CoacPoint

Isaac Hernández Baizán
Grado de Multimedia
Ingeniería web

Consultor: Ignasi Lorente Puchades

Profesor responsable de la asignatura: Carlos Casado Martínez

A mi familia ...

Resumen

CoacPoint es una aplicación web destinada a los aficionados al concurso oficial de agrupaciones carnavalescas (COAC) del Carnaval de Cádiz. Uno de los principales objetivos de esta aplicación, es fomentar el seguimiento del concurso mediante funcionalidades que van desde la puntuación, por parte de cada usuario, a las agrupaciones con la respectiva difusión a través de las principales redes sociales hasta la recopilación de noticias relacionadas con el concurso y aspectos generales de la fiesta.
El desarrollo tecnológico de la aplicación se ha llevado a cabo haciendo uso principal de un framework de Javascript llamado Angular, en su versión 5, complementada con recursos extraídos de las APIs oficiales del mismo framework como Angularfire2 o Angular CLI y otras pertenecientes a terceros como Firebase, NgShare Buttons y Bootstrap 4, siendo esta última, artífice de proporcionar el grado de usabilidad correcto aplicando características propias del diseño responsivo.

Abstract

Coacpoint is a web application destined to the enthusiastic about official contest of carnival groups of the carnival of Cadiz. One of the main objectives of this application is to promote the following of the contest with functionalities that go from to give a score for carnival groups, by each user, with a broadcasting through main social networks, to the news summary related with the contest and other general aspects of the town's festivity.
The technologic development of the application is based in a Javascript framework called Angular (version 5), complemented with official APIs like Angularfire2 o Angular CLI, and other external APIs like Firebase NgShare Buttons and Bootstrap 4 that provide a correct usability grade applying typical features of the responsive design.

Palabras clave

Carnaval de Cádiz, COAC, concurso de agrupaciones, puntuación, jurado, carnaval, Angular, framework, firebase, ingeniería web, coacpoint.

Índice

Lista de figuras

Lista de figuras

1. Introducción

1.1 Contexto y justificación

El Carnaval de Cádiz continúa siendo la fiesta por excelencia de la ciudad. Ésta alberga un Concurso Oficial de Agrupaciones Carnavalescas (COAC) cuya repercusión mediática es muy elevada, no sólo a nivel local y provincial, sino que se traslada también a nivel regional en un menor grado, e incluso con alguna repercusión diseminada pero latente por el ámbito nacional.
El COAC está compuesto, además de las diferentes agrupaciones participantes, de un jurado formado por un presidente y un número de vocales que dictaminan la calidad de cada una de las agrupaciones mediante una puntuación numérica. Estas valoraciones, es el principal motivo del gran flujo de opiniones y comentarios que se generan en referencia al concurso, llegando a adquirir tal relevancia en los aficionados, que la temática podría mantenerse como temática central del mayor número de conversaciones interpersonales durante un largo periodo de tiempo.
El concepto de jurado (no oficial) fue un proyecto pionero del periódico principal de la ciudad de Cádiz (Diario de Cádiz), qué al tener un éxito tan elevado, fue utilizado como principal recurso para aumentar el número de ventas de ejemplares y desbancar a otros periódicos de la zona. Este arma mediatica, aún continúa en uso por parte de algunos periódicos, y es divulgada tanto de manera tradicional como en la versión digital del mismo, aunque no con la misma calidad ni cantidad de detalles en la información ofrecida.
El principal motivo de la creación de la aplicación es trasladar el formato tradicional que se ha llevado hasta ahora en la práctica de la crítica social, en referencia al concurso de carnaval y sus agrupaciones, a un formato digital basado en una aplicación web cuyo centro holístico es el usuario y, por tanto, debe ofrecer una usabilidad notoria que produzca el manejo simple e intuitivo para que pueda ser utilizado por el mayor número de usuarios posibles.

1.2. Definición

CoacPoint es una aplicación web cuya principal funcionalidad es permitir que un determinado usuario pueda realizar una valoración, atendiendo a su opinión, para cada una de las agrupaciones que participen en el COAC haciendo uso del baremo de puntuación que utilizan los miembros del jurado oficial del concurso. De esta manera, y aprovechando el concepto de web 2.0, o web social virtual, tan presente en la actualidad, las personas podrán compartir sus preferencias con otras y que estas puedan ser discutidas, creando así un espacio social idóneo para los aficionados del Carnaval, de forma inicial en el proyecto, aprovechando el impulso que otorgan las principales redes sociales actuales.

De manera sistemática, la aplicación web cuenta como componente principal al sistema de puntuaciones vinculado a cada uno de los usuarios, reforzado con un componente social intrínseco reflejado en típicas funciones de compartición de información que ofrecen los sistemas implementados en las redes sociales, junto a la adición del factor de la competición entre los usuarios que puede fomentar la diseminación del uso de la aplicación entre los aficionados. El mecanismo de fusión entre una red social y la aplicación conlleva la gestión y el control del registro de usuarios con cuentas propias de la red social, con el consiguiente potencial enriquecimiento relacional entre ellos al establecer un sistema de navegación usable y fluido.

Cabe destacar, que la aplicación alberga una sección de noticias cuya primer prototipo será mostrar noticias sobre eventos y otras curiosidades de la fiesta, introducidas directamente por el desarrollador o administrador. En versiones posteriores, albergará sistemas de comentarios y un sistema de gestión de contenidos para el administrador/a del sitio.

1.3. Objetivos

1.3.1. Objetivos primarios

- Impulsar un espacio virtual de carácter social entre los aficionados del Concurso Oficial de Agrupaciones Carnavalescas (COAC) y otros aspectos generales de la fiesta del Carnaval de Cádiz.

- Fomentar el uso de la aplicación entre la comunidad de aficionados.

- Realizar la curva aprendizaje del framework Angular y todas las APIs involucradas en el proyecto.

1.3.2. Objetivos secundarios

- Crear un diseño intuitivo y responsivo de la información e interfaces de la aplicación para el usuario final.

- Unificar las temáticas del COAC y de la fiesta del Carnaval.

- Asegurar la fiabilidad y calidad del funcionamiento.

- Establecer estrategias básicas de promoción y difusión.

1.4. Enfoque y metodología empleada

El enfoque o alcance general que queremos otorgar al proyecto nos lleva a desarrollar un prototipo funcional de aplicación web multiplataforma desde cero, con

un diseño adaptado mediante técnicas responsivas. No obstante, el prototipo alberga elementos y funcionalidades comunes de una aplicación web de carácter social junto a la adaptación de elementos más característicos del sistema.

Una vez definidos los objetivos, determinados los requisitos y el alcance general del proyecto, la metodología de trabajo que llevaremos a cabo consta de la estructuración del diseño de la aplicación se hará en base a un diseño centrado al usuario (DCU), mientras que la ejecución de los procesos internos del proyecto se basa en los principios del modelo en cascada (*waterfall*). El proyecto cuenta con dos grandes líneas de trabajo cuya implementación se realiza de forma paralela durante toda la vida del proyecto: la línea de investigación y la línea de desarrollo.

La línea de investigación alberga los procesos de recopilación y estudio de información esenciales para llevar a cabo todas las tareas planificadas con una ocupación total del tiempo invertido en el proyecto. Por otro lado, la línea de desarrollo está compuesta por las fases pertenecientes al modelo en cascada: estudio de requisitos, fase de diseño, la fase de desarrollo o implementación (creación del prototipo) y la fase de pruebas unitarias, comprobación y mantenimiento.
Con la creación del prototipo funcional, se procederá al cierre del proyecto con la finalización de la memoria y la preparación de la exposición del proyecto.

1.5. Planificación

Las tareas de la planificación que mostramos a continuación reflejan, únicamente, la fase de desarrollo anteriormente mencionada. Debido a que la fase de investigación (estudio y recopilación de información variada) se realiza de manera necesaria y constante para llevar a cabo todas y cada una de las tareas planificadas de la fase de desarrollo, no aparece representada en el diagrama de Gantt.

La elaboración del plan establecido se ha llevado a cabo haciendo mención a apartados más generales, de esta manera, le otorga un mayor grado de libertad al proyecto a la hora de establecer prioridades y decisiones que sufren modificaciones a lo largo del desarrollo general del mismo. No obstante, se han realizado tareas a partir de las partes principales que componen el proyecto.

1.5.1. Diagrama de Gantt

	Nombre de la tarea	Fecha de Inicio	Fecha final	Duración
1	**Definición y planificación**	**20/09/17**	**03/10/17**	**14d**
2	Análisis y objetivos	20/09/17	22/09/17	3d
3	Enfoque y metodología	23/09/17	26/09/17	4d
4	Planificación	27/09/17	03/10/17	7d
5	Entrega PAC 1 (Hito)	03/10/17	03/10/17	
6	**Diseño**	**04/10/17**	**01/11/17**	**29d**
7	Diseño centrado en el usuario (DCU)	04/10/17	18/10/17	15d
8	Diagramas UML	19/10/17	25/10/17	7d
9	Arquitectura de software	26/10/17	01/11/17	7d
10	Entrega PAC 2 (Hito)	01/11/17	01/11/17	
11	**Desarrollo**	**02/11/17**	**10/12/17**	**39d**
12	Implementación de código	02/11/17	01/12/17	30d
13	Seguridad, pruebas y verificación	02/12/17	06/12/17	5d
14	Demostración del prototipo	07/12/17	10/12/17	4d
15	Entrega PAC 3 (Hito)	10/12/17	10/12/17	
16	**Entrega Final**	**11/12/17**	**08/01/18**	**29d**
17	Finalización de la memoria	11/12/17	16/12/17	6d
18	Presentación del proyecto	17/12/17	29/12/17	13d
19	Defensa del proyecto	30/12/17	08/01/18	10d
20	Entrega del TFG (Hito)	08/01/18	08/01/18	

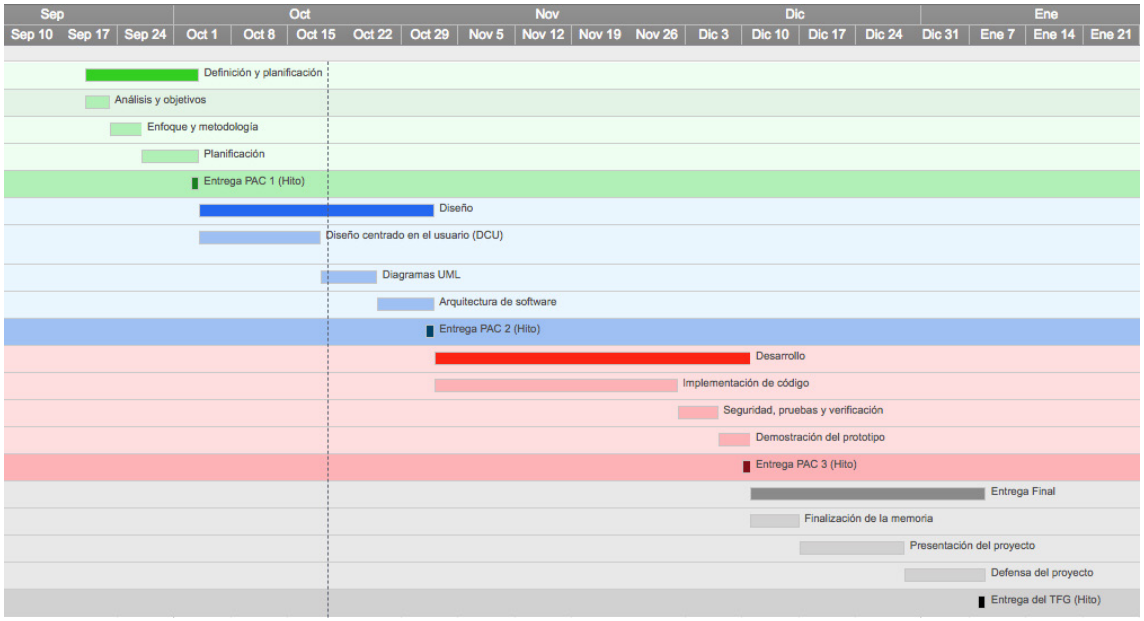

Figura 2. Representación de Diagrama de Gantt.

1.5.2. Descripción de tareas

<u>Definición y planificación</u> (20/09/17 – 03/10/17)

- *Análisis y objetivos (20/09/17 – 03/10/17)*
  - Propuestas y selección de ideas.
  - Establecimiento de la definición de la aplicación.
  - Marco teórico y contextual.
  - Análisis del mercado específico actual.

- *Enfoque y metodología (23/09/17 – 26/09/17)*
  - Determinación de la metodología de trabajo a utilizar.
  - Alcance a grandes rasgos de la aplicación.

- *Planificación (27/09/17 – 03/10/17)*
  - Descripción de todas las tareas realizadas.
  - Establecimiento de hitos y entregas en diagrama de Gantt.

Diseño (04/10/17 – 01/11/17)

- *Diseño centrado en el usuario -DCU- (04/10/17 – 18/10/17)*
  - Análisis de usuarios potenciales, objetivos y contenidos.
  - Diseño conceptual y visual.
  - Arquitectura de la Información: Wireframes (Lo-Fi) y árbol de contenidos.
  - Prototipo (Hi-Fi).
  - Evaluación de usabilidad y experiencia del usuario.

- *Ingeniería de software (19/10/17 – 25/10/17)*
  - Diagramas UML: casos de usos, diagrama de clases y de actividades.
  - Diseño de Base de Datos: diagrama ER, árboles de datos JSON.

- Arquitectura de software (26/10/17 – 01/11/17)
  - Definición de la arquitectura general de Angular.
  - Estructura y fundamentos del API REST de *Firebase*.

Desarrollo (02/11/17 – 10/12/17)

- Implementación de código (pruebas, configuraciones, etc.)
- Muestra preliminar del prototipo mediante vídeo.

1.6. Estructura del resto de los capítulos

Capítulo 2: Análisis. En este apartado se realiza un estudio del mercado actual basado en la especificidad de las funcionalidades de la aplicación, haciendo énfasis en la recopilación de datos sobre los estilos y funciones que desempeñan las aplicaciones potencialmente competentes, además de un primer acercamiento a la definición de target o público objetivo.

Capítulo 3: Diseño. En este capítulo nos adentramos en un estudio más exhaustivo de la interfaz mediante el diseño centrado en el usuario (DCU) que abarca desde la etapa de análisis y modelado del usuario hasta la presentación de prototipos y una evaluación preliminar de usabilidad. Además, se definen las arquitecturas de software utilizadas y los fundamentos que las rigen, como son el framework Angular y el API REST de Firebase.

Capítulo 4: Desarrollo. Implementación de la estructura y el código del prototipo inicial. Como primer hito de la fase de desarrollo, demostraremos, a nivel visual (vídeo), las estructuras creadas y la explicación del funcionamiento principal de la aplicación.

Capítulo 5: Conclusiones. Exposición de las conclusiones finales y las posibles líneas de futuro del proyecto.

2. Análisis

2.1. Estado del arte

La búsqueda exhaustiva sobre la temática de la aplicación, como núcleo de la investigación, nos lleva al estudio del mercado actual haciendo énfasis en diferentes aspectos que no sólo están basados en la especificidad funcional de la aplicación. Debido a la existencia pobre de aplicaciones de esta índole, tanto web como de entornos nativos (*Android* o *iOS*), la dirección de la investigación ha tomado el cauce, en primer lugar, en base a la detección de funcionalidades similares a la nuestra y luego en algunas referentes a la temática general del Carnaval de Cádiz.

Carnavapp (2015)
Plataforma: Android (Google Play)
Desarrollador: GreenerSoft
URL: https://play.google.com/store/apps/details?id=com.greenersoft.carnaval

Es la única aplicación, tanto en plataforma web como para dispositivos móviles, que ha tratado el tema principal de nuestra aplicación: puntuación de las agrupaciones carnavalescas por parte de los usuarios. Aunque no hace referencia al sistema de puntuación oficial que realiza el jurado, utiliza una escala de puntuación parecida donde el usuario puede otorgar un determinado valor a cada agrupación. A partir del resultado total de la puntuación que realiza cada usuario, esta pasa a formar parte de una media aritmética global de la nota de cada uno de los usuarios, estableciendo una clasificación descendente de las agrupaciones que han obtenido una mayor media de las puntuaciones.
Cabe destacar que la aplicación no se encuentra operativa, aunque puede ser descargada, y el sitio web del desarrollador tampoco está disponible.

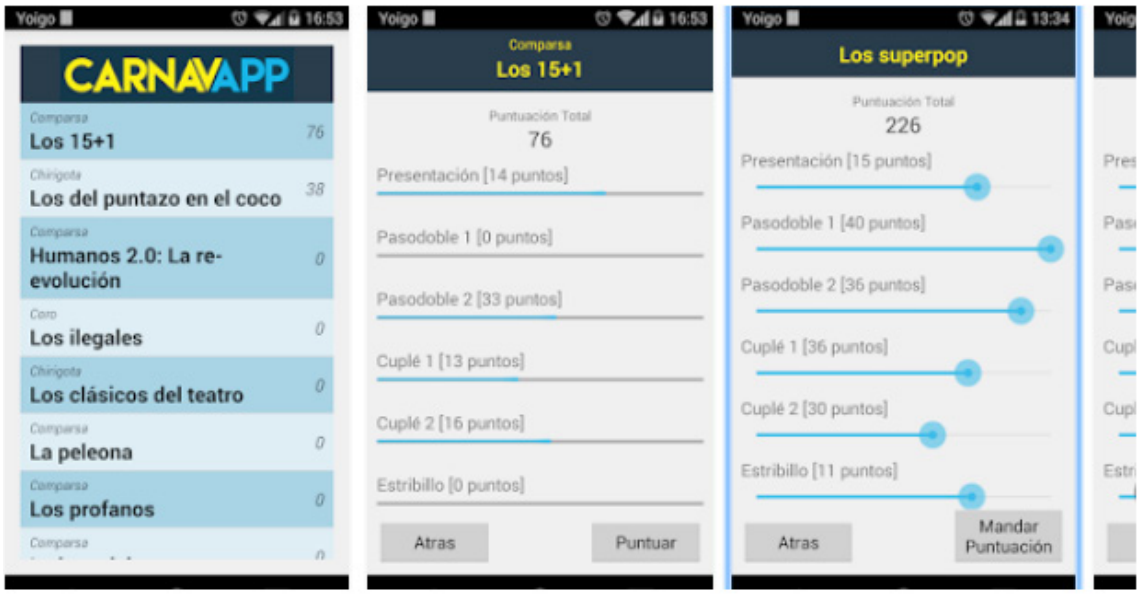

Figura 3. Pantallas de la aplicación Carnavapp (2015)

Carnavapp (2013)

Plataforma: Android (Google Play)

Desarrollador: Jesús Corralejo Banda

URL: https://play.google.com/store/apps/details?id=es.jcorralejo.android.carnavapp

La principal funcionalidad que ofrece la aplicación es brindar al usuario información sobre el COAC, desde datos de agrupaciones, fases del concurso, clasificación, etc.) a la retransmisión en video del concurso (gestión de grabaciones y visualización online). Además, ofrecen los comentarios que han realizado los diferentes medios digitales de comunicación. La aplicación de componente social alguno, junto con el sistema de puntuaciones para los usuarios, característica principal de nuestra aplicación.

Como ocurre con la aplicación anterior, no se encuentra operativa, aunque si descargable. El sitio web del desarrollador tampoco se encuentra disponible.

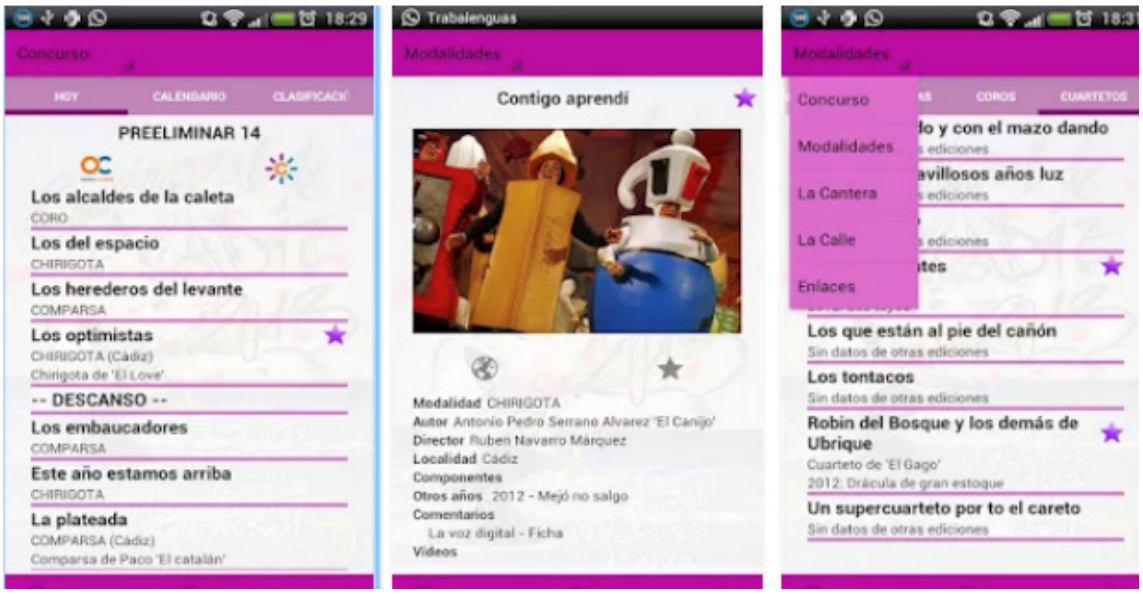

Figura 4. Pantalla de la aplicación CarnavApp (2013)

El Jurado de la Voz (2017)

Plataforma: Web

Desarrollador: Vocento

URL: http://carnaval.lavozdigital.es/eljuradodelavoz/

El Jurado de la Voz es un apartado dentro de la sección de carnaval del periódico gaditano La Voz de Cádiz. Este caso muestra la tradición, que aún sigue en uso, de emitir una puntuación, en formato digital, por parte de algunos miembros del periódico. No existe ninguna interacción por parte del usuario, ni tan siquiera un apartado de comentarios que suele ser usual en las noticias de los periódicos digitales.

P	Chirigotas	Pres.	Pas 1.	Pas 2.	Cup1.	Cup2.	Est1.	Est2.	Pop.	Tipo	Prelim.	Cuartos	Semif.	TOTAL
1	Los del planeta rojo...	15	27	32	45	47	10	10	46	15	241	240	247	728
2	Los de Cádiz norte	14	32	33	40	42	9	9	47	14	228	248	240	716
3	No te vayas todavía	16	34	36	45	52	11	11	48	15	206	235	268	709
4	Mi suegra como...	13	28	32	38	42	9	9	44	15	236	239	230	705
5	No valemo un duro	13	26	30	36	40	10	10	44	14	229	245	223	697
6	Pa religión la mía	13	31	33	39	43	8	8	45	12	222	233	232	687
7	Los Indiana Jones...	12	29	30	40	46	10	10	45	13	226	223	235	684
8	Qué penita de...	12	29	27	36	35	8	8	43	13	212	228	211	651

Figura 5. El jurado de la Voz (La Voz de Cádiz)

Atendiendo a estos resultados de análisis de mercado que existe actualmente en torno a este tipo de aplicación, se puede concluir que la aplicación posee una situación frente al mercado inmejorable debido a la ausencia de competidores directos y la falta de innovación de los modelos alternativos que habitan el entorno digital.

2.2. Público objetivo

El público al que está destinada la aplicación no se encuentra indicado para un rango determinado de edades sino más bien para un público de carácter universal, comprendiendo desde la adolescencia hasta la senectud. En definitiva, está

indicado para usuarios aficionados del COAC que quieran compartir sus gustos, preferencias u opiniones y, además, tengan un conocimiento suficiente sobre el manejo de los dispositivos tecnológicos (ordenadores, tabletas o teléfonos móviles).

3. Diseño

3.1. Diseño Centrado en el Usuario (DCU)

Esta metodología utiliza al usuario como componente principal para el estudio del diseño de sus interfaces, realizando un análisis de las interacciones que éste tiene con el producto. La finalidad del DCU es otorgar a la aplicación del mayor grado de usabilidad posible ofreciendo al usuario final una experiencia óptima durante la utilización de la misma. Debido a cuestiones de tiempo y de la naturaleza del proyecto, hemos obviado una de las técnicas más importantes que se suelen llevar a cabo en un proyecto real, el test o pruebas con usuarios reales. Por este motivo, la fase de evaluación del diseño se ha realizado de manera retroactiva por cada proceso implementado. El DCU consta de las siguientes fases: análisis, diseño, prototipo y evaluación.

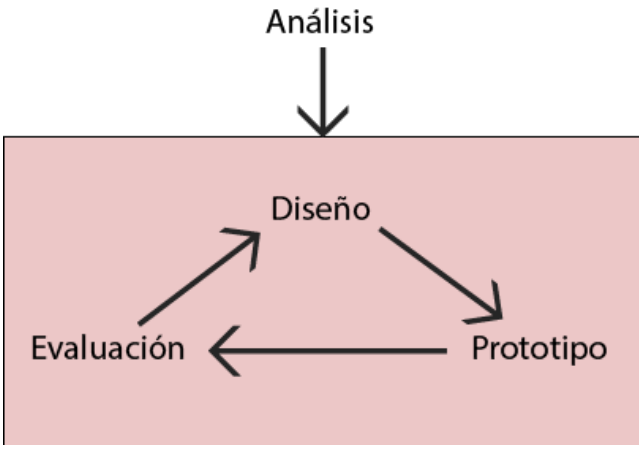

Figura 6. Diagrama DCU

3.1.1. Análisis

En esta etapa, el enfoque de estudio recae en la obtención de información de los usuarios potenciales vinculado al cumplimiento de los objetivos que se han planteado para la aplicación. Esta información estará formada por datos de los usuarios como sus intereses y/o necesidades, edades comprendidas, público objetivo, etc., y a la vez por datos pertenecientes a la aplicación como el tiempo de vida de la misma y características de su contenido.
En primer lugar, y como hemos referido anteriormente, la aplicación estará dirigida a un público universal con un abarcamiento del mayor rango posible de edades. Existen varios factores que acotan este amplio rango como el ámbito de su difusión, quedando aislada solamente a personas aficionadas al COAC o con cierto interés en el concurso como medios de comunicación, etc. Otro factor determinante es el grado de conocimiento que pueden poseer los usuarios potenciales en el uso de las tecnologías para llegar a utilizar la aplicación, cuyo resultado es crucial para alcanzar un nivel alto de usabilidad. Aunque la dirección de la aplicación se encuentre, por definición, dirigida a un público universal, existe un sector considerable de los aficionados al carnaval que pertenecen a grupos de edades tardías comprendidas entre los 55 y 70 años, hecho debido a la antigüedad que posee la fiesta en sí.

Haciendo referencia al producto, es importante hacer énfasis al tiempo de vida de la aplicación y a la actualización de sus contenidos. En este caso, el periodo de vida se encuentra bastante delimitado a la duración del concurso, donde los usuarios podrán hacer uso de las principales funcionalidades de la aplicación como es el sistema de puntuación. Una vez que el concurso finaliza, los usuarios no podrán desempeñar la acción por excelencia de la aplicación, puntuar a las agrupaciones, por lo tanto, el interés por la misma se prevé que sufra una caída drástica del tráfico en la web una vez pasado este periodo de tiempo.
Esta característica permite que las actualizaciones de contenido puedan darse de manera más espaciada identificando los intervalos de tiempo donde las actualizaciones deben ofrecerse con un ritmo más acentuado.

3.1.2. Diseño

Modelado del usuario

La etapa de diseño comprende las fases de modelado de usuario, de diseño conceptual y visual. Para alcanzar el óptimo modelado del usuario se debe trabajar con personas reales, factor muy importante debido a que facilita una visión objetiva de los requisitos y necesidades verdaderas de estas. Como se menciona anteriormente, no utilizamos información sobre un grupo amplio de personas, sino que realizamos una estimación basada en la experiencia real de un grupo muy reducido de personas, en nuestro caso se trata de dos personas. Como objeto de recopilación de información hemos realizado un estudio de casos de utilización (*scenarios*) de cada una de estas personas, en lugar de obtener información a través de historias de usuarios. A pesar de la utilidad que estas aportan, su función principal está indicada para entornos de trabajo ágiles o colaborativos.

Scenario 1

Nombre: Mª Dolores
Edad: 58 años
Profesión: Ama de casa

Descripción de la persona: Mª Dolores es una ama de casa que vive en Cádiz, casada y tiene dos hijos. Todos en su casa son aficionados al carnaval desde siempre y sigue todas las actuaciones de las agrupaciones del concurso que puede, ya que las sesiones suelen terminar muy tarde. Cuando no tenían conexión a Internet en su casa, siempre lo escuchaban por la radio y, cuando llegaban las semifinales, veían la retransmisión en diferido o en directo que ofrecían en Canal Sur TV. A ella le gusta comentar con los vecinos y amigos qué le ha parecido la actuación de una determinada agrupación, sobretodo, cuando le toca a su marido cantar con el coro carnavalesco. Uno de sus hijos, le anima para que utilice

Internet que retransmiten todas las fases del concurso en directo, sin necesidad de radio, pero ella no se aclara mucho con las tecnologías, aunque tiene ordenador en su casa (conectado con fibra óptica) y un *smartphone* que no utiliza mucho.

Scenario: Es el día de la primera actuación de su marido con el coro, y está muy nerviosa. Cuando concluye la actuación, Mª Dolores quiere saber que le ha parecido a la gente y como lo ha visto el público en el teatro y en sus casas. Por desgracia, al coro le ha tocado cantar en penúltimo lugar y son casi las 2:00 de la mañana. A la mañana siguiente, lo primero que hace después de desayunar es comprar el Diario de Cádiz para ver la clasificación que le han puesto en el medio de comunicación.

Scenario 2

Nombre: Elías
Edad: 35 años
Profesión: Enfermero

Descripción de la persona: Elías también es de Cádiz, aunque hace unos años se fue a Munich por motivos laborales, y también es un aficionado del carnaval desde siempre y que, además, salió muchos años como componente de chirigota. Sigue el carnaval a través de Internet y lo ve en directo siempre que las circunstancias se lo permitan. En caso contrario, graba las actuaciones y las ve al día siguiente. Elías se considera un tanto radical a la hora de defender ciertas agrupaciones que le gustan, aunque sus repertorios, ese año en concreto, dejen mucho que desear. Además, suele molestarle mucho los comentarios negativos hacía sus agrupaciones favoritas y detesta que tengan una mala puntuación por parte de los jurados no oficiales (medios de comunicación, etc.).

Scenario: El concurso ya va por la fase de semifinales y está que arde. Elías sabe que es muy difícil llegar a semifinales porque llegan muy pocas agrupaciones de cada modalidad, y dar un paso en falso a estas alturas significa el fin del concurso para la agrupación. Después de la actuación de una de sus agrupaciones, se conecta a Internet y le echa un

vistazo a todas las redes sociales, que él se encuentra registrado, y lee todos los comentarios sobre la agrupación y escribe mensajes de ánimo a los componentes y critica a los que hablan mal de ella.

Diseño conceptual

La definición de la Arquitectura de la Información (*AI*) y el diseño de la misma se llevan a estudio en esta fase de la metodología DCU. Las principales premisas que debe cumplir la AI es conseguir la organización de los contenidos y la estructura de navegación que tendrán las diferentes secciones de la aplicación.
Para llevarlo a cabo, requiere de dos documentos cuya importancia radica en que muestra cómo el usuario debe utilizar el sistema de navegación para llegar a una determinada sección de la aplicación. Estos documentos son los diagramas (*wireframes*) y el árbol de contenidos.
En primer lugar, debemos establecer la estructura de los contenidos a través de un esquema que muestra todas las secciones (páginas) que forman la aplicación y sus correspondientes niveles de profundidad (árbol de contenidos).

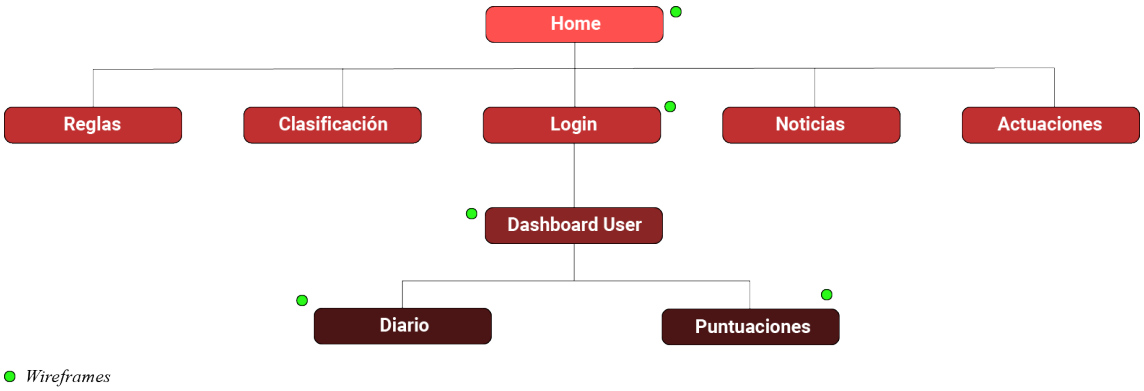

Figura 7. Árbol de contenidos

Los *wireframes* son prototipos de baja fidelidad (Lo-Fi) cuya función es establecer una pauta visual de la organización de los contenidos mediante la colocación de distintos elementos típicos de una aplicación web. Los diagramas no plasman ninguna pauta de diseño visual de la aplicación, siendo un aspecto que se trata en la etapa de confección del prototipo. A continuación, mostramos los *wireframes* más relevantes de la aplicación señalados en el árbol de contenidos. La principal razón de que no se muestren todas las páginas que componen la aplicación es debido a que comparten la misma línea de estilo, y las leves variantes que pueden tener no se consideran de fuerte transcendencia en el diseño e implementación programada.

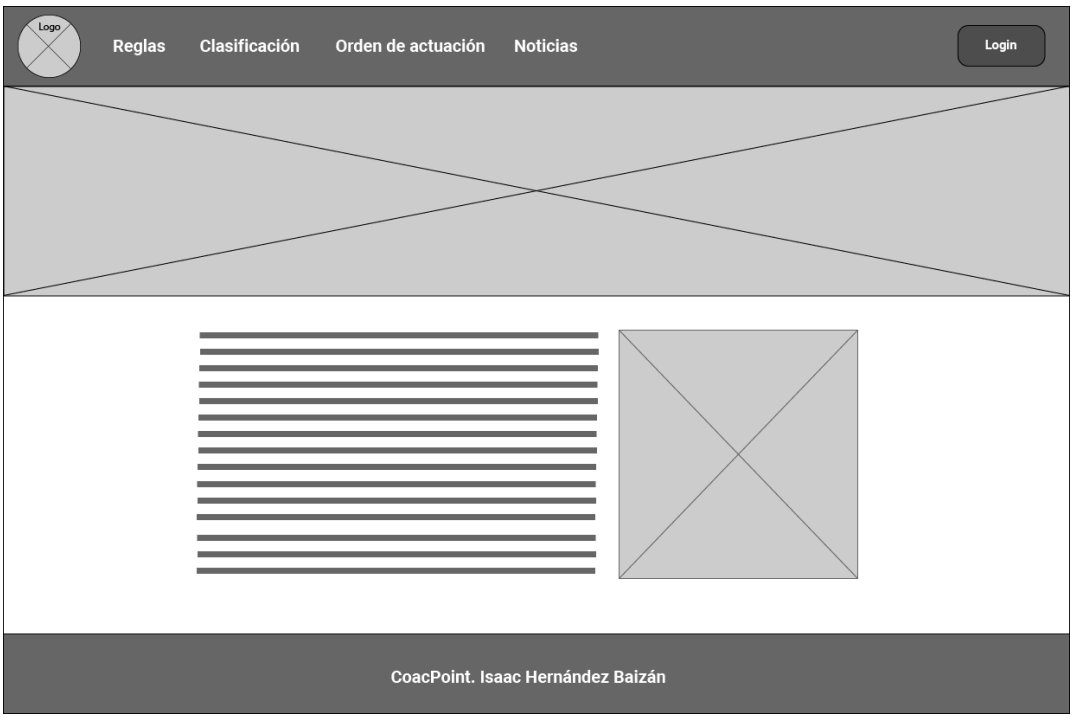

Figura 8. Wireframe Página Principal (Home)

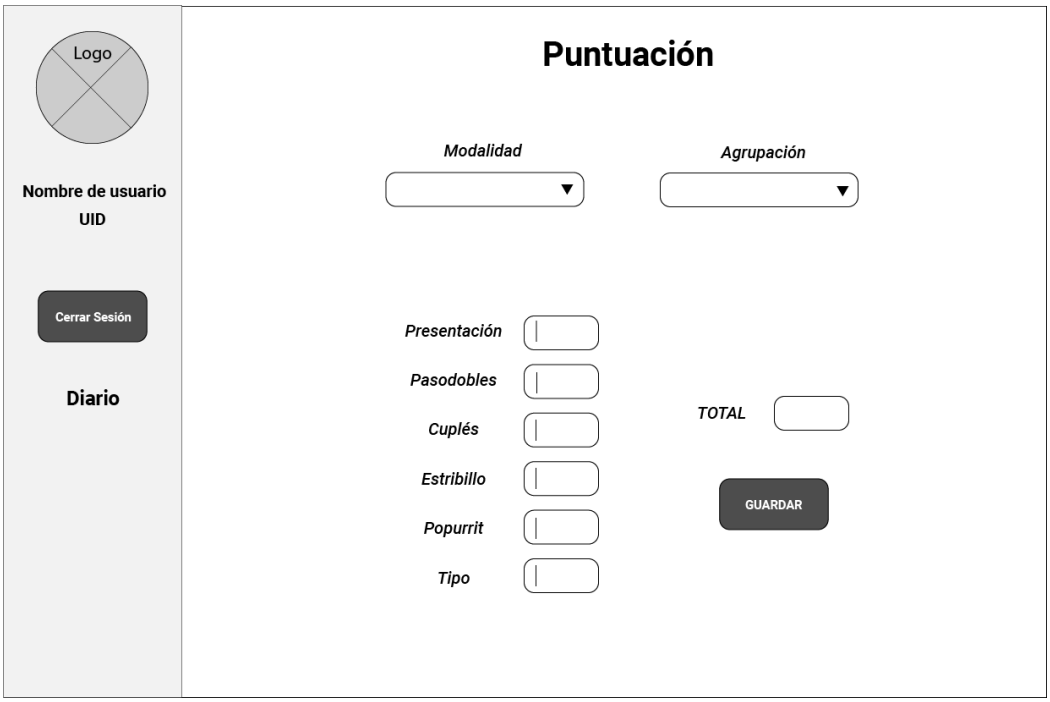

Figura 9. Wireframe Dashboard Usuario (Puntuación)

Figura 10. Wireframe Dashboard Usuario (Diario)

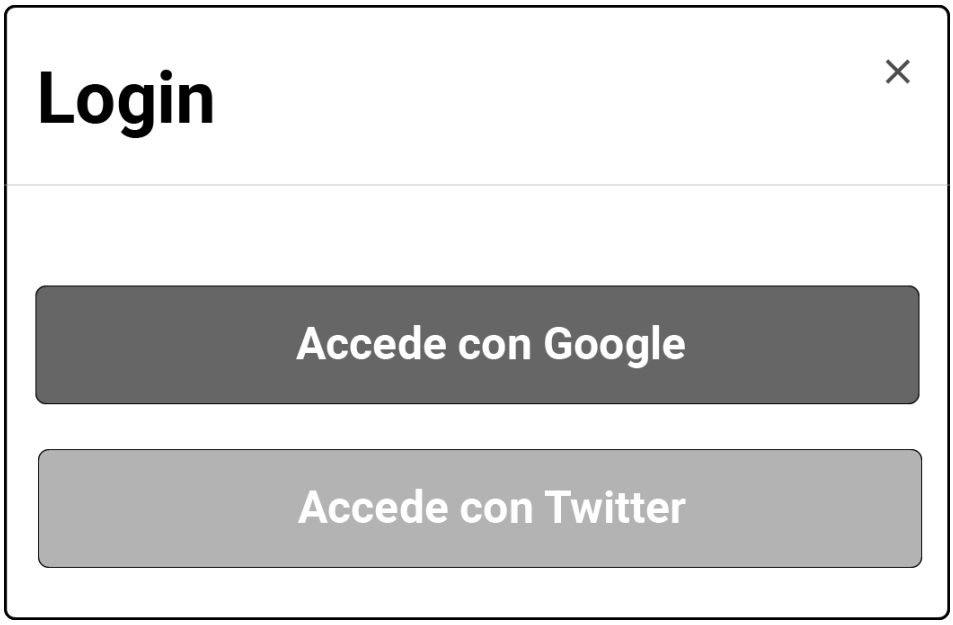

Figura 11. Wireframe Login

Diseño visual

Con la finalidad de conseguir una coherencia estilística apropiado a la hora de elaborar los prototipos de diseño que mostramos en el siguiente apartado, se ha llevado a cabo la elaboración de elementos visuales necesarios como el logotipo y estilos de botones, así como la toma de decisiones para la elección de colores de marca gráfica, tipografías e imágenes utilizadas.

***Estudio de logotipo**. La morfología del logotipo de la aplicación ha estado centrada siguiendo los patrones de diseño que posee el logotipo oficial del Gran Teatro Falla, sede donde se celebra el COAC.*

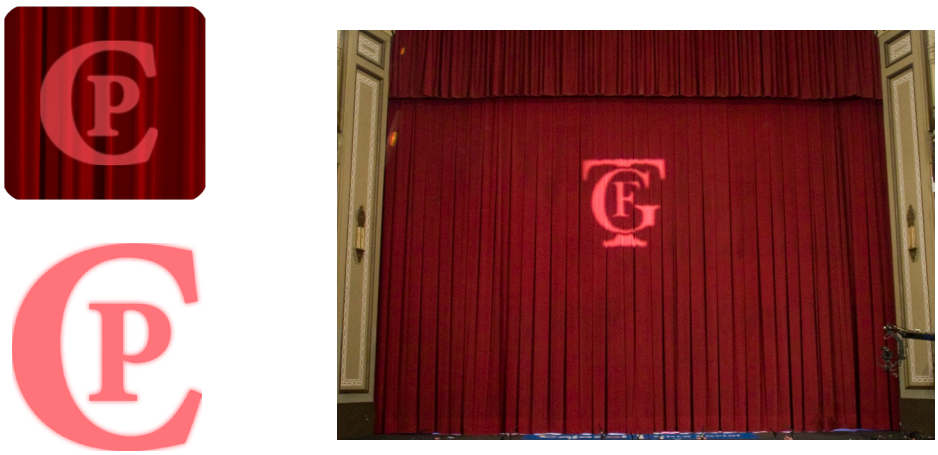

Figura 12. Fundamentos y creación del logotipo y sus variantes

***Gama cromática definida**. Vendrá dada por los tonos utilizados por el framework Bootstrap, tanto para los botones como para elementos de resaltado.*

Estados principales de los botones cuando el usuario se encuentra en el enlace (activo), y en los que no se encuentra (inactivo / normal) y la superposición del cursor del ratón (hover). El contraste establecido determina, de forma clara, un grado de usabilidad adecuado en la navegación.

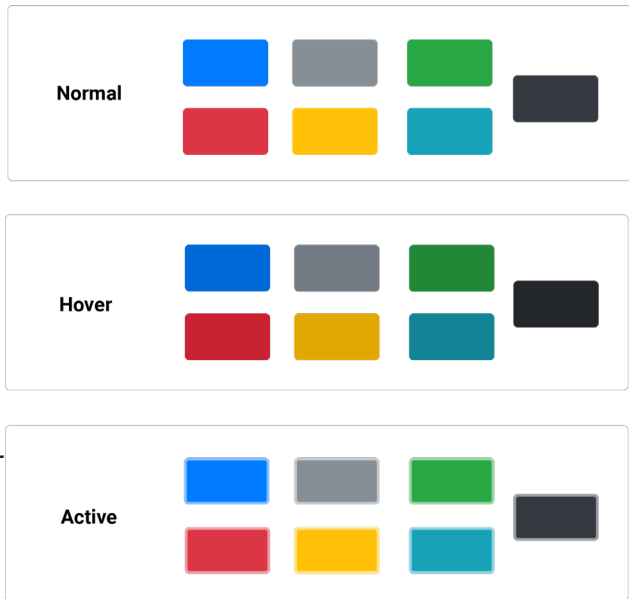

Figura 13. Colores para botones y otros elementos

***Tipografía**. Se han elegido los tipos de letras correspondientes a dos familias tipográficas: Roboto (sans serif – sin serifa) para facilitar la lectura en extensiones mayores de texto y detalles de información (tablas), y Martel (serif - con serifa) por tener patrones de diseño tipográficos muy similares a la forma de letra dibujada para el símbolo corporativo, produciendo un imagotipo coherente.*

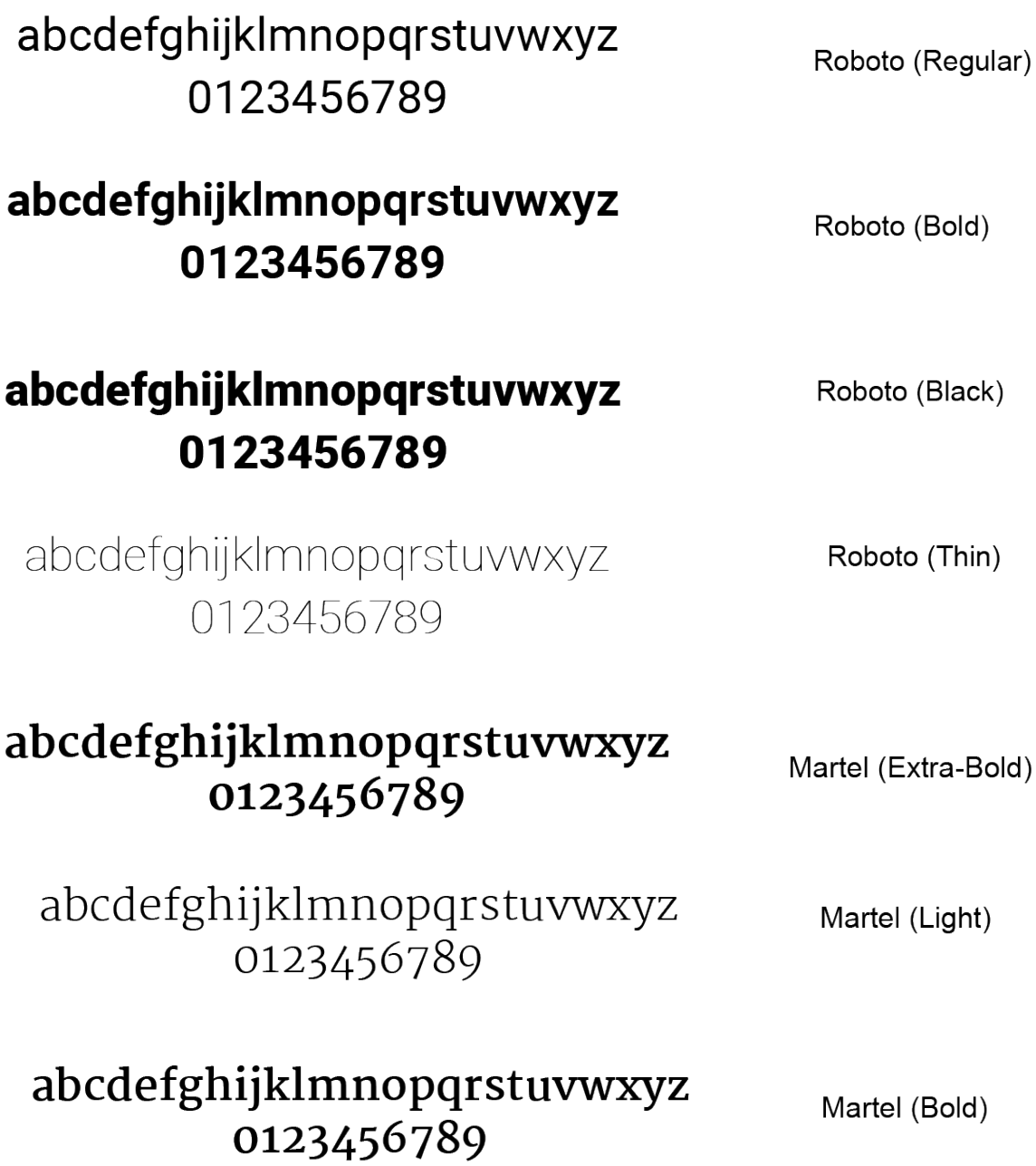

Figura 14. Familias tipográficas utilizadas

3.1.3. Prototipo (Hi-Fi)

Para la creación de los prototipos que seguiremos durante la fase de desarrollo o implementación de código, se han preparado, a partir de herramientas de edición de imagen, las imágenes que formarán parte de estructuras como, por ejemplo, el *header* y el *footer*.

Cabe destacar que se han seguido las estructuras de la arquitectura de la información establecidas en los wireframes descritos anteriormente. Los elementos con los que podrá interactuar el usuario, como los inputs de los formularios, botones, barra de navegación y estados transitorios se han fundamentado principalmente en la definición estilística de esos objetos aportada por el framework *Bootstrap*, con el principal objetivo de preservar el diseño responsivo para diferentes tamaños de dispositivos.

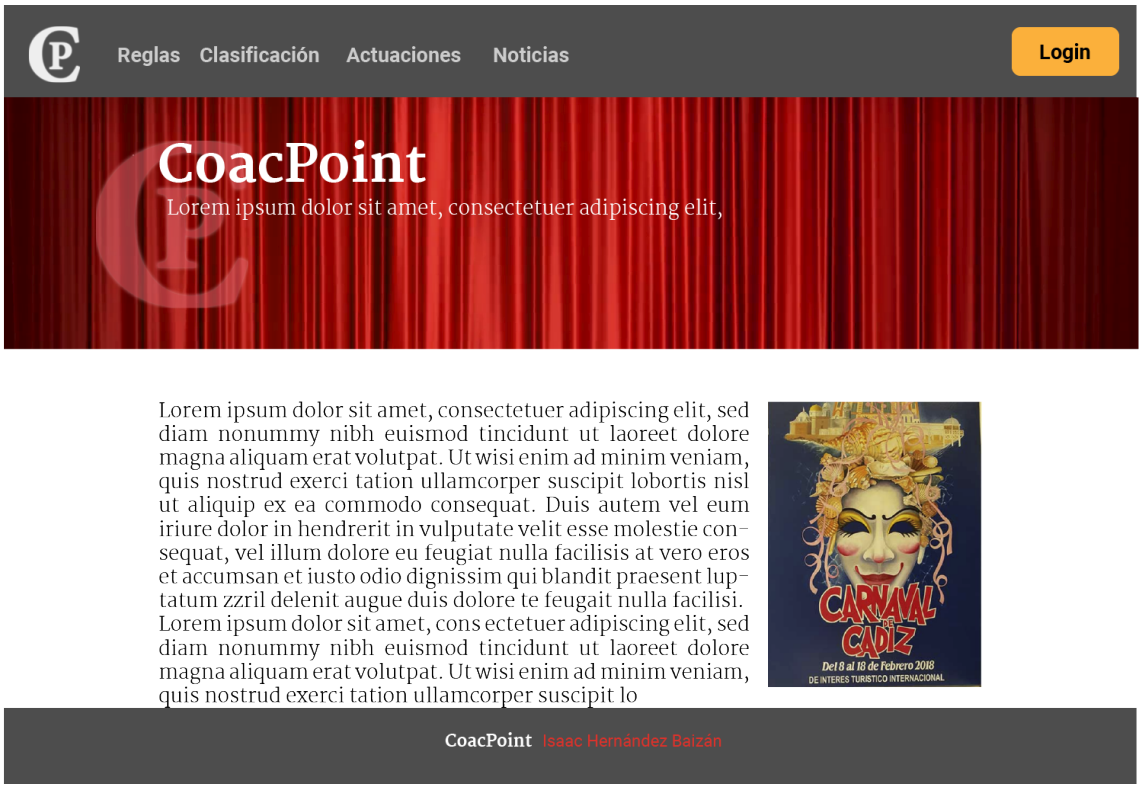

Figura 15. Prototipo página principal (home)

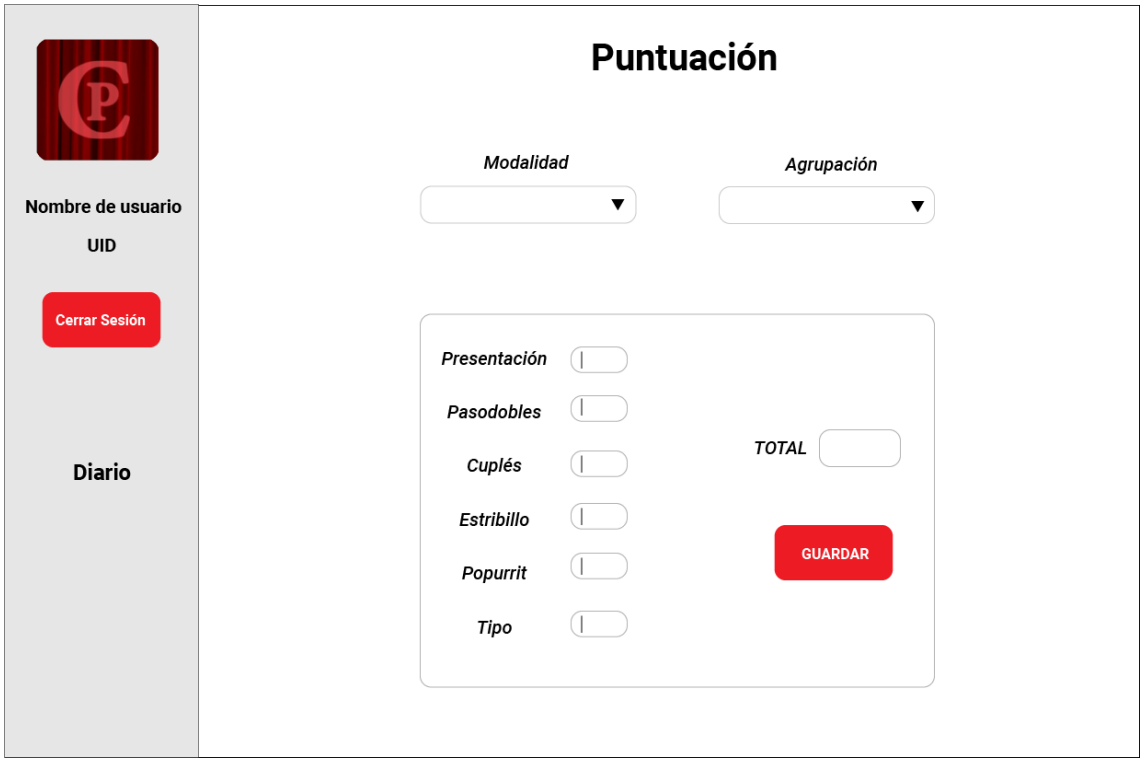

Figura 16. Prototipo dashboard usuario (puntuación)

Figura 17. Prototipo dashboard usuario (diario)

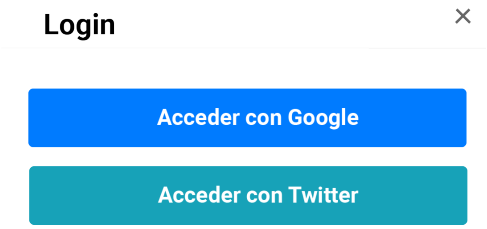

Figura 18. Login / Registro de usuarios

3.1.4. Evaluación de usabilidad y experiencia del usuario

La correcta evaluación de la usabilidad, está basada principalmente en la medición de parámetros o características de la aplicación para que el usuario pueda conseguir los diferentes objetivos que pueda plantearse a la hora de su utilización. Además de estos parámetros, existen otros factores que pueden ser también valorados como la navegación, uso del color, diseño de interfaces, etc. Sin realizar una evaluación mediante la aplicación de pruebas por parte de usuarios reales, principal mecanismo de medición de usabilidad, puede observarse que, a priori, cumplen los principios generales más importantes.

En primer lugar, la información ofrecida es concisa y facilita la comprensión de la temática y funcionalidad del sitio, otorgando una consistencia notable en los elementos de la aplicación. De esta manera, el usuario siempre sabrá las acciones que puede desempeñar y las que forman parte del sistema automatizado. Cabe destacar que existe una total coherencia entre los elementos de navegación y la información disponible en cada uno de sus apartados o secciones.

La aplicación se compone de dos bloques principales, claramente diferenciados según si el usuario está autenticado o no. Evidentemente, el aprovechamiento de la funcionalidad completa se consigue mediante el registro y acceso del usuario al sistema para que, de esta manera, pueda acceder a las secciones corres-

pondientes a la puntuación de agrupaciones, compartición de sus valoraciones, etc., cuyos contenidos se encuentran, en todo momento, adaptados al usuario. El otro bloque (usuarios no registrados), sólo muestra la parte informativa de la aplicación como la sección de blog o noticias, el orden de actuación de las agrupaciones y otra información representativa.
El motivo estilístico general (uso del color y elementos de diseño) adoptado en la aplicación, surge de elementos reales reconocibles por la mayor parte de los usuarios estableciendo una metáfora válida que no conlleva a cometer errores en la utilización, ya que existe una probabilidad muy remota de que un usuario no conozca ciertas características que acompañan al concurso como el lugar de celebración, su logotipo, etc.

Para la prevención de errores en el manejo, se tomarán medidas de autenticación de los tipos de datos introducidos por el usuario, para que no existan registros incoherentes en las bases de datos que puedan ser mostrados en la aplicación, produciendo desconcierto y, con ello, un descenso drástico de la usabilidad.

3.2. Ingeniería del software

3.2.1. Diagramas UML

El lenguaje UML es el lenguaje de referencia para la creación de distintos modelos de software, permitiendo reflejar las propiedades del mismo independientemente de las tecnologías que se utilicen para su desarrollo. Existen varias alternativas de uso que derivan del lenguaje UML, aunque mayoritariamente suele utilizarse para la realización de croquis del sistema, que es el método que abordaremos para llevar a cabo nuestros diagramas.

Diagrama de casos de uso

Este diagrama refleja la información del comportamiento del sistema, y permite mostrar las funcionalidades que ofrece el sistema para los diferentes actores (*stakeholders*) que interfieren en él. En nuestro proyecto, se encuentran dos actores principales: el aficionado/a y la figura del administrador.

Existen casos de uso que requieren la inclusión de otros casos de uso para reflejar las restricciones del sistema. En nuestro diagrama, para poder puntuar a las agrupaciones o compartir esa puntuación a través de redes sociales, requiere de la inclusión (<<*include*>>) de la autenticación de usuario, es decir, el usuario se debe identificar de manera correcta para poder realizar ciertos usos del sistema. La mayoría de los casos de usos reflejados pertenecen al mismo nivel (nivel usuario), sin embargo, tanto la autenticación de usuario como la del administrador pertenecen al nivel tarea y para relacionar casos de uso de diferente nivel requiere de la extensión (<<*extend*>>).

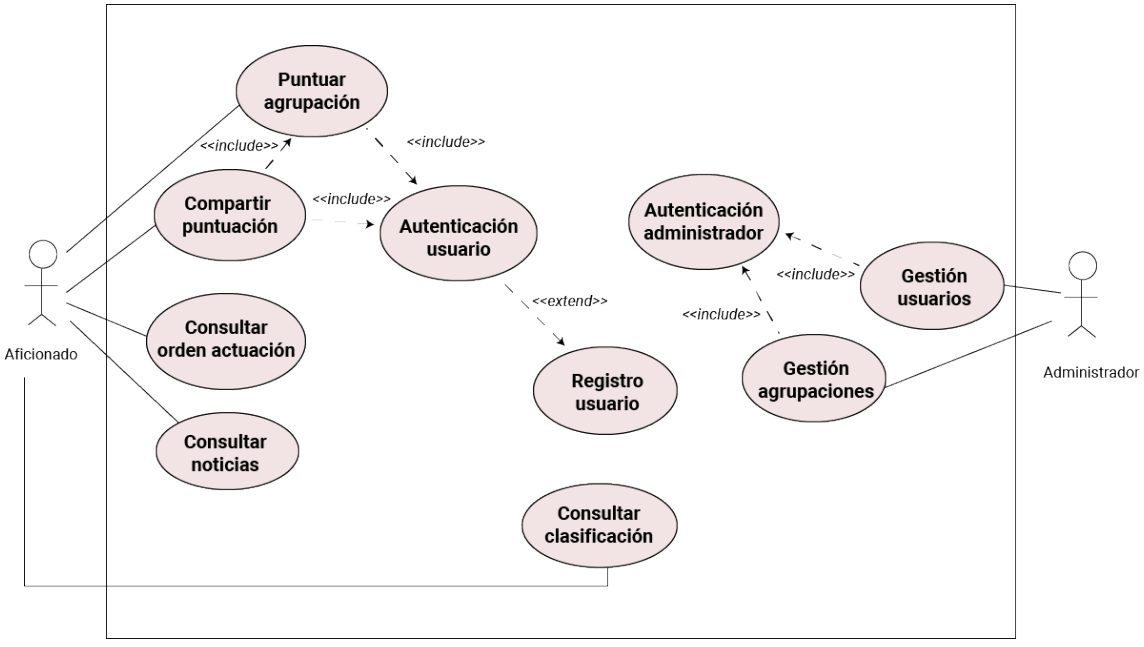

Figura 19. Diagrama de casos de uso

Diagrama de actividades

En este diagrama se muestran las acciones que se producen en cada caso de uso del apartado anterior. En nuestro caso, la representación diagramática estará centrada en los casos de uso “Puntuar actuación” y “Compartir puntuación”, por el principal motivo de que se tratan de las principales acciones que se realizan en la aplicación por parte de los aficionados.

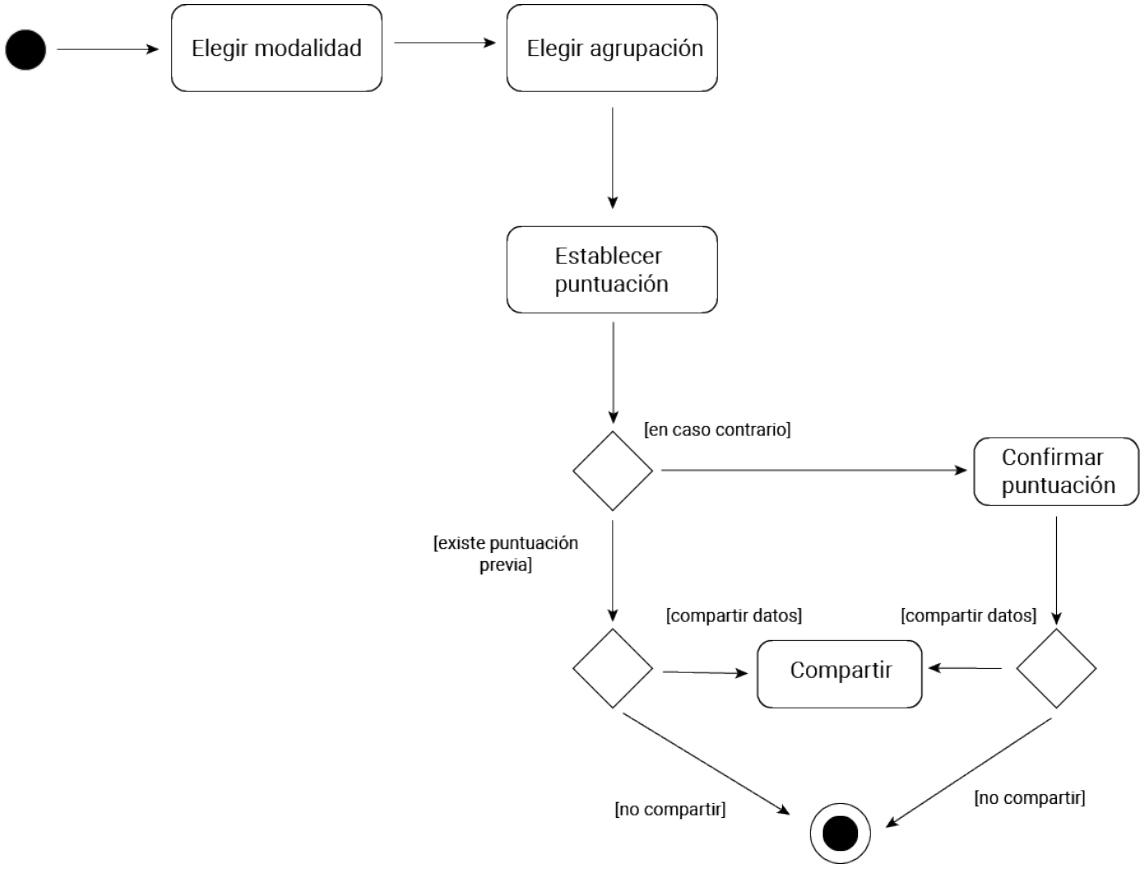

Figura 20. Diagrama de actividades

Diagrama de clases

El diagrama de clases es una representación visual de la estructura de las clases del sistema que conforman el software, al igual que sus métodos (acciones), atributos (propiedades) y sus interrelaciones (asociaciones, herencia, …), y son necesarios para documentar el modelo del dominio con los conceptos del mundo real que queremos representar. El principal cometido de un diagrama de clases es que ofrece una visión global de las funcionalidades del sistema diseñado y de qué manera puede ser elaborado.

En nuestro caso, existen algunas peculiaridades que sería necesario destacar:

- La clase asociativa (*Score*) nos ayuda a determinar que, por cada actuación de una determinada agrupación, habrá una puntuación asociada (realizada por un usuario).

- La actuación vendrá etiquetada con uno de los valores posibles que tiene su atributo ***fase***. Como sólo es posible que una actuación se realice en una de las cuatro fases del concurso, es conveniente utilizar una enumeración con el tipo de datos que se pretende moldear y sus respectivos valores.

- El otro punto es el caso de herencia (generalización) que se produce en nuestro diseño. Como una agrupación debe estar englobada, forzosamente, en uno de los tipos de agrupaciones, la clase *Group* debe ser abstracta.

En el siguiente apartado, podemos observar que el diagrama E-R de la base de datos, comparte la misma nomenclatura que en el diagrama de clases. El único aspecto en que difieren es que no se tratan de clases sino de entidades. Este es un ejemplo claro de la aportación que realiza un diagrama de clases para el diseño de software de cualquier componente de una aplicación.

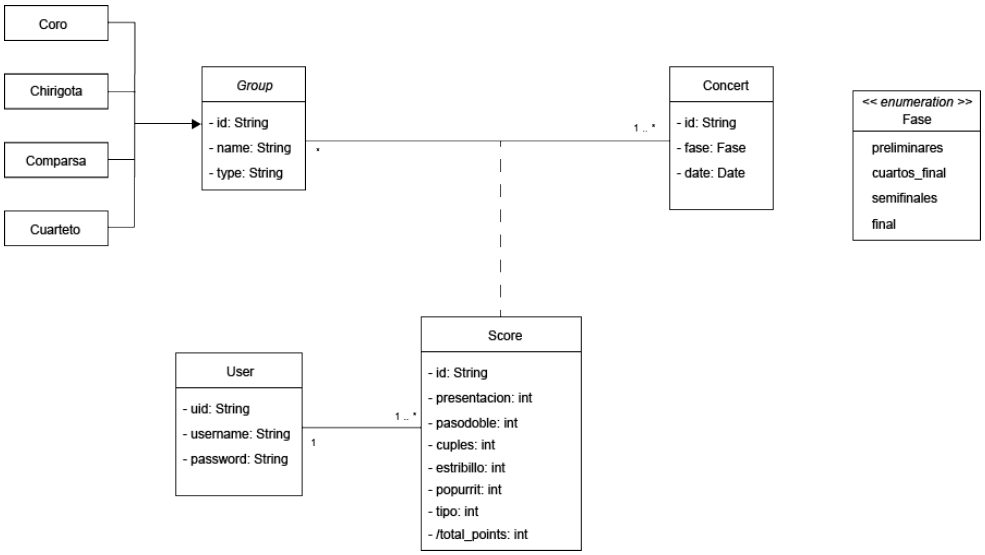

Figura 21. Diagrama de clases

3.2.2. Base de datos (Firebase)

Una de las principales características de la API que ofrece *Firebase*, viene marcada por la falta de necesidad de elaborar ninguna infraestructura de tablas y registros típica en la creación de una base de datos relacional (SQL). El motivo de la elección de *Firebase* es, principalmente, que se trata de una base de datos con una naturaleza técnica distinta a todas las temáticas sobre base de datos que se han trabajado durante todo el grado y, por otro lado, por la gran cohesividad que ofrece para la implantación de un entorno *back-end* con bases de datos en una aplicación de *Angular*. *Firebase*, en pocas palabras, es una base de datos de tipo no relacional (noSQL) que se aloja en la nube de Google y ofrece una sincronización de datos a tiempo real. Una característica muy especial de la *API*, es que todos los datos se estructuran a través de árboles de objetos *JSON* (nodos) y que pueden ser creados, modificados o eliminados, reflejando cualquier cambio en la aplicación, de manera inmediata.

No obstante, aunque nuestra base de datos no es relacional, estableceremos un modelo ER (entidad – interrelación) que define el diseño conceptual de la aplicación, cuyo planteamiento está centrado en la estructuración de la información y

es independiente del tipo de tecnología que use o implemente la base de datos. Una vez determinado nuestro modelo ER, pasaremos a la elaboración de los árboles *JSON*, mediante la API *Firebase Cloud Store*, con datos de prueba para comprobar el funcionamiento de la aplicación durante el desarrollo de la misma.

Modelo ER (Entidad – Interrelación)

Para establecer el diagrama del modelo ER, debemos concretar los **requisitos** de los principales *stakeholders* para la elaboración de nuestra base de datos:

- Para cada actuación de una agrupación sólo podrá recibir una puntuación por cada usuario. De la actuación nos interesa saber la agrupación que la ejecuta, la fase clasificatoria en la que está actuando, y un identificador único de la misma. También, de cada actuación de la agrupación, nos interesa almacenar la fecha en la que esta se produce. De la agrupación queremos tener un parámetro que le identifica de manera única, el nombre de la agrupación (clave candidata), la modalidad a la cual pertenece y el nombre de agrupación que llevó en el concurso del año anterior.

- Un usuario (registrado) puede calificar una actuación por cada agrupación y cada vez que participe (fase), es decir, si la agrupación supera la fase del concurso en la que se encuentre, pasará a la siguiente fase en la que también actuará. De cada usuario queremos almacenar el identificador único, el nombre de usuario (clave candidata), la dirección de correo electrónico y la contraseña para el sistema de autenticación de usuarios.

Otras restricciones y aspectos importantes

- Una vez el usuario haya puntuado y grabado la puntuación de la actuación, no se podrá modificar, al igual que tampoco está permitido la

puntuación de la agrupación antes de su actuación. La fecha límite para puntuar una agrupación, es la fecha a la que corresponda la comunicación del fallo del jurado oficial del COAC, es decir, la fecha del último día de cada fase del concurso.

- Las agrupaciones compiten por modalidades y cada agrupación sólo puede pertenecer a una sola modalidad (existen 4 modalidades: chirigotas, comparsas, coros y cuartetos). De esta forma, las agrupaciones de cada modalidad compiten entre ellas hasta que se declara la agrupación campeona del concurso en su modalidad. Por ejemplo, un cuarteto no puede competir con otra agrupación que no sea otro cuarteto.

- Todas las modalidades comparten las mismas características (atributos) de puntuación. La única característica relacionada que cambia de una agrupación a otra es el nombre del tipo.

Además de la asignación de herencia, merece un cierto interés la mención de la entidad asociativa repertorio. La interrelación repertorio almacena el repertorio de todas y cada una de las actuaciones que realizan las agrupaciones, pero, al convertirla en una entidad asociativa, nos permite mostrar que puntuación recibe el repertorio ejecutado en la actuación de una determinada agrupación. Así, la entidad puntuación no requiere de una clave primaria identificativa, ya que cada puntuación se identifica por la asociación entre cada agrupación y su actuación en concreto (véase que hace uso de dos claves foráneas: *id_group* e *id_concert*).

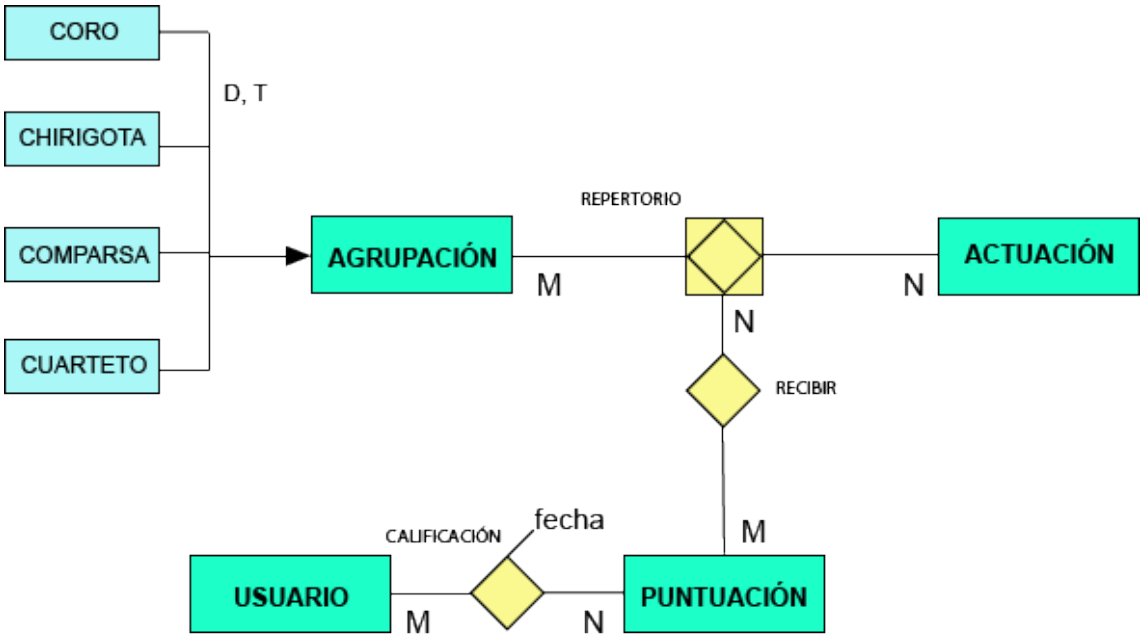

Figura 22. Modelo E-R de la base de datos

Entidades y atributos

Group (superclase)
id, name

Coro (subclase)*
Chirigota (subclase)*
Comparsa (subclase)*
Cuarteto (subclase)*

Concert
id, date, fase, id_group

Score
id, presentación, pasodobles, popurrit, tipo, cuples, estribillo, total_points, id_group, id_concert

User
id, username, email, password

* Comparten todos los atributos de la clase padre (superclase)

Estructura de Firebase Database (árboles JSON)

La etapa de desarrollo requiere la elaboración de una base de datos con información de prueba, con el fin de establecer la comunicación y el flujo de datos de manera correcta. Para ello, se elaborarán diferentes árboles de datos de tipo *JSON* correspondientes a las clases que se describen a continuación (*ver 3.2.2. Diagramas UML: diagrama de clase*).

Mediante la API de *Firebase*, creamos los nodos necesarios para la implementación inicial del proyecto. Al tratarse de una base de datos no relacional (NoSQL), permite crear los grupos de nodos mediante anidaciones de ella y, aunque no sea una base de datos de tipo relacional, debe seguir una determinada coherencia en sus anidaciones para que la trasmisión de información entre la aplicación y la base de datos sea lo más robusta posible.

Por eso, y como primer paso en la creación de elementos para la base de datos, crearemos sólo una pequeña lista de agrupaciones con sus respectivos atributos. Los usuarios serán introducidos en nuestra base de datos de forma directa a través de un formulario de registro presentado en la aplicación.

3.3. Arquitectura del software

3.3.1. Angular Framework

Angular es un conjunto de librerías *JavaScript* (*framework*) de código abierto y mantenido por Google, orientado a objetos y basado en *Web Components* (componentes reutilizables con características específicas relacionadas con la manipulación del *DOM* y plantillas HTML) que hace uso predilecto del lenguaje de programación *TypeScript* (TS). Por definición, *TypeScript* es un conjunto de colecciones (superconjunto) de *JavaScript* (JS) creado por Microsoft, cuyo objetivo es simplificar la escritura de código JS, otorgando una mayor rapidez y eficiencia para elaborar aplicaciones, sin renunciar a las características y potencia de este último. Para ello, se realiza una compilación del código escrito con TS a código JavaScript, para la posterior interpretación por parte de los navegadores webs.
Una de las principales características de Angular, es permitir la carga de contenidos procedentes de distintos componentes sin necesidad de refrescar el navegador, ya que su funcionalidad HTTP está basada en la interfaz `XMLHttpRequest` (al igual que la tecnología *AJAX*).

Angular está indicado principalmente para el desarrollo del diseño de la aplicación e interacción con el usuario (desarrollo *Front-End*) cuya ejecución se realiza del lado del cliente. No obstante, Angular podría orientarse al desarrollo *Back-End* mediante librerías especializadas como *Nodejs* y trabajar así de lado del servidor.

Figura 23. Logotipo de Angular

Arquitectura de Angular

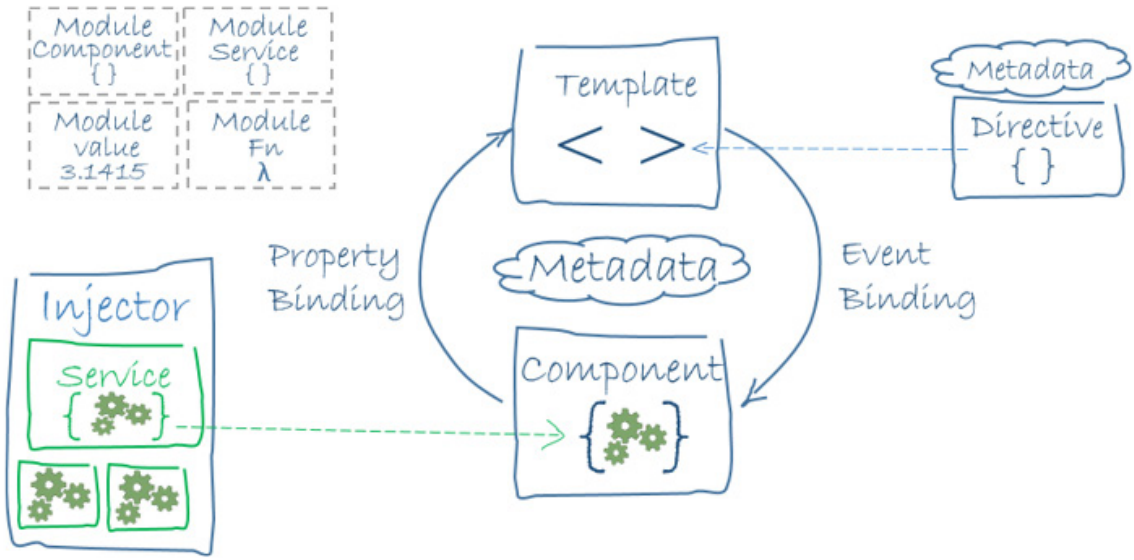

Figura 24. Esquema de la arquitectura de Angular (https://angular.io/guide/architecture)

A continuación, la definición que haremos sobre la arquitectura de este *framework* consistirá en la enumeración de las principales partes de este sistema y una explicación, con carácter muy preliminar, de los mecanismos que se utilizan para la interacción entre ellos.

La arquitectura de software que posee una aplicación de Angular está basada, en una estructura modular compuesta por **módulos** llamada *NgModules*, cuyo decorador utiliza el mismo nombre `@NgModule`. Una aplicación debe contar, como requisito fundamental, con un módulo raíz (*root Module)* cuyo nombre por defecto es `AppModule`. Un módulo no es más que un bloque de código con una funcionalidad específica. Su principal cometido es realizar la carga de la aplicación y mostrar todas las dependencias que se vayan incorporando además de albergar componentes y servicios necesarios para el desarrollo de la aplicación, así como vincular las posibles librerías externas que se deseen usar.

Además de los módulos, existe un elemento esencial en la arquitectura de Angular, son los **componentes**. Estos se asemejan a la definición de controlador en la *arquitectura MVC* y su función principal es gestionar las vistas dejando la lógica de la aplicación a cargo de los servicios. Es por eso, que siempre se encuentran vinculados a ellas, habiendo un archivo con extensión HTML (*template*) por cada componente creado.

Cada componente tiene un ciclo de vida desde su creación hasta su destrucción, pasando por fases de comprobación de vistas y componentes, inicialización de directivas y detección de los cambios que puedan ocurrir. *Angular* se encarga de crear, actualizar y destruir componentes a medida que el usuario utiliza la aplicación.
Las vistas en *Angular*, están compuestas por 3 tipos de elementos: componente, metadatos y *template* (plantilla). Un **template** o plantilla se compone de código HTML y hace uso de una sintaxis específica para trabajar con la información que se procesa en el componente (*data binding*). La función principal del *template* es determinar cómo se mostrará el componente y sus datos.

Los **metadatos** (*metadata*) son un tipo de datos que determinan la forma en que se procesa una clase, en nuestro caso un componente. Sin los metadatos, un componente sólo sería una clase corriente. Utilizan el decorador `@Component` y pueden ser de diferentes tipos, entre ellos están: selector (determina el nombre de la etiqueta HTML que servirá para instanciar el componente en el template), templateUrl (dirección relativa de la ubicación del template), *template* (código HTML directo situado entre *backticks* ``) y *providers* (servicios que realiza la inyección de dependencias).

Data Binding (asociación de datos) es el mecanismo que utiliza Angular para coordinar partes de un *template* con partes de un componente. Existen 4 formas de direccionar los datos:

• Interpolaciones (hacia el DOM): muestran los valores de propiedades de un componente. Ejemplo: `{{ user.name }}`

• Asociación de propiedad (*property binding* – hacia el DOM): establece una propiedad del componente a un elemento de la vista.

```
<app-home [usuario] = "usuarioSeleccionado"> </app-home>
```

• Asociación de evento (*event binding* – desde el DOM): a partir de un evento establecido en un elemento de la vista, llama a un método o establece una propiedad de un componente.

```
<li (click)="seleccionarUsuario(usuario)"></li>
```

• Asociación de las dos maneras (*Two-way binding*). Combina las dos formas de direccionamiento (hacia y desde el DOM), de manera que permite establecer propiedades a un elemento y actualizarlas al detectar cualquier cambio en el evento.

```
<input [(ngModel)]="usuario.nombre">
```

Las **directivas** (*directives*) son instrucciones específicas sobre cómo debe producir la renderización de cada template alterando elementos del DOM. (*Document Object Model*). Las directivas más utilizadas son:

• Directivas estructurales (***NgIf / *NgFor / NgSwitch***): alteran la capa de presentación de la página al eliminar, añadir o reemplazar elementos del DOM.

• Directivas de atributo (***NgClass / NgStyle / ngModel***): alteran la apariencia o comportamiento de elementos HTML, atributos, propiedades y componentes.

Los **servicios** son clases con una función concreta y bien definida. Se utilizan para encapsular la lógica de la aplicación, dejando a los componentes como gestores de los contenidos de los *templates*. Un servicio puede tener asignada cualquier función, pero una de sus funciones más destacadas es la de realizar operaciones *CRUD* (*create, read, update y delete*), llevadas a cabo por el protocolo *HTTP*, para modificar o consultar la base de datos. Los componentes obtienen los servicios a través del motor de inyección de dependencias de *Angular*.

3.3.2. API REST: Firebase

La arquitectura *REST* (*REpresentational State Transfer* – **Transferencia de Estado Representacional**) se define como una interfaz entre sistemas que utilizan el protocolo HTTP para la obtención de datos y otras operaciones.

Firebase es una suite de productos de software enfocada al desarrollo de aplicaciones para las principales plataformas: *Android, iOS, Web, Unity* y *C++* entre otras. Estos productos, llamados kits de desarrollo de software, (*SDK - Software Development Kit)* son herramientas que facilitan el desarrollo de una temàtica o funcionalidad concreta. Para nuestro proyecto, utilizaremos el SDK de *Firebase Cloud Firestore,* que es considerada como el siguiente paso evolutivo de su compañera *Realtime Database (RD)*, ofreciendo un servicio REST-full más avanzado manteniendo todas las características de RD, con la adición de otras funciones con otros servicios de nube (como Google Cloud Platform). Esta API se caracateriza por emplear bases de datos no relacionales (NoSQL). Entre sus principales características podemos citar las siguientes:

- Permite la sincronización de datos en tiempo real haciendo uso de la arquitectura REST (HTTP).

• Almacenamiento de datos en formato *JSON* (*JavaScript Object Notation*).

• La aplicación sigue estando disponible, aunque pierda la conexión con la base de datos.

• Su lenguaje se construye a partir de reglas de seguridad que definen la forma de estructuración, lectura y escritura de datos para aumentar la protección de los mismos.

• Proporciona una gran escalabilidad para la creación de proyectos de cualquier tipo.

• Soporte para la programación del lado del cliente y del lado del servidor.

Su mecánica de funcionamiento entra en acción a partir del acceso del usuario, ya sea desde un dispositivo móvil o a través de un navegador. La gestión de la base de datos se complementa con la autenticación de usuarios, para la cual *Firebase* dispone de un SDK específico que utilizaremos a través de la librería *angularfire2*.

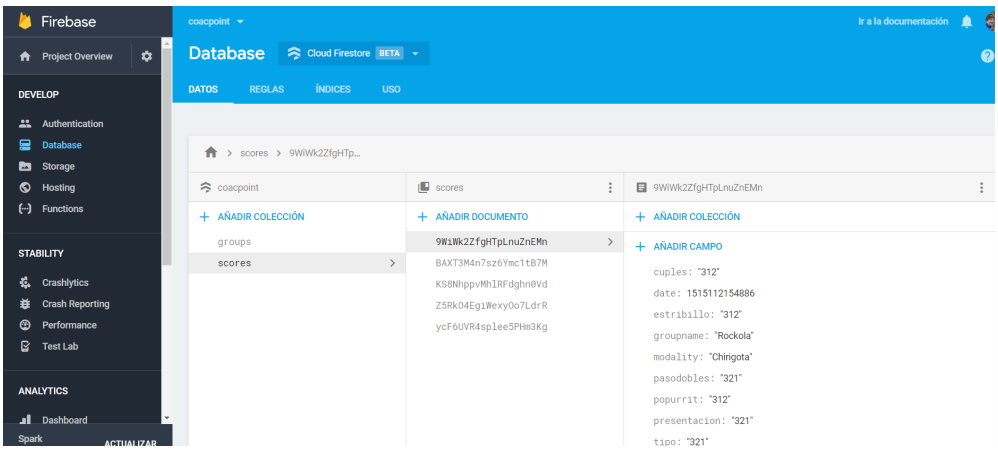

Figura 25. Dashboard de Cloud Firestore

4. Desarrollo

El ciclo de la fase de desarrollo transcurre de manera paralela a otras tareas de investigación e incidencias creadas durante la vida del proyecto, fruto de la retroalimentación continua, producida por tareas de reanálisis sobre el contenido global del trabajo.
En primer lugar, y antes de comenzar con la descripción de los dos grandes bloques de los cuales se compone la aplicación, se introducen las definiciones de las APIs utilizadas durante el proceso y sus características más importantes.

4.1. APIs en la implementación

Bootstrap 4
(*https://v4-alpha.getbootstrap.com/*)

Bootstrap es una librería (*framework*) creado por Twitter, cuya función principal es brindar al desarrollador un entorno de creación fluido, sencillo y soportado por todos los navegadores, para la construcción de aplicaciones web, a partir de lenguajes nativos como HTML5, CSS3 y JS. Entre su elenco de características y funcionalidades, una de las más destacadas (de ahí su fama mundial) es su sistema de rejilla de 12 columnas (*grid system*) para la producción de diseños responsivos en múltiples dispositivos.

Siempre, este framework ha tenido como filosofía principal tener en cuenta el desarrollo en dispositivos móviles antes que los demás (*mobile-first*), pero además, en su versión 4, el sistema de rejillas incorpora un nuevo componente llamado

FlexBox encargado de gestionar el layout de la aplicación mediante el alineamiento de columnas y elementos, justificación de contenidos, etc.
Además de los lenguajes nativos, este framework ofrece una integración robusta con *jQuery* y *Sass* (preprocesador de CSS) para la personalización y creación de temas de diseño y utilización de plugins con funcionalidades variadas.
Prácticamente, casi la totalidad de los estilos de los elementos interactivos de la aplicación (botones, navegación, formularios, …) han sido extraídos de la librería de Bootstrap.

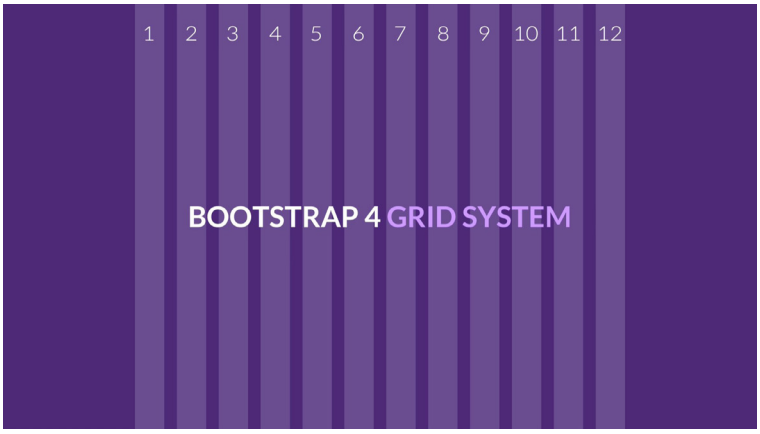

Figura 26. Sistema de rejillas de Bootstrap 4

Angular CLI
(*https://cli.angular.io*)

Como su propio nombre indica, se trata de una interfaz de línea de comandos (*Command Line Interface*) diseñada por y para la librería Angular. Gracias a esta herramienta oficial, permite la creación de aplicaciones en un entorno de desarrollo, a través de un servidor local – `ngServe`), además de la construcción de infraestructuras internas como componentes y servicios entre otras, de una manera rápida, sencilla y eficaz mediante la introducción de comandos en la consola del sistema operativo. Cabe mencionar que también permite realizar tareas de mantenimiento de código y ejecutar un entorno de pruebas (tests).

AngularFire2

(*https://github.com/angular/angularfire2*)

Es una librería oficial de *Angular* encargada de gestionar conexiones y transmisión de datos con *Firebase*. Se encuentra adaptada para los dos tipos de bases de datos que permite crear, *Cloud Firestore* y *Realtime Database*, basados en el almacenamiento de datos en la nube. En el proyecto, se hace uso del tipo de base de datos *Realtime Database* cuya infraestructura disfruta de una versión estable, al contrario que *Cloud Firestore*, que aún se encuentra en fase beta de desarrollo.

Entre sus funciones, destaca la posibilidad de proporcionar servicios de sincronización con la autenticación de usuarios que ofrece Firebase, que cuenta con sistemas de configuración de acceso y registro vinculados con las principales redes sociales: Facebook, Twitter, Github, LinkedIn, además del, ya rudimentario, acceso a través de elementos de formulario (*email / password*).

consiste en la sincronización de datos en tiempo real haciendo uso de la API Rest-full de *Firebase*. Esta sincronización se produce gracias a la figura de los observables, que son un tipo de patrón de diseño de software cuyo cometido es la ejecución de una instrucción o instrucciones programadas cuando se realiza algún cambio a nivel de la variable que sea asignada como observable. Más adelante, en la fase de desarrollo, explicamos este proceso con más detalle.

Angular Share Buttons

(*https://murhafsousli.github.io/ngx-sharebuttons/#/*)

Esta API permite al desarrollador añadir botones con funciones de compartición de contenidos basados en la definición de directivas de Angular.

Partiendo de una simple importación de módulos, permite customizar, de for-

ma amplia, el aspecto de los botones interviniendo en la modificación del color, formas y estilos de botón, textos, iconos e incluso la integración de contadores oficiales de las respectivas aplicaciones que representen.
Se trata de una librería con licencia MIT, cuyo repositorio se encuentra en Github. En su URL, alberga una herramienta para realizar pruebas de diseño y contenido del botón que se quiera incorporar al código de la aplicación.

Nota: Los enlaces no funcionan porque requiere que la aplicación se encuentre alojada en un servidor remoto. El único aspecto que tendríamos que implementar, una vez nuestra aplicación se encuentre en un entorno de producción, es la creación de etiquetas meta para añadir al contenido compartido un título, imagen, descripción, url y otros parámetros.

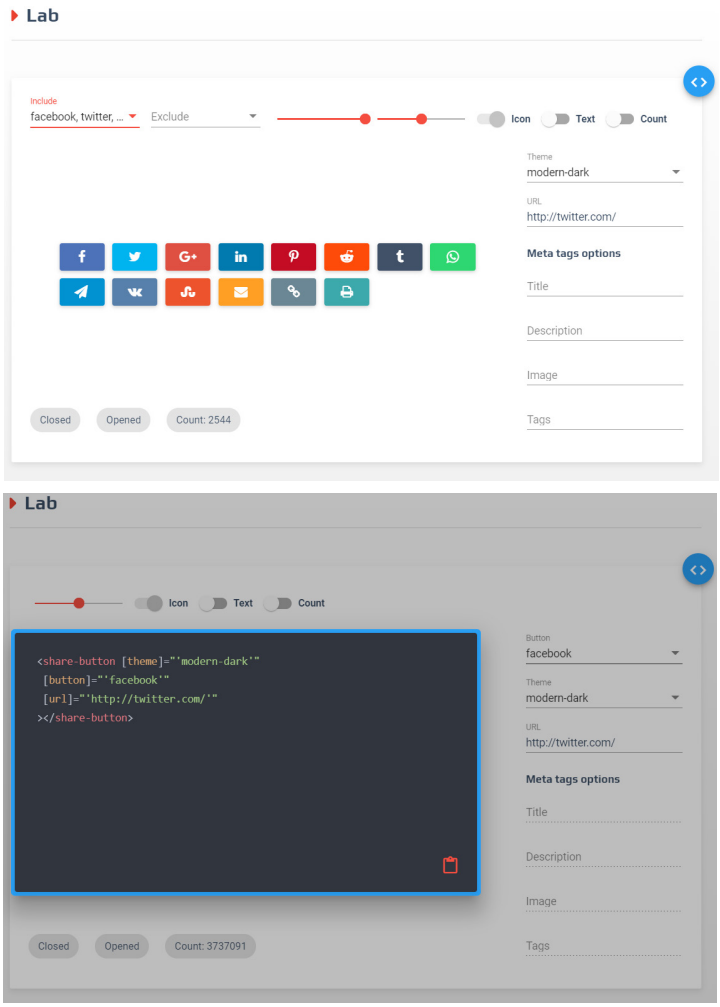

Figura 27. Interfaz de edición de Share Buttons

Git & GitHub

(https://git-scm.com/)

(https://github.com/)

Un aspecto importante que merece una dedicación necesaria es el control de versiones. *Git* es el sistema de control de versiones en sí mismo y se ejecuta en un entorno local, mientras que *Github* es una plataforma web que se encarga, inicialmente, de almacenar los repositorios creados con *Git* en la nube. La unión de estas dos herramientas, brinda al desarrollador una considerable mejora para almacenar cada uno de los cambios (*commits*) que se realizan en el código del proyecto, con el fin de gestionar las diferentes versiones que va alcanzando la aplicación y evitar incidencias irreversibles durante la implementación.

Entre las principales características que ofrece *Git* al desarrollador, se encuentra la creación y uso de ramas (*branches*), fusión (*merge*) de las mismas con la rama principal del proyecto, agregar colaboraciones de otros usuarios, posibilidad de retroceder al punto anterior de la implementación que se desee (función *recovery* o *backup*), entre otras.

En nuestro proyecto, hemos utilizado este sistema desde la interfaz que nos ofrece *Github Desktop* pues no requiere conocimientos de *Git* a nivel de línea de comandos y permite realizar las acciones más comunes que suelen emplearse en el control de versiones.

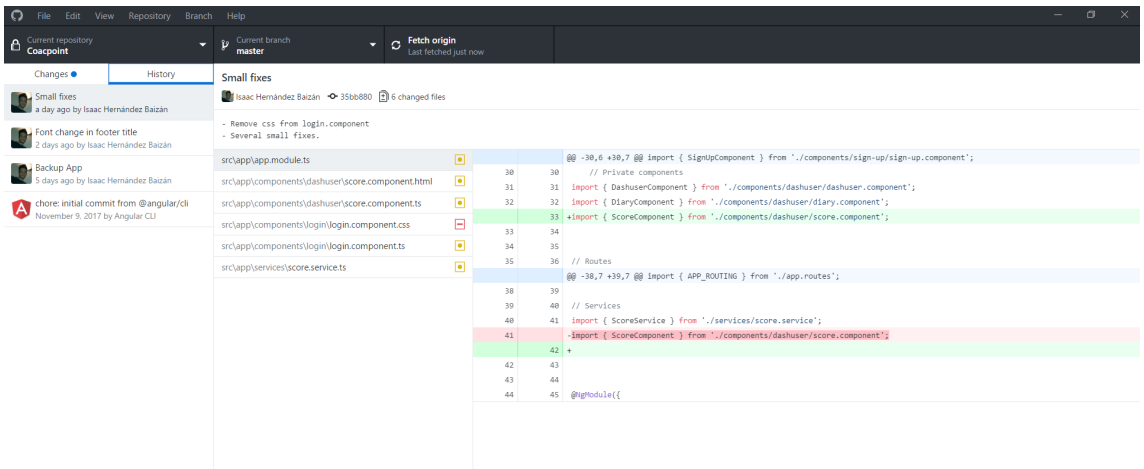

Figura 28. Control de versiones mediante Github Desktop

4.2. Front End

El planteamiento general del proyecto se ha llevado a cabo a partir de uno de los principales objetivos formulados, profundizar en el desarrollo *front-end* mediante el aprendizaje del framework *Angular*.
Esta terminología anglicista es utilizada de manera frecuente en el mundo del software y, en concreto, en el desarrollo web. Surge de la conveniencia de separar los componentes que conforman la parte que se encarga de la representación visual de la aplicación (diseño) y de la interacción con el usuario, incluyendo en esta, los datos que se reciben en la entrada que realiza el usuario a través de los diferentes elementos de la aplicación (*front-end*), y la parte lógica encargada del procesamiento y gestión de los datos, comunicación con la base de datos, sistemas de administración y componentes dinámico, entre otros. (*back-end*)

Como hemos mencionado al principio de la sección de desarrollo, la división de bloques que establecemos en la aplicación es realizada en referencia a la necesidad de autenticación (o no) por parte del usuario, de manera que si el usuario no es autenticado previamente sólo podrá acceder a los contenidos públicos de carácter general y, en caso contrario, podrá acceder a los contenidos privados y desempeñar las funcionalidades que confluyen en la sistemática principal del juego que se propone como centro holístico de la idea del proyecto.

4.2.1. Creación del proyecto

La creación de los archivos necesarios para llevar a cabo un proyecto en Angular, parte de la utilización de la interfaz de la línea de comandos (*Angular CLI)* comentada anteriormente. A partir de esta, y con un sencillo comando creamos la aplicación y la estructura general de archivos, entre los cuales se incluyen los módulos de nodejs iniciales necesarios además de los componentes iniciales para poner en marcha el proyecto.

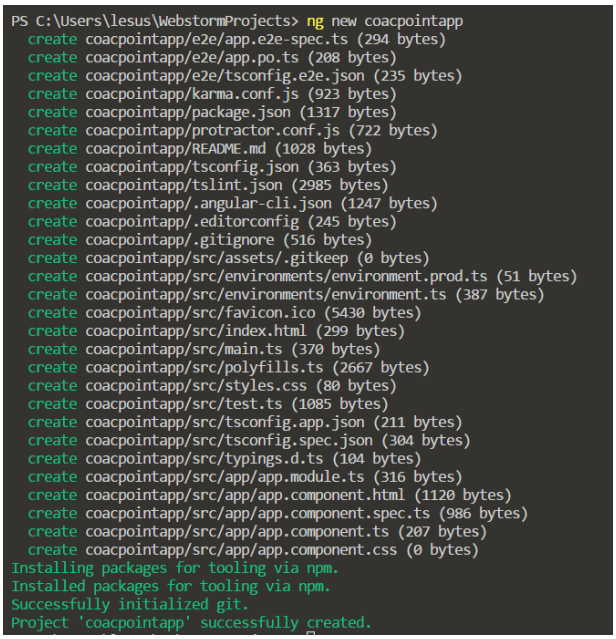

Figura 29. Creación de una aplicación en Angular (vista de terminal)

Puesta en marcha del servidor local (webpack)

Angular cuenta con un servidor local para ejecutar las innumerables pruebas que se realizan durante la fase de desarrollo del proyecto. Su ejecución se realiza a través de la consola del sistema operativo o el terminal del IDE que se este utilizando.

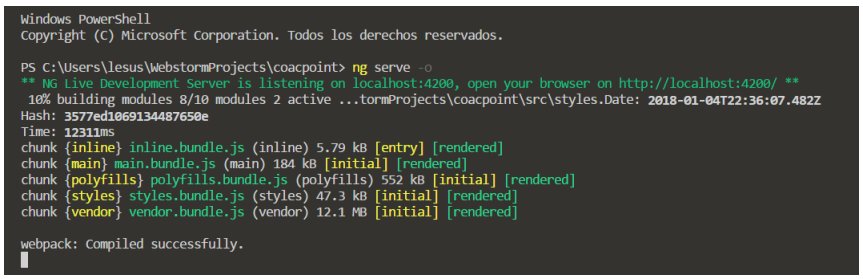

Figura 30. Puesta en marcha del servidor local

Componentes y plantillas HTML (templates)

Las diferentes partes que componen la aplicación serán creadas a partir de componentes que están asociados a unos archivos HTML denominados plantillas (a partir de ahora las denominaremos con su nombre en inglés – *templates*).
La creación de los distintos componentes también se realiza de manera sencilla a través del uso de *Angular CLI* con un sencillo comando,

```
ng generate component <<ruta de destino>>
```

Al crear un componente, se crea una carpeta que contiene tres archivos base: el componente (archivo con extensión de .ts -*TypeScript*-), el *template* (.html) y un archivo de estilos (.css), todos ellos vinculados entre sí de manera automática, gracias a *Angular CLI*, para una mayor facilidad en su manejo. Cabe destacar que, además de estos tres archivos, crea otro con extensión .spec que cuya utilidad no está reflejada no se utilizan en nuestro prototipo de proyecto ya que se trata de temas muy avanzados del *framework*.

Estructura general de la aplicación

Con esta breve explicación del proceso rutinario de creación de componentes, pasaremos a enumerar los diferentes apartados de la misma.
El bloque de sección en referencia al usuario no autenticado contiene los siguientes apartados:

- Un componente y un template por cada página perteneciente al menú principal de la aplicación: home, reglas, clasificación, actuaciones y noticias.

- Un apartado de elementos comunes (carpeta shared) que se comparten en cada una de estás páginas como son el header, footer y la barra de navegación.

- Un componente y un template para la creación del formulario de acceso (login) y de registro (sign-up) de usuarios.

El bloque referente al usuario autenticado, por algunos de los métodos disponibles de autenticación, contiene los siguientes apartados:

- Componentes y plantillas unificadas en una misma ruta (dashuser) que conforman el tablero (dashboard) de funcionalidades disponibles para el usuario. Estos componentes son: dashuser, diary y score.

- Servicio que se encarga de realizar las operaciones CRUD con la base de datos y de gestionar la autenticación de usuarios.

-Las interfaces que definen el tipo de datos para ciertas variables o elementos.

Aparte de estos elementos propios de esta aplicación en concreto, existen otros que son indispensables para asegurar un correcto funcionamiento y comunes a todas las aplicaciones en Angular. Estos elementos esenciales son:

- **`app.component`**: componente principal de la aplicación, cuya composición de archivos es similar a la de los demás (ts + html + css). Todos los contenidos producidos durante el desarrollo, aparecerán reflejados en el template de este componente (`app.component.html`) pues se trata de la vista general de la aplicación.

- **`app.modules.ts`**: archivo donde se realizan las importaciones de todos los archivos que forman la aplicación: componentes, servicios, archivo de rutas y los módulos necesarios para la ejecución de acciones específicas.

Además de estos elementos, hay otros que forman parte de la infraestructura básica de Angular que comentaremos a continuación, como son la hoja de estilos utilizadas y el sistema de rutas.

4.2.2. Estilos CSS

La mayoría de los estilos empleados surgen de las clases que nos proporciona Bootstrap. Para las restantes declaraciones de estilos efectuadas, se han obviado la utilización de los archivos css que se producen en la creación de cada componente, de manera que para personalizar parámetros de estilos utilizaremos el archivo general de estilos de la aplicación de Angular (`styles.css`), ubicado en la carpeta raíz del proyecto.

4.2.3. Rutas

El enrutamiento en Angular puede producirse de dos formas:

- A partir de los selectores definidos en el archivo component.ts del componente.

- A partir de un archivo de rutas cuyas características ofrecen alcanzar un grado de configuración avanzada en los enrutamientos establecidos.

Los selectores no son más que etiquetas de tipo HTML que se encuentran contempladas en el decorador de cada componente (**`@Component`**). Para la utilización de estos selectores modificables, deben colocarse en el template del componente principal de la aplicación (`app.component.html`), de este modo, todo el contenido que se introduzca en el template de cada componente, aparecerá reflejado como parte de la vista general de la aplicación.

Existen procedimientos de enrutamiento que no pueden ser ejecutados a través de selectores, como es el caso de la declaración y asignación de las rutas hijas. Para establecer rutas hijas de una ruta padre, es necesario hacer la declaración en un archivo que, por convención de buenas prácticas de utilización de Angular, suele llamarse `app.routes.ts` y situado en la carpeta raíz del proyecto.

Cabe mencionar, que cuando se utiliza el archivo app.routes.ts para establecer las distintas rutas que conectarán las distintas partes de la aplicación, es necesario la colocación de un selector específico en lugar del selector que posee cada componente como hemos explicado anteriormente. Este selector tiene el nombre de `<router-outlet> </router-outlet>`.

```
const APP_ROUTES: Routes = [

    { path: 'main',
      component: MainComponent,
      children: [
        { path: 'home', component: HomeComponent },
        { path: 'reglas', component: ReglasComponent },
        { path: 'clasificacion', component: ClasificacionComponent },
        { path: 'actuaciones', component: ActuacionesComponent },
        { path: 'noticias', component: NoticiasComponent },
        { path: '**', pathMatch: 'full', redirectTo: '/main/home'}
      ]
    },
    {
      path: 'dashuser/:uid',
      component: DashuserComponent,
      children: [
        { path: 'score', component: ScoreComponent },
        { path: 'diary', component: DiaryComponent },
      ]
    },
     { path: 'login', component: LoginComponent },
     { path: 'sign-up', component: SignUpComponent },
     { path: '**', pathMatch: 'full', redirectTo: '/main/home'}

];

export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES);
```

Figura 31. Archivo de rutas `app.routes.ts`

Como muestra la imagen, se establecen rutas hijas para el grupo de páginas que pertenecen a los componente de la “parte pública” de la aplicación (a partir de una página principal llamada ‘`main`’), y para el grupo de páginas vinculadas a la “parte privada” que requiere autenticación por parte del usuario.

4.2.4. Formularios

Aunque los formularios tienen un comportamiento híbrido que engloba parte de las dos capas de desarrollo, su función principal pertenece al front-end, gestionando la entrada de datos por parte del usuario a través de su interfaz.
No obstante, colinda con la capa de procesamiento de datos (*back-end*) debido a que, a través de sus componentes, envía, borra o actualiza información en la base de datos (*CRUD*).
Otro aspecto importante que llevamos a cabo en los formularios es la validación de los datos introducidos en sus campos, con el fin de asegurar que la información que se introduce en la base de datos es la esperada, proyectando así un mantenimiento eficaz en el futuro de la aplicación.
La estructura de los formularios está orientada por aproximación de *template*, aspecto que simboliza que la mayor parte de la programación de sus componentes se realiza en el código HTML.
En Angular, la mecánica de funcionamiento de un formulario se basa en un objeto `ngForm` que engloba todas las propiedades, valores y características que posee. Entre esas propiedades, hay algunas que se adjuntan como clases a los inputs del formulario, pudiendo ser utilizadas para establecer estilos visuales a las validaciones entre otras posibles acciones.
Un aspecto importante a destacar, es que cada componente del formulario (`input`), se comunica con la base de datos a partir de directivas `[(ngModel)]` controladas por el framework. De esta forma, logra una sincronización adecuada entre los componentes del formulario y los campos de los documentos de la base de datos.

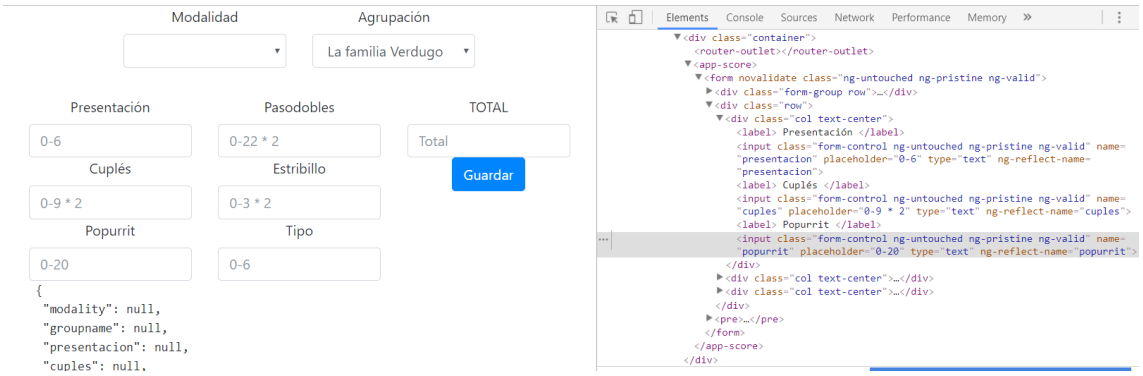

Figura 32. Vista y código fuente del formulario

4.3. Back-End

Al principio del apartado anterior, distinguimos los dos entornos de desarrollo que engloban a las aplicaciones web, aunque en realidad también pueden utilizarse en referencia a todo tipo de software. Esa distinción define al *back-end* de la aplicación como la encargada de gestionar toda la parte lógica de la misma, desde la conexión y transmisión de datos a través de bases de datos, hasta el tratamiento de procesos internos relacionado con el manejo de elementos dinámicos y control de usuarios.

En nuestro caso, el *back-end* de nuestra aplicación está centrado en dos componentes o procesos elementales para el funcionamiento de la aplicación que se asientan en herramientas pertenecientes a *Firebase*. Estos procesos son el *CRUD* y la autenticación de usuarios.

4.3.1. *CRUD*

Este acrónimo muy utilizado en el mundo de la ingeniería del software surge de las acciones principales que se utilizan en las peticiones HTTP que se realizan al servidor. Sus siglas significan - ***Create Read Update Delete*** -, para referir-

se a las acciones de crear, leer, actualizar y borrar elementos o registros de la base de datos respectivamente.
La mecánica del sistema de puntuación de nuestra aplicación consiste en:

- Calificar a una agrupación por parte de un usuario (inserción o creación de documentos).

- Mostrar todas las puntuaciones que ha realizado el usuario (obtención de documentos).

Al hacer uso de la API de *Firebase*, concretamente la modalidad de bases de datos *Cloud Firestore*, cada tipo de acción vendrá acompañada de un verbo HTTP específico:

- Creación de documentos: se realiza a partir de peticiones HTTP de tipo *POST*. La inserción de contenido en la base de datos en forma de documentos se realiza con el método `add()`.

- Obtención de documentos aislados o listar documentos: se realiza a partir de peticiones HTTP de tipo *GET*. Al utilizar la librería *angularfire2*, la obtención de datos a la base de datos se realiza mediante la utilización de observables.

- Actualización de documentos: se realiza a partir de peticiones HTTP de tipo *PATCH*.

- Eliminación de documentos: a partir de peticiones HTTP de tipo DELETE. Esta eliminación en la codificación, se ejecuta con el método `delete()`.

Este tipo de peticiones (actualización y eliminación) no se han efectuado debido a la naturaleza de los requerimientos iniciales del proyecto, en la cual señala que

un usuario, una vez ha introducido su puntuación, no podrá modificarla posteriormente.

Desarrollo de las operaciones CRUD

Para llevar a cabo la implementación de código para efectuar operaciones con la base de datos, requiere de la participación de dos elementos primordiales en la infraestructura de Angular: el componente y el servicio .

Servicio (score.service.ts)

Como se ha comentado anteriormente, el servicio debe contener toda la lógica encargada de realizar estos procesos. En primer lugar, importamos los módulos necesarios, entre los cuales tenemos los módulos necesarios para trabajar con las bases de datos a través de la librería *angularfire2*, módulos de enrutamiento, módulos para la autenticación de usuarios (utilizamos el mismo servicio para ambos procesos, de ahí que se encuentren importados) y módulos esenciales para el funcionamiento de la inyección de dependencias que realizan los servicios.

Continuamos con la declaración de variables, para ello, hemos creado una colección de elementos de tipo Score (definido previamente mediante una interfaz de tipo de datos - `score.interface.ts`), un array de puntuaciones (*scores*) para almacenar las puntuaciones que realizan los usuarios, una variable score de tipo Score para referirse a una puntuación en concreto (documento).

Es hora de realizar las funciones para guardar las puntuaciones realizadas en la base de datos (`saveScore()`) y para cargar esas puntuaciones en una tabla (`loadScore()`) perteneciente al apartado "Diario" del dashboard del usuario.

- saveScore (params....)

Esta función recibe todos los parámetros de una puntuación a una agrupación. La obligatoriedad de esto, es debido al establecimiento del tipo de datos que se realiza en una interfaz (*ver código de la interfaz* `score.interface.ts`)

Acto seguido, creamos un objeto de tipo *Score* que recibe todos y cada uno de los parámetros pasados en la entrada de la función, junto con la adición de la fecha en la que se efectua el almacenamiento de la puntuación, y el ***UID*** que es un identificador único que identifica a cada usuario (este punto lo hablaremos con más detalles en el apartado de autenticación).

Con la última instrucción de la función añadimos (`add`), mediante el método POST, los parámetros de la puntuación que hemos recibido en la entrada del usuario a través del formulario. Cabe destacar, que se realiza acciones de control de gestión de errores mediante las sentencias `then` y `catch`.

- loadScore()

Los observables son un tipo de patrón dentro de la ingeniería de software, cuya función es la de estar alerta a todos los cambios que se producen en la variable y notificarlos. En este caso, colocamos el observable (valueChanges()) en nuestra colección de puntuaciones que tenemos en la base de datos, cuya asignación se realizó previamente. Cuando se produzca la inserción de una nueva puntuación, el observable detectará que existe un registro más en la base de datos y, mediante el método map(), obtenemos la información que nos brinda el observable, la modificamos (en nuestro caso, introducimos las puntuaciones en un array) y lo devolvemos para que la información pueda ser procesada por otra función (en nuestro caso, en el constructor del componente score.component.ts).

Nota: Aunque es cierto que las instrucciones que aparecen en el constructor del servicio está practicamente dedicado a procesos de autenticación de usuarios, es muy impor-

tante la inyección de los módulos como parámetros de entrada. Sin ellos, no es posible realizar ninguna operación mencionada anteriormente.

```typescript
import { Injectable } from '@angular/core';
import { AngularFirestore, AngularFirestoreCollection } from 'angularfire2/firestore';
import { Score } from '../interfaces/score.interface';
import { AngularFireAuth } from 'angularfire2/auth';
import * as firebase from 'firebase/app';
import { Router } from '@angular/router';

@Injectable()
export class ScoreService {

    private itemsCollection: AngularFirestoreCollection<Score>;
    public scores: Score[];   // array para las puntuaciones de la BD
    public score: Score;
    public user: any = {};


  constructor( private afs: AngularFirestore, public afAuth: AngularFireAuth, private router: Router ) {

    this.afAuth.authState.subscribe( u => {
        console.log ('Estado del usuario: ', u);

        if (!u) { return; }
        this.user.name = u.displayName;
        this.user.uid = u.uid;
    });
  }


  loadScores() {
    this.itemsCollection = this.afs.collection<Score>('scores', ref => ref.orderBy('modality', 'asc'));

      return this.itemsCollection.valueChanges().map( (s: Score[]) => {
                                        console.log( s );
                                        this.scores = s;  // llenamos el array con las puntuaciones que haya en la BD
                                        });
  }


  saveScores( groupname: string, modality: string, presentacion: number, pasodobles: number,
              cuples: number, estribillo: number, popurrit: number, tipo: number) {

        // TODO Falta introducir el UID del usuario
        const score: Score = {
          groupname: groupname,
          modality: modality,
          presentacion: presentacion,
          pasodobles: pasodobles,
          cuples: cuples,
          estribillo: estribillo,
          popurrit: popurrit,
          tipo: tipo,
          date: new Date().getTime(),
          uid: this.user.uid
        };

        this.itemsCollection.add(score).then( () => console.log('Puntuación guardada'))
          .catch((err) => console.error('Error al enviar', err));
      }
}
```

Figura 33. Servicio: score.service.ts (CRUD)

Componente (`score.component.ts`)

La mecánica que rige la arquitectura de *Angular* continua con el procesamiento de los datos, tratado anteriormente en el servicio, por el componente. Todo componente se encuentra ligado a un template (`score.component.html`) para poder dar forma a la vista cuya interfaz será escenario de las distintas acciones que desempeñe el usuario.

De la misma forma que en el servicio, comienza con la importación de módulos requerido para las operaciones que realizará el componente. A continuación, se procede a la declaración de tres variables aunque, en este caso, las variables `itemsCollection` y `groups` son utilizadas para listar las agrupaciones en el `select` del formulario destinado a este proposito. Además de las mencionadas, existe un objeto `score` de tipo *Score* y con sus parámetros inicializados a `null`. Esto es debido por el mismo requisito explicado anteriormente, y es que, al ser de un tipo que está definido mediante una interfaz, la declaración e inicialización de todos sus parámetros es obligatoria.
Podemos observar que el componente está utilizando métodos definidos en el servicio, para ello es requerida la inyección del servicio en el constructor (`public _ss: ScoreService`), bajo previa importanción del servicio al principio del código.

- _ss.loadScore()

En este constructor, el único proceso que realizamos es la suscripción a la función `loadScore()` del servicio (`this._ss.loadScores().subscribe();`), ya que esta función hace uso de un observable en el servicio (`this.itemsCollection.valueChanges()`). El motivo de que se realice en el constructor es por su característica intrínseca de que todo el código dentro del mismo ocurre una vez se hayan cargado los elementos de la página.

- **saveScore()**

Esta función, simplemente, comprueba si existen valores validos introducidos en los diferentes inputs y, en caso afirmativo, asigna esos valores a cada uno de los parámetros que conforman cada puntuación.
Cabe destacar que el nombre de esta función se encuentra en singular y se refiere a la llamada por el (`ngSubmit`) del formulario, y no a la del servicio.

```
import { Component, ViewEncapsulation } from '@angular/core';
import { AngularFirestore, AngularFirestoreCollection} from 'angularfire2/firestore';
import { Observable } from 'rxjs/Observable';
import { ScoreService } from '../../services/score.service';
import { Score } from '../../interfaces/score.interface';


@Component({
  selector: 'app-score',
  templateUrl: './score.component.html',
  styles: [],
  encapsulation: ViewEncapsulation.None
})
export class ScoreComponent  {

  private itemsCollection: AngularFirestoreCollection<any>;

  groups: Observable<any[]>;

  score: Score = {
    groupname: null,
    modality: null,
    presentacion: null,
    pasodobles: null,
    cuples: null,
    estribillo: null,
    popurrit: null,
    tipo: null,
    date: new Date().getTime()
  };

    constructor( private afs: AngularFirestore, public _ss: ScoreService ) {
      this.itemsCollection = afs.collection<any>('groups');
      this.groups = this.itemsCollection.valueChanges();
      this._ss.loadScores().subscribe();
  }


  saveScore() {
    if (!this.score) {
      return;
    }

    this._ss.saveScores( this.score.groupname, this.score.modality, this.score.presentacion, this.score.pasodobles,
    this.score.cuples, this.score.estribillo, this.score.popurrit, this.score.tipo );

  }

}
```

Figura 34. Componente: score.component.ts

4.3.2. Autenticación de usuarios

Firebase cuenta con un *SDK* de autenticación ideal para su integración con las bases de datos dentro de cada proyecto creado con esta API.
El *SDK* ofrece una interfaz sencilla para su utilización, ofreciendo una serie de pestañas con una función específica en cada una de ellas. Estas funciones son:

- **Gestión de usuarios**: en esta sección podemos observar los usuarios que han sido añadidos a nuestra aplicación. Ofrece información sobre el método de acceso que ha utilizado el usuario, la fecha de su registro, la fecha del último inicio de sesión y por último el *UID* del usuario que, como hemos explicado anteriormente, se trata de un identificador único con el que trabajaremos a la hora de abrir una sesión individual. Cabe la posibilidad de añadir usuarios mediante esta plataforma, pero nuestra intención es registrar a los usuarios a partir de un sistema de *login / registro* en la aplicación.

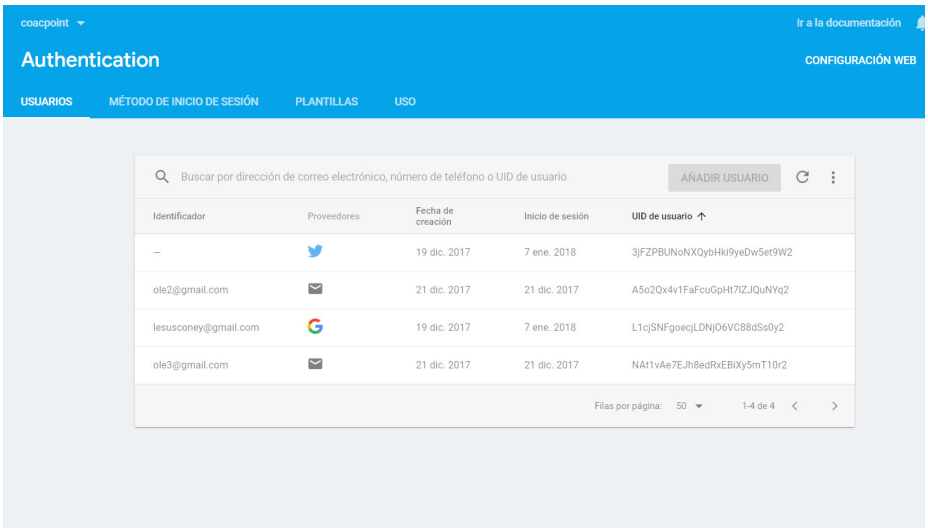

Figura 35. Firebase Authentication: Gestión de usuarios

- **Métodos de inicio de sesión**: muestra todas las variantes posibles de registro que podemos añadir a la aplicación. En nuestro caso, se permitirá registrarse a los usuarios mediante cuentas de *Google* y de *Twitter*.

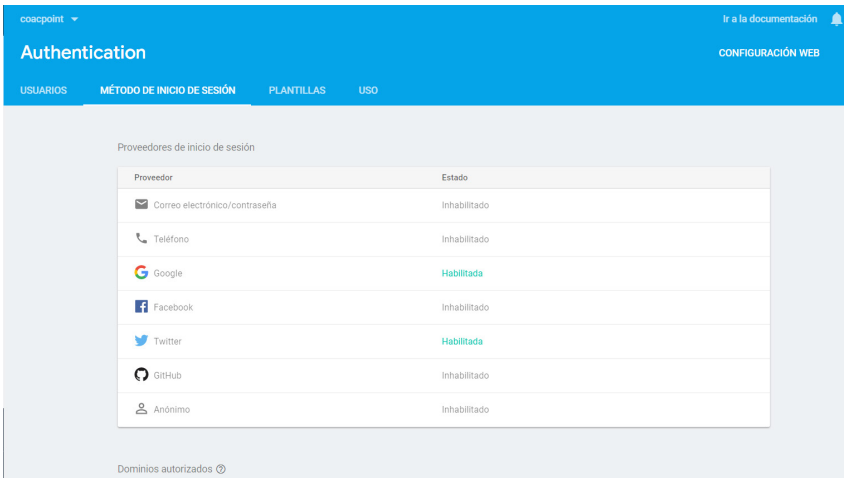

Figura 36. Firebase Authentication: Métodos de inicio de sesión

- **Plantillas**: para comunicación con los usuarios a través de correo electrónico o de SMS. Contiene plantillas para verificación de correo electrónico, cambio de contraseña, cambio de correo electrónico e iniciar sesión a partir de una contraseña de un solo uso.

- **Uso**: muestra mediante gráficos lineales la asiduidad en la que se produce las verificaciones telefónicas a través de SMS a teléfonos móviles.

Como hemos mencionado, la elección para el inicio de sesión será a través de cuentas de *Google* y *Twitter*, por lo tanto, se activan las dos opciones respectivas. Existe una diferencia respecto a la habilitación del servicio por parte de *Google* y *Twitter*. Como *Firebase* forma parte de *Google*, la habilitación del inicio por *Google* es más sencilla y no requiere de tokens de autenticación como es el caso de Twitter. Estos tokens se obtienen al registrar la aplicación en la red social (*Twitter Apps*).

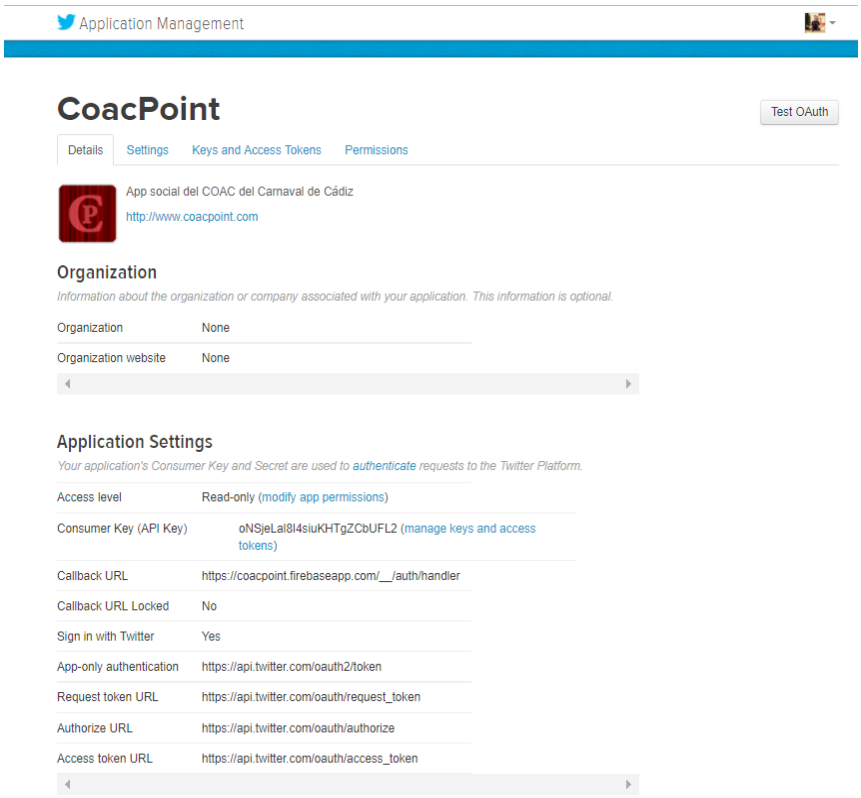

Figura 37. Configuración nicio de sesión con Twitter (Twitter Apps)

Desarrollo de la autenticación de usuarios

Una vez hemos habilitado las dos modalidades de sesión, pasamos a codificarlo en la aplicación haciendo uso de la librería *angularfire2*.

Servicio (score.service.ts)

La importación de módulo requerida para llevar a cabo tareas de autenticación de usuarios es

```
import { AngularFireAuth } from 'angularfire2/auth';
```

En la declaración de variables, declaramos un objeto vacío llamado user de tipo any que contendrá los aspectos que queramos que se muestren en pantalla.
En el constructor, además de la inyección de los módulos necesarios (incluido el `AngularFireAuth`), introducimos una instrucción que despliega un observable

al cual nos suscribimos. Esta instrucción nos permite saber en que estado se encuentra el proceso de autenticación del usuario. En caso de que no exista ningún usuario registrado, el estado será `null` y, en caso contrario mostrará el estado del usuario mostrando todos los parámetros asociados a él como el *displayName*, fotografía, *UID*, etc... (*ver imagen*)

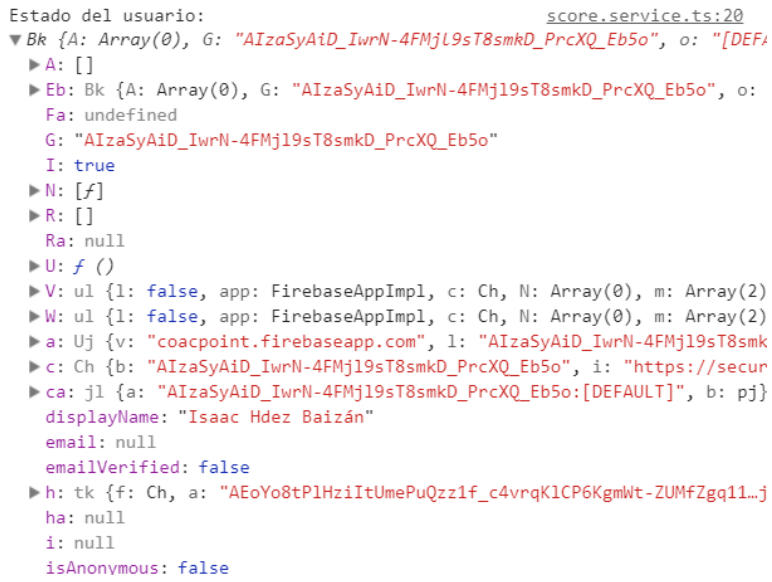

Figura 38. Estado del usuario reflejado en la consola del navegador

En nuestro caso, introducimos en nuestro objeto `user { }` declarado anteriormente, los parámetros *displayName* y *UID* para luego mostrarlo en el dashboard del usuario.
Para llevar a cabo la autenticación utilizando este tipo de cuentas, sólo nos hará falta crear una función de *login* y otra de *logout* para luego poder usarla en los componentes y templates que nos hagan falta.

- **`login (provider: string)`**

En el template del componente de login (*login.component.html*), creamos dos botones. Cada uno de estos botones posee un evento click, controlado por Angular, que ejecuta la función `loginUser('google')` o `loginUser('twitter')` según que inicio elija el usuario.

Esta función se encuentra en el componente (*login.component.ts*), y al recibir uno de los dos proveedores definidos (*Google* o *Twitter*). La única instrucción que realiza es asignar el nombre del proveedor a la función login del servicio para, posteriormente, ejecutar el inicio con el proveedor elegido mediante métodos de inicio del módulo *Auth* de *angularfire2*.
Una vez se ha realizado el registro y/o login del usuario, este será enviado a la página principal de su dashboard: la página de puntuación.

```
this.router.navigate([ '/dashuser', this.user.uid, 'score' ]);
```

- logout ()

Con esta función cerramos la sesión del usuario y borrando el objeto *user*, de esta manera no se producirán conflictos cuando otros usuarios intenten iniciar sesión. Como en el caso de `login()`, hacemos uso de *router.navigate*, pero en este caso, dirigimos al usuario a la pantalla de inicio de la aplicación.

```
1  import { Component, ViewEncapsulation } from '@angular/core';
2  import { ScoreService } from '../../services/score.service';
3
4
5  @Component({
6    selector: 'app-login',
7    templateUrl: './login.component.html',
8    encapsulation: ViewEncapsulation.None
9  })
10 export class LoginComponent {
11
12   constructor( public _ss: ScoreService ) { }
13
14
15   loginUser(provider: string) {
16     console.log( provider );
17     this._ss.login( provider );
18
19
20   }
21 }
22
```

Figura 39. Componente: login.component.ts

```typescript
import { Injectable } from '@angular/core';
import { AngularFirestore, AngularFirestoreCollection } from 'angularfire2/firestore';
import { Score } from '../interfaces/score.interface';
import { AngularFireAuth } from 'angularfire2/auth';
import * as firebase from 'firebase/app';
import { Router } from '@angular/router';

@Injectable()
export class ScoreService {

    private itemsCollection: AngularFirestoreCollection<Score>;
    public scores: Score[];   // array para las puntuaciones de la BD
    public score: Score;
    public user: any = {};


  constructor( private afs: AngularFirestore, public afAuth: AngularFireAuth, private router: Router ) {

    this.afAuth.authState.subscribe( u => {
        console.log ('Estado del usuario: ', u);

        if (!u) { return; }
        this.user.name = u.displayName;
        this.user.uid = u.uid;
    });
  }


  // Autenticación de usuarios

  login( provider: string ) {

    if (provider === 'google') {
    this.afAuth.auth.signInWithPopup(new firebase.auth.GoogleAuthProvider());
    } else {
      this.afAuth.auth.signInWithPopup(new firebase.auth.TwitterAuthProvider());
    }

    this.router.navigate([ '/dashuser', this.user.uid, 'score' ]);
  }

  logout() {
    this.user = {};
    this.afAuth.auth.signOut();
    this.router.navigate(['/']);
  }
```

Figura 40. Servicio: score.service.ts (Auth)

4.4. Validaciones de casos de usos

Resulta primordial que deba haber un mecanismo de control que permita validar las operaciones principales para poder cumplir los objetivos primarios del proyecto. Un reanálisis de los casos de usos puede proporcionarnos una herramienta excelente para determinar si existe alguna anomalía al ejecutarse cada caso de uso, al igual que nos ayuda a reflejar requisitos funcionales y no funcionales partiendo de la misma declaración.
Como apunte, la tabla también refleja validaciones respecto a los casos de uso para la figura del administrador.

VALIDACIONES DE CASOS DE USOS - Requisitos funcionales y no funcionales	
El usuario puede puntuar a una agrupación	✓
El usuario puede compartir la puntuación a través de redes sociales.	✓
El usuario debe autenticarse para acceder a su sesión personal.	✓
El usuario puede consultar información general sin necesidad de registrarse o utilizar sus datos personales.	✓
El administrador puede gestionar a los usuarios registrados en la aplicación y a las agrupaciones del concurso.	✓

Figura 41. Valicaciones de casos de usos

5. Conclusiones y líneas de futuro

5.1. Conclusiones

Teniendo en cuenta, que el aprendizaje de los lenguajes de programación utilizados han sido prácticamente desde cero, la elaboración del proyecto ha enriquecido, principalmente y de manera profunda, mis conocimientos sobre el entorno de desarrollo front-end, en especial sobre la librería Angular y la API REST de Firebase.

El núcleo de la investigación ha estado centrado en la teoría de la ingeniería de software, desde el estudio del lenguaje unificado de modelado (UML) hasta el análisis y diseño de patrones que se emplean para la creación del software. Los conceptos generales dieron paso al estudio, de forma paralela, de la arquitectura del framework, con todos sus componentes y directivas, y la infraestructura técnica y servicios de los que disponía Firebase en relación con Angular.

En principio, se pensó en utilizar bases de datos relacionales, ya que habían sido estudiadas durante el grado, y aportaban una mayor facilidad a la hora de su diseño e implementación, pero la materia no estaba clara para llevarla a cabo, y eso que aún no sabía que había cometido un gran error midiendo el alcance del proyecto. Ese fue el motivo principal de mi elección por Firebase.

La planificación comenzó siendo muy milimetrada, coincidian los tiempos entre tareas, y tenía claro lo que quería conseguir y cual era mi objetivo personal principal.

En la parte de diseño web, incluida la identidad gráfica, elaboración de elementos visuales, logotipos, tipografías y demás, no había problema teniendo las ideas bastante claras y además, teniendo un conocimiento avanzado en el manejo de herramientas gráficas como Illustrator o Photoshop.

El gran problema fue la enorme y costosa curva de aprendizaje que sostiene la adquisición de conocimientos sobre Angular. Pero no sólo es Angular, ya que también debes aprender TypeScript, añadiendose la ardua tarea de estudiar la forma de utilizar Firebase con Angular, y no con JavaScript como documentan

ampliamente en su sitio web.
Otro aspecto que no ayudó fue las actualizaciones completas de versión que tuvo Angular en un periodo de tiempo muy corto, en cuestion de pocos meses. Este hecho provoca que la información que existe en la red esté muy poco contrastada, y los conocimientos que aprendes para llevar a cabo una tarea de programación, te das cuenta de que ya no valen para lo siguiente que quieres hacer, y llega a ser un cúmulo de despropósitos enorme.
Se puede observar en la aplicación que quedan aspectos por pulir, como la integración de contenido estático en las páginas no restringidas y algunas reglas de estilos, sobretodo en la parte restringida. Cabe mencionar, que ha habido varias reconstrucciones, casi completas, de la aplicación, teniendo que volver a diseñar los wireframes y prototipos, reestructurar la arquitectura de la información e incluso dejar de utilizar librerías y herramientas debido a la dificultad para implementar la aplicación con Angular.

Todo este asunto, surgido en el proyecto, me ha hecho reflexionar sobre si las fases del desarrollo web están en un orden definido correcto, porque si una vez has realizado el estudio del diseño, la información a mostrar, los prototipos visuales, los diagramas UML, el diseño de la base de datos , y luego en la implementación de código no lo puedes llevar a cabo, me pregunto...¿estarán las bases teóricas sobre las fases que debe atravesar la producción de software bien fundamentadas? o ¿quizás deba de tener otro orden prioritario según el desarrollador? o ¿podría ser todo producido por un mal enfoque?

5.2. Líneas de futuro

La aplicación, en un primer lugar, estaba diseñada para cumplir ciertas funciones específicas con un componente más enriquecedor para el usuario en referencia a la experiencia en su interacción con las diferentes interfaces propuestas en la

aplicación. A continuación, detallamos las posibles líneas de desarrollo que se podrían llevar a cabo sobre la implementación de nuevos componentes y funciones:

- **Componente social interno**: la elaboración de un espacio de carácter social, en la que el usuario pueda realizar acciones típicas como establecer un sistema de seguimiento entre usuarios, permitir la compartición de archivos (puntuaciones) en un espacio dentro de la aplicación sin eliminar la posibilidad de transmitir contenido a redes sociales externas. Además, podría albergar un sistema de comentarios con el que pudieran comunicarse, sin necesidad de salir de la aplicación.

- **Sistema de gamificación**: el fundamento de la aplicación es un juego, y como tal deben existir premios, trofeos o algo similar. La participación de los usuarios, al usar los componentes sociales, también podría ser aumentada con la inclusión de características similares que puedan fomentar el uso de la aplicación. Podría confeccionarse un diseño de insignias (badges) en diferentes ámbitos: uno en el ámbito de las interacciones sociales entre los usuarios, y otro en referencia a la clasificación que produce el juego en sí.

Otras ideas que pueden producir nuevas vías de desarrollos podrían ser:

- Registro como agrupación con perfiles de información para la contratación del grupo en cualquier evento y espacio propio para realizar comunicados a los aficionados.

- Agenda de eventos relacionados con la agrupaciones de carnaval.

- Crear un apartado mostrando vídeos sobre las actuaciones, de manera que el usuario no se tenga que trasladar a otra página para ver los vídeos preferidos de sus agrupaciones.

Bibliografía y enlaces

Bibliografía

- **Monjo Palau, Tona**. *Diseño centrado en el usuario*. (Módulo 2. Asignatura: Diseño de interfaces multimedia). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Monjo Palau, Tona**. *Usabilidad*. (Módulo 3. Asignatura: Diseño de interfaces multimedia). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Monjo Palau, Tona**. *Diseño*. (Módulo 4. Asignatura: Diseño de interfaces multimedia). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Monjo Palau, Tona**. *Accesibilidad*. (Módulo 5. Asignatura: Diseño de interfaces multimedia). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Pradel i Miquel, Jordi; Raya Martos, José**. *Análisis UM*L. (Módulo 4. Asignatura: Ingeniería del software). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Pradel i Miquel, Jordi; Raya Martos, José**. *Requisitos*. (Módulo 3. Asignatura: Ingeniería del software). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Patricia Gil, Eva; de Lera Tatjer, Eva; Monjo Palau, Antónia**. *Usuarios y sistemas interactivos* (Módulo anexo. Asignatura: Arquitectura de la Información). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Morville, Peter; Rosenfeld, Louis (2006)**. *Arquitectura de la información para la World Wide Web*. California (USA): O'Reilly Media Inc.

- **Costal Costa, Dolors**. *Introducción al diseño de bases de datos* (Módulo anexo. Asignatura: Uso de Bases de Datos). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Pradel i Miquel, Jordi; Raya Martos, José**. *Catálogo de patrones*. (Módulo 2. Asignatura: Análisis y diseño de patrones). Universitat Oberta de Catalunya. Barcelona: FUOC.

- **Pradel i Miquel, Jordi; Raya Martos, José**. *Introducción a la ingeniería de software*. (Módulo 1. Asignatura: Ingeniería del software). Universitat Oberta de Catalunya. Barcelona: FUOC.

Enlaces

- **Google Play** (2017). *Plataforma de distribución digital de aplicaciones para sistemas Android*. [Consulta: Sept - Oct 2017] < https://play.google.com/store >

- **NSU** (2016). *Metodologías y técnicas de DCU*.[Consulta: Oct 2017]
< http://www.nosolousabilidad.com/manual/3_2.htm >

- **Google** (2017). *Angular Official Website*.[Consulta: Sept 2017 - Enero 2018]
< https://angular.io/ >

- **Wikipedia** (2017). *Transferencia de Estado Representacional*.[Consulta Oct 2017]
< https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional >

- **Armentano, Lucila** (2017). *Buenas prácticas para el diseño de una API REST ful Pragmática*. [Consulta: Oct - Nov 2017]
< https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-rest-ful-pragmatica/# >

- **Marqués, Asier**. *Conceptos sobre APIs REST*. [Consulta: Oct-Nov 2017]
< http://asiermarques.com/2013/conceptos-sobre-apis-rest/ >

- **Oriol, Enrique** (2016). *Introducción a Angular 2 (parte I) - Módulo, Componente, Template y Metadatos*. [Consulta: Nov 2017]
< http://blog.enriqueoriol.com/2016/06/introduccion-a-angular-2-parte-i-componente.html >

- Microsoft (2017). *TypeScript Language Website*. [Consulta: Nov-Dic 2017]
< https://www.typescriptlang.org/ >

- **Google** (2017). *Firebase Official Website*. [Consulta: Nov 2017- Ene 2018
< https://firebase.google.com/ >

- **Google** (2017). *Google Fonts Website*. [Consulta: Nov 2017]
< https://fonts.google.com/ >