llegando a la conclusión de que la mejor manera era enfrentando imágenes de manera que el usuario tuviera que introducir un *input*, en este caso elegir cuál es su favorita. Para esta investigación se consideraron diferentes alternativas, ya que hay muchas posibilidades en cuanto a pedirle participación a un usuario, pero siempre se consideró que nuestra primera opción era el enfrentamiento directo entre fotografías, ya que consiste en la polarización entre diferentes opciones, con lo que el usuario está diciendo si prefiere una u otra, en su caso imágenes, pero con un transfondo de características que esconden dichas imágenes, lo cual es lo realmente importante con una aplicación de este tipo como base.
- Se ha implementado un algoritmo para dirigir la selección de imágenes al usuario de forma que los enfrentamientos presentados sea interesante y tenga detrás algo más que el simple azar, de manera que los resultados obtenidos tienen realmente una función y un efecto sobre la base de datos, tratando siempre de enfrentar imágenes con puntuaciones similares a la vez que insertando en ciertas ocasiones enfrentamientos "trampa" para evaluar el comportamiento del usuario y dirigiendo el ciclo de vida del uso de la aplicación hacia donde es más correcto y funcional.
- Se llevó a cabo la investigación de cómo clasificar las imágenes, llegando a la conclusión de que el mejor sistema disponible y que se podía utilizar para esta aplicación era un sistema ELO implementado en JavaScript (explicado en el apartado 3.1).
- Como último objetivo secundario, el entendimiento de todas las herramientas utilizadas, de manera que la aplicación pueda ser escalable, utilizable posteriormente como base, entendiendo las investigaciones e implementándolas de forma que si alguien tiene la intención de utilizarlas esté correctamente explicado, además de integrando todo el sistema de forma transparente al usuario y simple para el desarrollador.

No obstante, algunos inconvenientes han surgido en el desarrollo del proyecto, siendo el mayor de ellos la integración de algunos sistemas externos o librerías en el código de la página. Por ejemplo, el uso del sistema de clasificación Glicko 2 no fue tan simple como el de elo-rating debido a la forma en la que uno y otro están programadas en JavaScript, ya que han tenido que ser traducidas desde Node.js y fue un proceso algo complejo.

Como conclusión final, he de decir que estoy satisfecho con el proceso seguido, ya que era un proyecto que en sus inicios parecía algo complejo y abstracto, pero que con la ayuda de los tutores

y dedicando tiempo ha sido posible de llevar a cabo y que en mi opinión cumple también con lo que se pretendía desde la oferta por su parte.

5.2 Propuesta de trabajo futuro

Como propuesta de trabajo futuro, se identifican dos posibles vías que se pueden seguir. Por un lado, continuar con la aplicación y realizarle mejoras, y por el otro, utilizar esta aplicación como base y, utilizando los resultados exportados, realizar análisis. Estas dos vías se pueden llevar de forma paralela, pero considero que sin mejorar la aplicación no terminará de ser posible realizar un análisis realmente exhaustivo e interesante.

En cuanto a mejorar la aplicación, lo primero y principal en lo que pienso es en una mejora de la base de datos de las imágenes, desde el uso de la misma base de datos para almacenarlas (lo cual sería útil al poder volcar directamente las fotografías, pero muy complejo, ya que por ejemplo la base de datos de Firebase tiene unos requisitos estrictos de almacenamiento) y con ello poder incluir no solo el dato de la URL, sino que se incluyan los metadatos de la imagen, como la resolución, distancia focal, apertura de la lente… y con ello realizar un análisis realmente importante de los gustos del usuario.

Otra mejora en este campo podría ser un mejor diseño de la presentación de la página. Como se ha explicado en la memoria, se eligió un sistema de Material Design de Google debido a su facilidad de implementación y simplicidad de presentación, pero se le podría dar un giro al diseño y crear una página diferenciada y moderna de acuerdo a la aplicación que se está implementando.

En la parte de análisis de los datos, y como se acaba de explicar, los metadatos de las imágenes otorgan una cantidad de posibilidades enorme a la hora de analizar. Con toda esta información que se puede obtener de una simple imagen y de las preferencias de los usuarios, se podría implementar un programa matemático que realizara cálculos teniendo en cuenta la puntuación de una imagen con respecto a las demás, así como sus características propias de imagen, y tomar decisiones en base a ello: caracterizar un grupo, sus preferencias, su forma de comunicarse…

En definitiva, la mejora de la base de datos sería mi principal sugerencia a la hora de continuar con este proyecto, ya que cuando se dispone de más información se dispone de una mayor cantidad de herramientas sobre las que realizar análisis, y con ello mayor versatilidad y posibilidades de tomar diferentes caminos con ello.

Capítulo 6. Bibliografía

- Luján Mora, Sergio (2002). Programación de Aplicaciones Web. Editorial Club Universitario, Alicante.
- W3Schools [*online*]. https://www.w3schools.com [Último acceso: 25 de Agosto de 2022]
- ReactJS [*online*]. https://es.reactjs.org [Último acceso: 25 de Agosto de 2022]
- Angular [*online*]. https://angular.io [Último acceso: 25 de Agosto de 2022]
- Bootstrap [*online*]. https://getbootstrap.com [Último acceso: 25 de Agosto de 2022]
- Materialize CSS [*online*]. https://materializecss.com [Último acceso: 25 de Agosto de 2022]
- Materialize CSS [*online*]. https://desarrolloweb.com/articulos/materialize-framework-css.html [Último acceso: 25 de Agosto de 2022]
- Firebase [*online*]. https://firebase.google.com/?hl=es [Último acceso: 25 de Agosto de 2022]
- AWS Lambda [*online*]. https://aws.amazon.com/es/lambda/ [Último acceso: 25 de Agosto de 2022]
- AWS Amplify [*online*]. https://aws.amazon.com/es/amplify/ [Último acceso: 25 de Agosto de 2022]
- Élő, Árpád Emrick (1978). The Rating of Chessplayers, Past and Present.
- Hendrikus Albers, De Vries (2001). Elo-rating as a tool in the sequential estimation of dominance strength. DOI: 10.1006/anbe.2000.1571
- Prof. Mark E. Glickman. A Comprehensive Guide To Chess Ratings. Department of Mathematics, Boston University.
- Ebtekar, Liu (2021). Elo-MMR: A Rating System for Massive Multiplayer Competitions. DOI: 10.1145/3442381.3450091
- Trueskill Ranking System [*online*]. https://www.microsoft.com/en-us/research/project/trueskill-ranking-system/ [Último acceso: 25 de Agosto de 2022]
- El sector de los esports en cifras [*online*]. https://www.visibilitas.com/sector-esports-cifras/ [Último acceso: 25 de Agosto de 2022]
- Bonetto, María Julia (2016). El uso de la Fotografía en la investigación social. Revista Latinoamericana de Metodología de la Investigación Social. ISSN 1853-6190. Pp. 71-83.
- Documentación de Firebase [*online*]. https://firebase.google.com/docs [Último acceso: 25 de Agosto de 2022]
- Documentación de JavaScript [*online*]. https://developer.mozilla.org/es/docs/Web/JavaScript [Último acceso: 25 de Agosto de 2022]
- Repositorio de GitHub donde se encuentra almacenado todo el código que implementa la aplicación: https://github.com/diepela/TFM_2022