UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Tecnologías de la Información

TRABAJO FIN DE GRADO

Mejora y refactorización de componentes para la celebración de concursos personalizados

Miguel Ángel Rodríguez Fernández de Simón

febrero, 2023

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

DEPARTAMENTO DE TECNOLOGÍAS Y SISTEMAS DE INFORMACIÓN
(cont.)

Tecnologías de la Información

TRABAJO FIN DE GRADO

Mejora y refactorización de componentes para la celebración de concursos personalizados

Autor: Miguel Ángel Rodríguez Fernández de Simón

Tutor: Carlos González Morcillo

Co-tutor: Francisco Manuel García Sánchez-Belmonte

febrero, 2023

Mejora y refactorización de componentes para la celebración de concursos personalizados

Este texto ha sido preparado con la plantilla LaTeX para Trabajo Fin de Estudios en Ingeniería Informática para la UCLM publicada por Jesús Salido en el repositorio público Zenodo, DOI: 10.5281/zenodo.4561708, como parte del curso *«LaTeX esencial para preparación de TFG, Tesis y otros documentos académicos»* impartido en la Escuela Superior de Informática de la Universidad de Castilla-La Mancha [19].

Tribunal:

Presidente: _____

Vocal: _____

Secretario(a): _____

Fecha de defensa: _____

Calificación: _____

Presidente	Vocal	Secretario(a)
Fdo.:	Fdo.:	Fdo.:

Al chico de 18 años
Que no sabía si atreverse a cambiar de carrera

TRABAJO FIN DE GRADO - ESCUELA SUP. DE INFORMÁTICA (UCLM)

Mejora y refactorización de componentes para la celebración de concursos personalizados

Miguel Ángel Rodríguez Fernández de Simón
Ciudad Real, febrero 2023

Resumen

El objetivo de este trabajo es la mejora y refactorización de algunos de los sistemas usados por la empresa Furious Koalas para la creación y celebración de concursos personalizados. Concretamente, este trabajo aborda tres componentes bien diferenciados:

- Sistema de visualización de las preguntas, junto a las respuestas, archivos multimedia asociados, comodines y rankings.
- Sistema de programación y control de focos, máquina de humo y demás dispositivos DMX.
- Configuración de un dispositivo ATEM para la producción de contenido de vídeo en directo, su control a través de un Stream Deck, y la redacción de un breve manual de uso para futuros proyectos.

Para ello se han utilizado lenguajes como Python, Javascript o CSS, entre otros, así como el framework Flask, tecnologías y aplicaciones de terceros como Chart.js, PyArtNet o Bitfocus Companion, y dispositivos como un ATEM de Blackmagic, una tarjeta Raspberry Pi o un controlador DMX, además de varios focos.

Tras el desarrollo, se ha fabricado un nuevo visualizador de preguntas que abandona Unity (el sistema usado hasta la fecha), se ha creado un servidor web encargado de aceptar, verificar y traducir órdenes en formato JSON a mensajes DMX y se han configurado tanto el ATEM como el Steamdeck, introduciendo numerosas macros y vistas, además de la creación de un manual que se ha añadido a la documentación interna del sistema.

BACHELOR DISSERTATION - ESCUELA SUP. DE INFORMÁTICA (UCLM)

Refactoring and upgrade of subsystems for custom quiz contest's celebration

Miguel Ángel Rodríguez Fernández de Simón
Ciudad Real, February 2023

Abstract

This dissertation's objective is the upgrade and refactoring of some of the systems used by the company Furious Koalas for the creation and celebration of custom quiz contests. Specifically, this dissertation tackles three very different components:

- *Question visualization system, along with the answers, associated multimedia files, wildcards and rankings..*
- *Spotlight, hazer and other DMX devices' programming and control system.*
- *Configuration of an ATEM device for live video production, control through a Stream Deck and writing of a brief use manual for future projects.*

For this purpose there have been used languages such as Python, Javascript or CSS among others; the Flask framework, thirdparty technologies and applications such as Chart.js, PyArtNet or Bitfocus Companion, and devices like an ATEM from Blackmagic, a Raspberry Pi card or a DMX controller, along with several spotlights.

After the development, a new question visualizer that abandons Unity (the system used until this date) and a web server in charge of accepting, verifying and translating JSON orders to DMX messages were created. Furthermore, the ATEM device and the Stream Deck were both configured, introducing various macros and views, in addition to the creation of a manual that was added to the inner documentation of the system.

Agradecimientos

Tengo tantas cosas que agradecer a tantas personas que daría para otras 40 páginas, pero intentaré ser breve.

A mis padres por haberme apoyado sin condiciones incluso cuando yo mismo no sabía lo que estaba haciendo, y a mis hermanos por la paciencia y por haberme aguantado en mis momentos de mayor estrés.

A Noe, por estar siempre en primera línea y ayudarme incluso más de lo que ella está dispuesta a reconocer, sacando energía y tiempo de donde no tenía para darme siempre su mejor versión.

A mis abuelos Emilio, Toni y Gloria por haberme animado a perseguir lo que de verdad quería y por transmitirme su paciencia y experiencia.

A mis tíos y primos por haberme dado una familia tan grande, completa y unida, y en especial a mi tía Gloria, que me guió y aconsejó cuando me di cuenta de que lo que estaba estudiando no me llenaba como debería.

A Jose, Fernando y Fran, porque sin ellos no habría vivido la mitad de cosas que viví, y porque como dice Jose, "ninguno de nosotros es tan bueno como todos nosotros juntos".

A Edu, Bru, Digi, Carlos y todos mis amigos, que siempre han estado ahi para hacerme saber que nunca he estado ni estaré solo.

A Julio, Adrián, Tamara, Javier, Alejandro y todos los demás amigos y compañeros que amenizaron la vida universitaria (y que también me salvaron en más de una asignatura).

A Clara, Fran, Guille, Montse y Nieves por acompañarme en lo que para mi fue un auténtico salto al vacío.

A tantos profesores que me guiaron, se preocuparon y se aseguraron de que acabara el curso como un auténtico ingeniero, más allá de mis conocimientos teóricos.

Y, por supuesto, a Carlos y a Paco, por acompañarme y ayudarme en este TFG, por la oportunidad y por la (infinita) paciencia que han demostrado conmigo.

Miguel Ángel Rodríguez Fernández de Simón
Ciudad Real, 2023

Índice general

Índice de figuras

Índice de listados

CAPÍTULO 1
Introducción

Furious Koalas Interactive se define a sí misma como "una empresa de base tecnológica creada en 2015 como spin-off de la Universidad de Castilla-La Mancha que crea y desarrolla productos y servicios basados en soluciones interactivas comprendiendo el diseño, desarrollo y explotación de proyectos de software"[1]. Entre los numerosos productos y servicios que se mencionan en su descripción se encuentra la celebración de concursos personalizados.

Para la realización de estos eventos se hace uso de un gran número de tecnologías y herramientas. Entre ellas, las tres en las que se centrará el trabajo serán el visualizador de las preguntas (que se verá en una gran pantalla durante el concurso), el sistema de control de los focos y la máquina de humo, y el sistema de producción de vídeo para la grabación o difusión en directo de los eventos.

Estas tres herramientas necesitan recibir una actualización debido a una poca fiabilidad o un proceso de preparación tedioso que puede provocar la necesidad de más tiempo previo a la celebración del concurso o la presencia de más errores.

Debido a esto, se acordó enfocar este TFG como una realización de prácticas durante tres meses, dedicando un mes a cada herramienta de manera que reciban un tratamiento independiente, de maner similar a "subproyectos". Sin embargo, pese a este enfoque "por separado", el objetivo general de los tres es el mismo: la refactorización de todos estos sistemas, realizando los cambios que sean necesarios en busca de una mejoría en rendimiento, fiabilidad y eficiencia. Además, debido a que se busca una mejora enfocada en estas partes, se debe tener especial cuidado de no realizar cambios que requieran de una actualización a gran escala en el resto de componentes, aumentando exponencialmente el tiempo necesario y la complejidad del proyecto.

1.1. ESTADO DEL ARTE

La gamificación (o ludificación) se define como "la aplicación de técnicas o dinámicas propias del juego a actividades o entornos no recreativos para potenciar la motivación y la participación, o facilitar el aprendizaje y la consecución de objetivos". Este concepto se ha hecho especialmente interesante con la aparición de las nuevas tecnologías, ampliando tanto los métodos que se pueden utilizar para ese efecto como los distintos ámbitos en los que es aplicable.

Un framework especialmente útil para comprender y enumerar estas técnicas es el framework *Octalysis*[5]. Creado tras numerosas investigaciones sobre gamificación y diseño del comportamiento, se ha creado un sistema con ocho "unidades principales", que clasifican las técnicas de gamificación. Estas unidades son:

- Significado épico y llamada.
- Desarrollo y logro.
- Empoderamiento de la creatividad y retroalimentación.
- Propiedad y posesión.

[1] https://www.furiouskoalas.com/quienes-somos/

- Influencia social y relación.
- Escasez e impaciencia.
- Impredictibilidad y curiosidad.
- Pérdida y evitación.

Además, estas unidades están ordenadas de manera concreta, ya que, como se puede comprobar en la figura 1.1, se distingue entre las técnicas de la izquierda, que tienden más a una tendencia extrínseca y las de la derecha, que favorecen una tendencia intrínseca. Además, las técnicas presentes en la zona superior del octágono se consideran más positivas, mientras que las de la zona inferior son técnicas de motivación negativa.

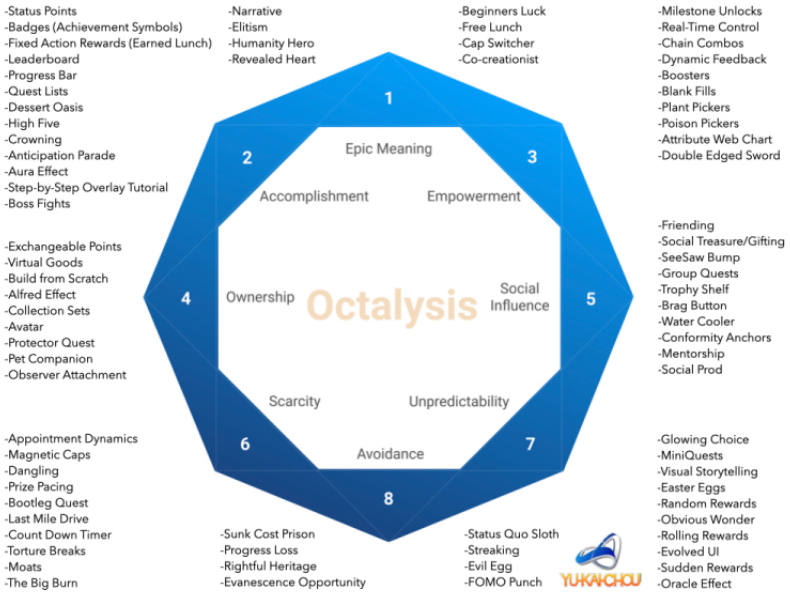


Figura 1.1: Framework Octalysis (Fuente: Yu-Kai Chou)

Otro concepto que puede beneficiarse mucho de estas técnicas es el marketing interno, que consiste en tener empleados satisfechos "tratándoles como clientes, aplicando los principios del marketing al diseño de puestos de trabajo y motivación del empleado"[1]. La gamificación, por tanto, es especialmente útil para este objetivo, y se han hecho numerosos estudios sobre las aplicaciones de dichas técnicas a un campo tan amplio como el del marketing[15] y el ámbito empresarial en general[17].

Aprovechando esta sinergia, una de las líneas de negocio de la empresa *Furious Koalas* es la de los concursos de preguntas en directo, aprovechando todas estas unidades principales, ya que como explica Yu-Kai Chou, "cuando no hay ninguna de estas unidades detrás de una acción deseada, hay *cero* motivación y no ocurrirá ninguna acción"[5].

Estas unidades se aplican a los concursos de preguntas en general de maneras diversas. Por ejemplo, responder las preguntas desde un atril podría provocar una motivación basada en la primera y segunda unidades (llamada y logro). Por otra parte, una motivación para acertar la pregunta puede basarse tanto en el logro al como en la evitación de una respuesta errónea. Como último ejemplo, la posibilidad de que el público participe y gane puntos en un ranking aprovecha unidades como el logro o la influencia social, además del "significado épico" de estar en un puesto alto.

Además, estas unidades influyen también en quienes no participan de manera directa en ellos. Las respuestas invitan a responderse y a jugar en compañía incluso cuando se está viendo, por ejemplo, desde la televisión. Es por razones como estas que programas como Pasapalabra continúan batiendo récords de audiencia[11] y los concursos de preguntas siguen siendo un éxito.

Los concursos que ofrece Furious Koalas, además, son personalizados para cada cliente y evento, por lo que las empresas que requieran de estos servicios podrán aplicarlos fácilmente a sus propios ámbitos y empleados, facilitando el marketing interno mencionado anteriormente.

Esto produce la necesidad de utilizar tecnologías y métodos de trabajo que permitan una alta personalización y adaptación, de manera que tanto las preguntas como los recursos multimedia, efectos de luz y sonido o incluso los propios componentes del escenario (como pueden ser los atriles) puedan cambiar rápidamente y responder a diseños cambiantes y muy diferentes.

Otro aspecto que influye en el sistema de los concursos es la producción en directo, que introduce toda una serie de trabajos independientes y requisitos. El primero (y más crítico) es la necesidad de un sistema muy robusto, ya que cualquier error puede afectar de forma muy negativa a la experiencia de los participantes. Esto no sólo aplica al propio concurso, sino que la producción de contenido multimedia debe ser también resistente a errores que dificulten o vuelvan imposible la difusión en directo y/o la grabación del evento.

1.1.1. Estructura del sistema

El sistema utilizado para la celebración de los concursos utiliza numerosas tecnologías (estándar MIDI, redes locales o incluso motores de videojuegos). En esta sección se tratarán alguno de los componentes más relevantes para el trabajo.

Todo el funcionamiento está basado en una red LAN conectada a través de cables Ethernet (a excepción de una tablet para el presentador, que se encuentra conectada por Wifi). En la figura 1.2 se pueden ver todos los componentes involucrados.

Los más interesantes para este trabajo son “Question Viewer”, “DMX Controller” y “Atem Studio Switcher”. El primero es el que proyecta las preguntas en la gran pantalla para que el público pueda verlas y, como se verá más adelante, está implementado en Unity. El segundo es el encargado de traducir los mensajes ArtNet a señales DMX (también se tratará en una sección posterior) y, por último, el tercero es el encargado de la producción de contenido en directo y también se comentará con más detalle más adelante.

Una gran ventaja de esta configuración es que la lógica del sistema está completamente separada de los dispositivos que vayan a usarse para controlarla o mostrarla (siempre y cuando estos dispositivos tengan conexión con la red). De esta manera, para controlar el concurso en general se puede acceder al servidor principal a través de cualquier ordenador o dispositivo móvil.

Un detalle muy importante a tener en cuenta es que esta red está completamente aislada del exterior. No tiene conectividad con Internet, y por lo tanto es vital que las aplicaciones no dependan de bibliotecas o servicios alojados en la nube.

1.1.1.1. Flask

Para la creación de los dos servidores web que abarca este proyecto se utilizará la biblioteca Flask para Python[2]. Esta biblioteca permite la creación de una API REST de manera muy sencilla, definiendo ciertas rutas para que un servidor pueda actuar de distinta manera para todas ellas.

Esto resuelve la problemática de seguir siendo compatible con el resto del sistema, ya que funciona de manera idéntica a los componentes anteriores. De hecho, el proyecto ya estaba en gran parte implementado en Flask, por lo que se aumenta aun más la coherencia y cohesión del sistema, eliminando plataformas innecesarias.

[2] https://flask.palletsprojects.com/en/2.2.x/

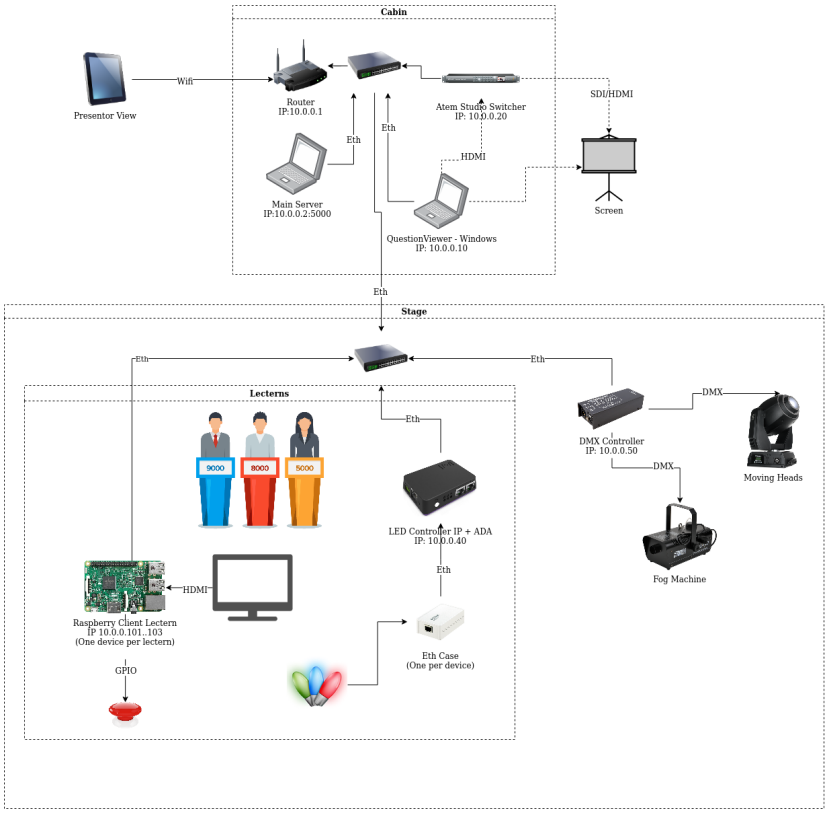


Figura 1.2: Esquema de la arquitectura del sistema previo a los cambios.

1.1.2. Unity

Unity[3] se trata de un motor de videojuegos creado en 2005 muy ampliamente utilizado y con presencia en grandes éxitos como Fall Guys, Among Us o Pokemon Go.

De estos ejemplos también se puede inferir otra de sus grandes ventajas, y es el hecho de tratarse de un motor multiplataforma, permitiendo la creación de aplicaciones tanto para consolas, ordenadores, móviles o navegadores. Este último ejemplo es el que nos atañe, ya que, como se ha dicho antes, el sistema del concurso está formado por componentes que se comunican a través de una red privada. Las pantallas que muestran las preguntas, la tablet del presentador, la pantalla de control o las de los atriles no son más que navegadores en pantalla completa, a través de los cuales se introducen y visualizan las distintas operaciones y sus resultados.

Es por eso que anteriormente el visualizador estaba hecho sobre WebGL, la plataforma web de Unity. Esto permitía un grado de interactividad que es vital en los concursos de este tipo, sobre todo si son en directo. Sin embargo, Unity en web es propenso a errores (por ejemplo, desaparición de manera aleatoria de ciertas letras en los títulos de las preguntas), además de contar con un entorno de desarrollo lento y pesado que hace más tediosa la realización de cambios en el código de la aplicación.

Por estas razones, Unity se ha abandonado en este trabajo a favor de otras soluciones más ligeras y de código abierto, y ha sido mencionado únicamente como reconocimiento al estado del sistema previo a este proyecto.

[3] https://unity.com/

1.1.3. DMX y ArtNet

El protocolo DMX[8] es utilizado en muchos eventos para el control de aparatos de iluminación, máquinas de humo y demás aparejos similares. Está basado en el uso de universos, que no son más que grupos de hasta 512 canales. Estos canales, a su vez, permiten la transmisión de valores entre 0 y 255, que los dispositivos reciben y aplican a un determinado parámetro.

Por ejemplo, una máquina de humo podría tener dos canales: el primero para la cantidad de humo deseada, y el segundo para la velocidad del ventilador. Este dispositivo puede configurarse para, una vez conectado a un universo DMX, comenzar en un determinado canal (no necesariamente el primero). Los dispositivos DMX se irán conectando de manera secuencial hasta completar el universo. En este ejemplo diremos, por plantear una situación común pero quizás menos intuitiva, que se ha establecido el inicio de los canales de la máquina de humo en el canal 20. De esta manera, los valores transmitidos por el canal 20 dictarán la cantidad de humo, mientras que los del canal 21 configurarán la velocidad del ventilador.

Además, el hecho de que la máquina de humo ocupe esos canales no provoca que los demás dispositivos no puedan acceder a ellos. Los canales siguen estando disponibles, pero no suele ser aconsejable solapar varios dispositivos en los mismos canales porque puede provocar un uso engorroso y antiintuitivo de los mismos (a no ser que, siguiendo el ejemplo, tengamos otra máquina de humo igual que queramos que se comporte de la misma manera en los mismos momentos).

Este protocolo se creó en 1986, pero no fue hasta 2004 que fue aprobado por ANSI[4]. No obstante, está comenzando a quedarse obsoleto debido a ciertos inconvenientes. Por un lado, con el avance de la tecnología y el aumento de complejidad de los aparatos, estos consumen cada vez más canales (por ejemplo, mientras que la máquina de humo consume 2 canales, los focos utilizados en este proyecto utilizan 16). Por otra parte, DMX no obtiene ningún tipo de retroalimentación sobre si los valores han sido recibidos o el estado de los aparatos. De hecho, es posible enviar mensajes a un canal sin que ningún dispositivo esté escuchándolo. Es por ello que ya se están creando otras soluciones como ACN, que está pensado para ser bidireccional, o RDM, que permite monitorear los aparatos a través de los propios cables DMX.

También es importante mencionar el protocolo ArtNet[5], que fue desarrollado en 1998, aunque la última versión (ArtNet 4) fue lanzada en septiembre de 2016. Este protocolo permite la transimisión de datos DMX (y RDM, aunque no se ha utilizado en este proyecto) a través de Ethernet. Para ello se basa en el protocolo UDP.

Está diseñado para poder gestionar numerosos universos DMX (lo cual proporciona una solución a la limitación mencionada anteriormente). Concretamente, ArtNet 4 es capaz de agrupar hasta 1000 puertos DMX en una única dirección IP.

1.1.4. FreeStyler

Este software[6] nació con la intención de realizar una tarea muy similar al servidor DMX de este proyecto, permitiendo controlar dispositivos DMX de manera gratuita, además de la creación de animaciones. Sin embargo, la aplicación es demasiado lenta y muy poco intuitiva, además de tener numerosos errores.

Debido a todas estas razones, su uso es frustrante y muy tedioso, y este es el motivo de que se desee crear una aplicación propia capaz de sustituirla.

[4] https://tsp.esta.org/tsp/documents/docs/ANSI-ESTA_E1-11_2008R2018.pdf
[5] https://art-net.org.uk/
[6] https://www.freestylerdmx.be/

1.1.5. Blackmagic

Esta empresa[7] se dedica a la fabricación de este tipo de dispositivos, y tiene bastante renombre gracias a su relación calidad-precio, llegando a colaborar con marcas como Netflix o Apple. Se ha mantenido el dispositivo, pero se ha prescindido de OBS[8], un programa de captura de pantalla que se utilizaba para la producción de contenido. El motivo de esta decisión es que este programa no ha resultado ser suficientemente fiable y se espera que, al eliminar un intermediario y usar tan sólo el dispositivo "nativo", puedan resolverse en gran parte estos inconvenientes.

Como se detallará más adelante, este aparato cuenta con diversas aplicaciones diseñadas por el propio proveedor que permiten configurar y aprovechar todas las funcionalidades. Además de estas, al ser una marca tan utilizada, tiene una gran comunidad de usuarios que, a través de foros tanto oficiales como externos discuten y desarrollan soluciones y programas para resolver los programas o los casos de uso que no contemple el aparato "de base".

1.1.6. Bitfocus Companion

Esta aplicación[9], de código abierto, se trata de un proyecto de código abierto orientado, precisamente, a actuar como intermediario entre diversos dispositivos de producción audiovisual (incluyendo el ATEM que se va a utilizar) y una botonore, o Stream Deck. Gracias a esta aplicación se pueden definir multitud de comportamientos y acciones para asociarlas a diversos botones, lo cual facilita gran parte del proceso de uso de estos dispositivos, sobre todo cuando se trata de macros.

1.2. ESTRUCTURA DEL DOCUMENTO

La memoria de este trabajo se ha organizado en distintos apartados como viene siendo habitual en este tipo de redacciones. Estos apartados son:

- **Objetivo**. En este apartado se comentarán de manera más específica los objetivos del proyecto, así como aquellos que sean propios de cada subsistema.
- **Metodología**. Este capítulo se aprovechará para detallar la manera en la que se ha trabajado para la realización de este proyecto, así como las tecnologías, aplicaciones y lenguajes utilizados.
- **Resultados**. Este capítulo será el más extenso, ya que en él se detallará el proceso de desarrollo, las decisiones que se han tomado a lo largo del mismo, y los artefactos resultantes.
- **Conclusiones**. En esta última sección se hará una reflexión sobre los resultados obtenidos y si cumplen con los objetivos que se han especificado para los mismos.

Cabe destacar que la mayoría de estos capítulos estarán divididos a su vez en tres secciones, ya que la redacción de esta memoria sigue la misma filosofía de afrontar las tres secciones del proyecto de manera separada.

[7] https://www.blackmagicdesign.com/
[8] https://obsproject.com/
[9] https://bitfocus.io/companion

CAPÍTULO 2
Objetivo

El objetivo de este TFG es la refactorización de ciertos componentes necesarios en la creación, celebración y grabación o transmisión de concursos personalizados. Por tanto, se abordan tres problemáticas muy distintas entre sí, que consisten en algunas de las necesidades más urgentes por parte de la empresa para realización de dichos eventos, y que son:

- **Visualizador de preguntas.** El sistema anterior, basado en Unity, era muy propenso a errores y se trataba de una aplicación pesada, lo cual producía un muy bajo rendimiento y muchas dificultades a la hora de crear nuevos concursos. Se busca una solución más flexible y eficiente que cumpla con las reglas de estilo del concurso y que respete también la existencia de animaciones y demás factores que aporten dinamismo a la aplicación.
- **Controlador DMX.** En los eventos anteriores los focos eran controlados a través de Freestyle, una aplicación poco intuitiva y usable. Se busca realizar una aplicación propia, capaz de controlar estos dispositivos de la manera más sencilla posible, así como poder definir movimientos o animaciones predefinidas de forma sencilla y a alto nivel.
- **Producción a través del ATEM Blackmagic.** Tras el uso de aplicaciones externas para la producción de contenido multimedia como OBS, se desea realizar dicha producción a través del propio dispositivo ATEM para aumentar su fiabilidad. También se desea utilizar una botonera (o Streamdeck) para lanzar las distintas macros o acciones predefinidas que tenga el dispositivo, de manera que su uso sea lo más rápido y sencillo posible.

2.1. REQUISITOS COMUNES

Tras comprobar los distintos frentes a los que se enfrenta este trabajo, es importante reconocer cuáles son los requisitos de cada uno. Algunos de ellos son comunes a estos tres subproyectos, concretamente los siguientes:

- **Los sistemas deberán ser fiables.** Estos concursos se realizan en directo, por lo que es de vital importancia que todos estos sistemas puedan recuperarse rápidamente de posibles errores. Las soluciones deben ser estables para evitar problemas que afecten a la celebración de dichos eventos.
- **Las soluciones deben ser usables.** Estos sistemas implican un cierto grado de interacción, en la mayoría de los casos siendo en directo, y manejados por un encargado con poco margen de error o tiempo. Por ello, los resultados deben ser aplicaciones intuitivas, claras y que permitan la correcta celebración de los concursos.
- **Las aplicaciones no deben depender de la conexión a Internet.** Estos sistemas estarán conectados a una red interna, de manera que puedan comunicarse entre sí y con los demás componentes que, si bien son externos a este trabajo, son igualmente vitales para la celebración de estos concursos. Por tanto, las aplicaciones resultantes no deben depender de servicios ni bibliotecas alojados en la nube.

- **Compatibilidad e integración.** Las nuevas aplicaciones deben ser compatibles con el resto de componentes y sistemas que participan en la celebración de los concursos. Al cambiar sólo una fracción del entorno completo, este cambio debe ser transparente para aquellos componentes que no se desea actualizar, de forma que no sea necesaria la actualización de todos los demás factores externos a este trabajo. Por tanto, las aplicaciones deben comunicarse en la medida de lo posible de la misma manera que se hacía antes, aceptando y enviando los mismos mensajes y acciones.
- **Personalización.** El sistema debe ser altamente personalizable, y responder de manera rápida y eficaz a los distintos diseños, ya sean de preguntas y mecánicas como visuales (tanto del concurso como de los recursos necesarios para la producción de contenido). Estos cambios deben poder hacer hacerse de manera que aseguren un funcionamiento correcto en el mínimo tiempo posible, ya que se desea minimizar el tiempo de preparación necesario para cada concurso, así como la complejidad del proceso.

2.2. VISUALIZADOR DE PREGUNTAS

Para el visualizador de preguntas, los requisitos propios son:

- **Mayor flexibilidad y agilidad**. Unity ha demostrado ser útil, pero el proceso de preparación de cada concurso es tedioso y propenso a fallos. Se desea buscar una solución que sea más eficiente y flexible, a ser posible que utilice una única "plantilla" para todas las preguntas.
- **Respeto del aspecto audiovisual**. El diseño general de la página debe mantenerse, respetando tanto el apartado visual como los efectos de sonido o las transiciones y efectos que se utilizaban en Unity.

2.3. SERVIDOR DMX

Para el servidor DMX, el objetivo principal es crear una aplicación propia para que el control de los dispositivos sea más sencillo e intuitivo que en la aplicación Freestyler, que no cumplía suficientemente con los requisitos de usabilidad esperados. Sin embargo, sí se desean mantener algunas de las funciones de esta aplicación, como la definición de comportamientos prefabricados de manera que no haya que mover manualmente cada dispositivo siempre que se vaya a necesitar.

- **Interfaz de usuario sencilla.** Se desea obtener una interfaz de usuario que sea más sencilla e intuitiva que Freestyler, ya que esta aplicación no se consideraba lo suficientemente usable.
- **Definición de componentes de alto nivel.** Es necesario contar con la capacidad de almacenar animaciones o órdenes prefabricadas con la intención de agilizar el proceso de preparación y celebración de los concursos, minimizando el trabajo previo y la posibilidad de errores de definición.
- **Facilitar la integración con el resto de módulos**. Aunque la integración forma parte de los requisitos comunes, es necesario reiterar en ella en este subproyecto en concreto, ya que en futuras actualizaciones se planea construir sobre él para, por ejemplo, desarrollar nuevos componentes de alto nivel que utilicen esta tecnología.

2.4. CONFIGURACIÓN DEL DISPOSITIVO ATEM

Esta sección cuenta con varios requisitos propios, y podría argumentarse que se trata de la parte del trabajo que supone mayor cambio para el funcionamiento anterior del proyecto. Estos requisitos son:

- **Uso del Atem Studio Switcher**. Se desea abandonar intermediarios innecesarios, como era OBS. Por esta razón la solución debe hacerse a través de la configuración del propio dispositivo para que haga la producción de manera nativa. Es especialmente aconsejable el uso y definición de macros, colecciones de instrucciones que se ejecutan simultáneamente, para la definición de ciertas utilidades y acciones.

- **Control desde Stream Deck**. La empresa cuenta con una botonera, un aparato ampliamente usado para la ejecución de macros gracias a su enorme capacidad de personalización. Es por esta razón que se desea poder controlar el Atem desde la misma, lo cual simplificaría mucho su uso.
- **Animaciones y fondos animados**. La solución debe incluir ciertas animaciones con el objetivo de hacer el vídeo resultante más agradable e interesante a la vista. También debe haber fondos animados para, al igual que con el visualizador de preguntas, se respete el aspecto previo de la aplicación.
- **Manual de uso**. Se debe escribir un breve manual de uso y configuración para permitir un correcto aprovechamiento de este trabajo en futuros eventos.

CAPÍTULO 3
Metodología

3.1. METODOLOGÍA DE DESARROLLO

3.1.1. Desarrollo Rápido de Aplicaciones (RAD)

Para el desarrollo de las dos aplicaciones resultantes (pues la tercera fase del proyecto no necesitó de ningún desarrollo, tan sólo una investigación sobre distintas aplicaciones y manuales) se ha utilizado una metodología de Desarrollo Rápido de Aplicaciones (RAD). De esta forma, en todo momento se cuenta con prototipos que permiten comprobar la correcta integración de los mismos con el resto de componentes, evitando potenciales problemas de compatibilidad o la necesidad de reestructurar o rehacer grandes fragmentos de código.

Siguiendo esta filosofía, al inicio cada fase se acordaron una serie de requisitos, y se otorgó acceso a los repositorios del proyecto existentes (o se crearon aquellos que hicieran falta). Una vez se contaba con todo lo necesario, el desarrollo del software se abordó de forma iterativa e incremental, comenzando con prototipos poco o nada funcionales, que permitían asegurar en todo momento un buen funcionamiento y una base estable para más tarde ir refinando su funcionamiento o aspecto.

Estos avances eran después compartidos con el supervisor del proyecto, que comprobaba la correcta evolución del trabajo y proporcionaba una retroalimentación así como una guía para la siguiente iteración. Además, en todo momento se mantuvo un contacto para responder rápidamente a dudas o problemas que pudieran surgir en el proceso.

De esta manera, se estableció un ciclo semanal en el que se desarrollaban cambios o se implementaban nuevas funciones, para más tarde recibir una retroalimentación por parte de la empresa. Esta retroalimentación, a su vez, serviría para el nuevo ciclo, enfocando el trabajo en las partes más necesarias o "críticas".

Por otro lado, en el caso del tercer módulo del proyecto, no se realizó un desarrollo de software como tal, sino una investigación sobre el manual de uso del dispositivo ATEM, así como un trabajo de investigación sobre el funcionamiento del software propio del aparato, así como sobre soluciones, aplicaciones y proyectos de la comunidad de usuarios.

Cabe destacar que el trabajo ha combinado las modalidades presenciales y telemáticas durante su desarrollo, respondiendo a las necesidades del proyecto. De esta manera, se ha trabajado a distancia cuando era posible, pero cuando se ha hecho necesario el uso de focos, cámaras y demás dispositivos se ha llevado a cabo el desarrollo desde la oficina de la empresa.

También es importante mencionar la periodicidad de las reuniones, que podrían dividirse en dos tipos:

- **Con el supervisor de la empresa**. Estas reuniones (virtuales en su mayoría) se han realizado de manera semanal, con el objetivo de llevar un registro del trabajo realizado, así como de otorgar ciertas directrices para la siguiente semana. Además, cuando se ha hecho algún avance importante o alcanzado algún hito de desarrollo se han celebrado otras reuniones para poder mostrarlos y consultar si cumplen lo especificado.

- **Con el tutor del TFG**. Estas reuniones fueron presenciales y mensuales. El objetivo de las mismas era mostrar el resultado del trabajo sobre esa fase, así como introducir los requisitos de la siguiente. Habitualmente, se anotaban también ciertos cambios o actualizaciones que debían realizarse sobre el módulo resultante. Al igual que con las reuniones semanales, estos encuentros podían hacerse con anterioridad debido a avances importantes, y el supervisor de la empresa también estaba presente en la mayoría de los casos.

3.1.2. Gestión de versiones

Para poder llevar un correcto control de las versiones y los cambios que se realizan sobre el programa se utilizó la técnica de *branching*, que se basa en distintas ramas sobre las que se trabaja y se hacen los cambios necesarios. La estructura de estas ramas puede variar entre proyectos, pero en general se siguió el flujo de trabajo *GitFlow*. Como se ha mencionado anteriormente, el objetivo de este desarrollo no es crear una nueva aplicación, sino refactorizar la existente (si bien es cierto que en el segundo módulo se creó un servidor completamente nuevo). Es por ello que en la primera fase del proyecto no se creó un nuevo repositorio, sino que se utilizó una nueva rama dentro del repositorio de *BitBucket* de la empresa. Para la segunda fase de este proyecto sí se creó un nuevo repositorio, en el que se utilizaron dos tipos de ramas:

- **Rama Master**. Esta rama se trata de la principal del proyecto, en la que se encuentra la última versión "oficial" del programa, aquella que es estable y que otorga una base para futuras modificaciones.
- **Ramas Feature**. En este caso fueron dos. Las ramas de este tipo son aquellas dedicadas al desarrollo de nuevas funcionalidades, de manera que puedan hacerse modificaciones y adiciones al código sin comprometer el funcionamiento correcto de la versión que se encuentra en la rama Master. La primera rama se utilizó para el desarrollo del propio servidor, siendo el grueso del trabajo y aquella sobre la que más cambios se han realizado. La segunda, por otra parte, fue utilizada de forma puntual, pues decidí reestructurar el proyecto y también añadir un archivo *.gitignore*, que permite especificar los directorios o ficheros que no se desea sincronizar en el repositorio (normalmente archivos de las bibliotecas, de configuración del entorno virtual, etc). Al ser un cambio que podría llegar a ser problemático, el uso de una segunda rama se consideró una opción segura y necesaria.

3.1.3. Herramientas utilizadas

Durante el trabajo se han utilizado numerosos dispositivos y programas, que se detallan en esta sección.

3.1.3.1. Herramientas Hardware

- **Ordenador portátil ASUS TUF GAMING F15**. En este ordenador se ha llevado a cabo la práctica totalidad del desarrollo.
- **Placa Raspberry Pi 3B+**. Esta placa se ha utilizado para alojar el servidor encargado de recoger, almacenar, verificar y traducir los archivos JSON que se han creado con la intención de configurar los focos y dispositivos DMX.
- **Controlador DMX Enttec Ode MK2**. Este dispositivo es el que se encuentra conectado a la red LAN utilizada por todos los componentes implicados en la celebración de los concursos y que sirve de intermediario entre los aparatos DMX y el servidor alojado en la Raspberry.
- **Dos focos FL-233 BEAM**. Estos focos son los utilizados en los concursos, y se colocaron en la oficina para realizar las pruebas de funcionamiento de la segunda parte del proyecto. También fue necesario el manual de uso[1] para conocer las especificaciones en cuanto a cantidad de canales y correspondencia de cada uno.

[1] https://www.manualslib.com/manual/2624172/Flash-Fl-233-Beam.html

- **ATEM 2 M/E Production Studio 4K**. Este dispositivo ATEM es el que se ha estado utilizando en la producción audiovisual de los eventos, y es el que se ha configurado y sobre el que se han creado las macros y vistas necesarias, tareas para las que ha sido vital el uso del manual de uso[3].
- **Stream Deck de 15 botones de Elgato**. Este aparato es el que se ha utilizado para lanzar las macros y transiciones entre distintas vistas del ATEM. Cuenta con 99 páginas para sets distintos de 15 botones (12 si eliminamos los dedicados a cambiar de página).
- **Tres cámaras de vídeo de la serie IR DOME, con conexión HD-SDI a 1080p**. Al ser la configuración del ATEM dirigida a un evento con numerosas cámaras y fuentes de vídeo, se han utilizado 3 cámaras enfocando a lugares distintos de la oficina para poder comprobar correctamente el cambio entre ellas y su funcionamiento.

3.1.3.2. Herramientas Software

- **Windows 10 Pro**. Sistema operativo utilizado principalmente en la tercera fase para instalar todos los programas necesarios para la correcta configuración y utilización del ATEM.
- **Ubuntu 22.04**. Sistema operativo utilizado sobre todo en las dos primeras fases. La razón de no haber hecho todo el desarrollo en Windows es que el uso de Linux y la consola se ha vuelto muchas veces más sencillo a la hora de instalar las dependencias y de programar las aplicaciones necesarias.
- **Ubuntu Server 20.04**. Sistema operativo de la placa Raspberry Pi.
- **Git**. Sistema de versiones empleado para poder gestionar correctamente los cambios realizados y las ramas que se han utilizado en el desarrollo.
- **BitBucket**. Servicio de repositorios utilizado por la empresa FK, por lo que los componentes a refactorizar se encontraban ya en esta aplicación. Además, en el caso de los programas nuevos se ha decidido seguir usando BitBucket por coherencia con el resto del sistema.
- **Slack**. Aplicación de comunicación orientada al entorno profesional y utilizada por la empresa. Ha sido el medio principal de comunicación con el supervisor.
- **Teams**. Herramienta utilizada para las reuniones de control semanales.
- **Notion**. Software de gestión de proyectos en el que se encuentra la mayoría de la documentación del sistema. Además, es en esta aplicación donde se ha escrito el manual de uso del ATEM para añadirlo a la guía existente para la creación y celebración de los concursos.
- **Visual Studio Code**. IDE utilizado para todas las tareas de programación, así como de edición de los archivos XML utilizados para guardar la configuración del dispositivo ATEM.
- **Chrome Dev Tools**. Colección de herramientas de desarrollo integradas en Google Chrome que se ha utilizado para la depuración y las pruebas de las distintas aplicaciones desarrolladas (pues todas ellas se basan en servidores web).
- **ATEM Software Setup**. Aplicación de Blackmagic destinada a configurar ciertos aspectos como la dirección IP del dispositivo, y actualizar el software interno del ATEM.
- **ATEM Software Control**. Aplicación de Blackmagic destinada a la utilización y configuración del ATEM más orientada al uso normal (macros, vistas, etc).
- **Bitfocus Companion**. Aplicación utilizada para servir de intermediaria entre el ATEM y el Stream Deck, que también sirve para configurar la propia botonera.
- **Your Animated Transition Macro (YATM)**[10]. Aplicación de código abierto creada por un grupo de usuarios de Blackmagic con la intención de facilitar la creación de transiciones animadas para los ATEM compatibles con la vista SuperSource (una vista con hasta 4 vistas secundarias).

3.1.3.3. Lenguajes de programación y bibliotecas

- **HTML**. *HyperText Markup Language*[13], se trata de un lenguaje de marcado utilizado para crear páginas web. Se ha utilizado tanto en el visualizador de preguntas como en el controlador DMX, ya que ambos se tratan de aplicaciones basadas en el navegador.
- **CSS**. *Cascading Style Sheets*[12], es un lenguaje para poder definir hojas de estilo en cascada. Al igual que antes, se ha utilizado en los dos primeros subproyectos, siendo vital especialmente en el visualizador de preguntas, pues debía adecuarse al estilo visual del visualizador anterior.
- **JavaScript**[14]. Este es un lenguaje de programación de alto nivel orientado a definir ciertos *scripts* o comandos que permiten darle a las páginas web ciertos comportamientos o incluso capacidad de interacción. Al igual que los dos lenguajes anteriores, ha sido vital en el primer y segundo módulos.
- **Python**[9]. Un lenguaje de programación también de alto nivel. Gracias a ciertas bibliotecas, es un muy recomendable para la programación de servidores web, aunque se trata de un lenguaje bastante generalista. Ha sido especialmente útil en la segunda fase, aunque en la primera ha sido necesario hacer ligeros cambios en los servidores hechos previamente por la empresa, también basados en este lenguaje.
- **XML**. *eXtensible Markup Language*, se trata de un lenguaje de marcado, muy útil para almacenar datos estructurados de manera que sea fácil de leer. En la tercera fase ha sido necesario utilizar este lenguaje, pues la aplicación de control de Blackmagic almacena todos sus datos en este formato.
- **Chart.js**[4]. Se trata de una biblioteca de código abierto para JavaScript creada para facilitar la visualización de datos a través de ciertos gráficos, como diagramas de barras, de sectores, etc. Esta biblioteca ha sido útil para implementar el comodín del público en el visualizador de preguntas, pues se utiliza un gráfico de barras para mostrar los botos del público sobre cuál es la respuesta correcta a una determinada pregunta.
- **TextFit**[18]. Este componente, también de código abierto e implementado en JavaScript, se utiliza para poder ajustar el tamaño de las letras de un texto a su contenedor HTML. Esto ha sido útil también en el visualizador de preguntas, pues no hay manera de saber cómo de larga va a ser una pregunta o respuesta, y de esta manera aseguramos que va a aprovechar correctamente el espacio que se le ha dado.
- **Flask**[16]. Esta biblioteca para Python sirve para, de manera sencilla, crear servidores web basados en Python, y se ha utilizado en los dos primeros subproyectos.
- **DMX**[8]. *Digital MultipleX*, es un protocolo utilizado para el control de focos, fuentes de luz, máquinas de humo y demás maquinaria común en la celebración de espectáculos. Este protocolo es bastante común en dicho campo, y es compatible con un enorme rango de aparatos.
- **PyArtNet**[20]. Se trata de una biblioteca de código abierto basada en Python que permite facilitar la comunicación del servidor web con el controlador DMX. Su colección de instrucciones simplifica el proceso, pues se tratan de métodos que traducen los parámetros dados a mensajes del protocolo ArtNet, diseñado para transmitir órdenes DMX a través de una red Ethernet.
- **JSON**. En este caso se hace referencia a dos dependencias distintas. Por un lado, se ha utilizado el formato de archivos JSON (*JavaScript Object Notation*[7]), un tipo de archivos de texto orientados al intercambio de datos, que se ha utilizado para definir los datos necesarios para el visualizador de preguntas y las órdenes DMX. Por otro lado, la biblioteca básica de Python con el mismo nombre se ha utilizado para poder leer dichos archivos y extraer los datos en el caso del servidor DMX.
- **JSONSchema**[2]. Esta biblioteca, también de Python, permite el uso de esquemas JSON, en los que se puede definir la estructura de un archivo y así establecer ciertos datos obligatorios, su tipo, valores mínimos o máximos, etc. La biblioteca se encarga de leer estos esquemas (definidos

en sus propios archivos) y comprobar si un JSON recibido cumple la estructura en cuestión. Esto es muy importante en el servidor DMX, pues así se desechan aquellos archivos con datos incorrectos, campos inexistentes, etc.

- **Threading**. Esta biblioteca básica de Python está orientada a la concurrencia y a la definición de hilos de ejecución, utilizada también en el servidor DMX. En un principio no se contemplaba su uso, pero luego se hizo evidente su necesidad al comprobar que ciertos tipos de órdenes dejaban el servidor bloqueado durante su ejecución (que podía ser infinita en el caso de los bucles).
- **Asyncio**. Esta biblioteca básica de Python permite el uso de instrucciones asíncronas. Es la biblioteca que utiliza PyArtNet para funcionar, y es necesaria para la llamada de métodos asíncronos. La estructura del programa, a grandes rasgos, crea un hilo a través de *Threading*, para desde él ejecutar la tarea asíncrona que se encarga de interpretar y "activar" las órdenes.
- **Toastr**[6]. Esta biblioteca para HTML permite la creación y configuración de notificaciones. Fue útil en el desarrollo del controlador DMX, pues el uso de alertas HTML normales bloqueaba la ejecución y ralentizaba al usuario, mientras que con Toastr las notificaciones pueden dejarse a un lado de la página, de forma que sean visibles pero no bloqueen el uso de la aplicación.

CAPÍTULO 4

Resultados

En esta sección se mostrará tanto el proceso como el resultado de las distintas fases ya mencionadas del proyecto. Para mantener una cierta organización, se ha dividido este capítulo en tres secciones que tratarán específicamente cada subproyecto en el orden en el que fueron abordados.

4.1. VISUALIZADOR DE PREGUNTAS

El primer paso de esta fase fue analizar el funcionamiento del módulo que estaba en uso antes del inicio de este trabajo. La aplicación, también basada en el navegador, hacía uso de Unity WebGL. Esta tecnología no se adecuaba completamente al objetivo de la empresa, pues su utilización era bastante poco eficiente al ser una aplicación pesada y lenta, además de ser bastante propensa a los errores (por ejemplo, se dieron casos en los que desaparecían letras del texto).

Por esta razón se decidió utilizar HTML a través de un servidor implementado en Flask. A través de estas tecnologías podemos definir "plantillas", con sus propios campos, hojas de estilo y comportamientos, de manera que sólo necesitamos definir el formato de la pregunta una vez y aplicarlo a todas ellas. Otra ventaja es que no supone un cambio notable en la estructura del proyecto, pues simplemente se ha sustituido el componente web correspondiente a Unity por todo un árbol de componentes HTML. Además, se puede hacer uso de las interfaces existentes en el proyecto, que permiten a la aplicación recibir todos los parámetros necesarios (texto de la pregunta, de las respuestas, ubicación del componente multimedia, etc) para poder aplicarlos sobre la marcha.

Tras decidir la tecnología que se iba a usar, se tomó nota del estilo visual y comportamiento que tenía la web original, de forma que el nuevo prototipo fuese lo más fiel posible al mismo. Se ejecutaron también diversos escenarios, para entender la manera en la que los componentes interactuaban entre ellos y cómo reaccionaban a ciertos eventos.

Tras estas pruebas, se comprobó que el visualizador está dividido en tres zonas bien definidas, de manera que cada una ocupa un tercio de la pantalla en vertical. La primera zona sirve para mostrar el gráfico de barras cuando se necesite el comodín del público, quedando vacía en el resto de casos. La zona central es donde se encuentran a su vez otras tres secciones diferenciadas: un pequeño visualizador multimedia, el texto de la pregunta y las cuatro opciones (o los caracteres ocultos en el caso de las preguntas de ahorcado), además de un pequeño temporizador en la zona superior derecha de esta zona. Por último, la parte de la derecha tiene la lista de jugadores o equipos concursantes, su puntuación y el turno en el que podrán responder. Todas estas zonas se pueden ver diferenciadas en la figura 4.1, que se encuentra más adelante y se corresponde con el primer prototipo que se hizo, precisamente con la intención de marcar las distintas secciones del visualizador

También es necesario implementar la posibilidad de mostrar multimedia a pantalla completa, como introducciones o las pantallas de los comodines, así como diversos efectos de sonido, cambios de color (las respuestas tendrán tres colores además del azul dependiendo si están seleccionadas o de si son correctas o incorrectas, y un caso similar se da en los nombres de los participantes al cambiar

de turno). Otro elemento multimedia que no se ve a primera vista es una señal de prohibido, que aparecerá junto al nombre de un participante que haya sido penalizado.

Por último, es importante anotar la presencia de animaciones en prácticamente todos los elementos, ya sea para la entrada o salida de las preguntas y respuestas, para los cambios de turno o de color, etcétera, además de la adición de efectos de sonido para las transiciones y distintos eventos que se han mencionado en las líneas anteriores.

Tras anotar todos estos datos, se decidió abordar las siguientes porciones de forma separada:

- **Video Comodín**. Este componente es invisible la mayoría del tiempo. Se trata de un vídeo que ocupa la totalidad de la pantalla y que cubre todos los demás elementos. Se utiliza para poder reproducir vídeos como introducciones, animaciones sobre el comodín escogido o un fondo animado mientras comienza el concurso. Aunque el nombre pueda engañar, es importante recalcar que no se trata de un vídeo para comodines exclusivamente, sino que se ha escogido llamarlo de esa manera debido a la gran cantidad de usos que puede tener.
- **Pregunta**. Esta porción se encuentra en el centro de la pantalla, y comprende cuatro componentes principales. En el tercio superior se encuentra un recurso multimedia (ya sea una imagen o un vídeo) que acompaña las preguntas. En el centro se encuentra el texto de la pregunta, y en la parte inferior se posicionan las respuestas, que pueden ser cuatro opciones o una sola palabra que se va revelando en el tiempo en el caso de las preguntas de ahorcado. El último componente importante es un pequeño cuadro de texto en la zona de arriba a la derecha, que muestra el tiempo que le queda a un participante para contestar.
- **Participantes**. Colocada a la derecha, esta sección está bastante menos congestionada, y muestra los tres participantes, sus puntuaciones y el orden en el que pueden responder. Además, en el caso de recibir una penalización aparecerá una pequeña imagen que lo ilustre.
- **Comodín del público**. La porción derecha de la pantalla se encuentra vacía durante la gran mayoría del tiempo (de hecho, es posible que durante toda la ejecución). Sin embargo, cuando se utiliza el comodín del público se utilizará el espacio para mostrar un gráfico de barras que muestre los votos de los asistentes.
- **Ranking**. Esta escena eclipsa toda la pantalla y muestra la puntuación y orden de los distintos participantes del público. Además, debe ser posible cambiar de página (ya que la cantidad de participantes puede ser bastante alta) y contar con sus propias animaciones.

Las secciones siguientes tratarán cada una de estas porciones de manera separada, emulando así el proceso de desarrollo seguido en el que se trabajaba sobre una de las porciones para luego pasar a la siguiente. También conviene considerar que no se acababa completamente una sección antes de pasar a la próxima, sino que, al seguir una metodología iterativa e incremental, se iban haciendo pequeños avances en todas ellas hasta que se construyó la aplicación por completo.

Antes de empezar con las porciones, es destacable también que primero se hizo un prototipo básico para definir las zonas presentes. Para no perder mucho tiempo con esa primera toma de contacto, que era más un *placeholder* que una aplicación al uso, simplemente se utilizaron elementos *div* para definir la posición de estas zonas. Después se les puso un color básico como fondo para poder verlos en el navegador y se pasó al desarrollo específico de cada uno.

Estas secciones, además, abordarán los tres aspectos de su desarrollo en aquellos casos que se considere interesante o central: HTML (estructura), CSS (estilo), JavaScript (comportamiento). Como varios elementos y secciones pueden compartir métodos (por ejemplo, el método que carga inicialmente la página edita tanto los nombres de los participantes como la pregunta y sus respuestas), se abordarán en todas las apariciones que se consideren importantes o destacables, poniendo el foco en los fragmentos de los mismos que sean importantes para el elemento en cuestión.

Un último detalle a mencionar es que, como se podrá reparar a la hora de comprobar los archivos CSS, la gran mayoría de coordenadas o tamaños se han especificado en unidades relativas (ya sea al

Figura 4.1: Primer prototipo de visualizador, que consistía más bien en una definición de las distintas zonas.

tamaño de ventana o del contenedor). La razón de esto es que la página pueda mantener el mismo aspecto a la hora de reescalar la ventana del navegador para ocultar su interfaz en la celebración de los concursos.

4.1.1. Primera sección: Vídeo comodín

Esta sección fue la primera en abordarse al ser la más sencilla. Aunque el nombre pueda dar lugar a error, este componente no se encarga de los comodines del concurso (o al menos no en su totalidad). la razón de llamarlo así es que este elemento funciona como una especie de "herramienta multiusos", reproduciendo todos aquellos vídeos y recursos que deban mostrarse en pantalla completa y tapando el resto de elementos del concurso.

Estos recursos pueden ser vídeos que se reproduzcan una sola vez o en bucle, y formar parte de un comodín o de el desarrollo habitual del concurso (la introducción, un vídeo en bucle mientras se colocan el público y los participantes, etcétera). En el segundo caso, el único que tiene la posibilidad de un bucle, se conocen de antemano los nombres de los archivos, por lo que cuando se solicita una entradilla, basta con leer el mensaje para saber si se va a tratar de un bucle o no, y aplicar esa información al documento HTML a través de las variables existentes para ello.

Listado 4.1: Método encargado de cargar las entradillas cuando se recibe un mensaje de ese tipo.

```
1  socket.on('question_viewer_change_video_entradilla', function ↩
       ↪ (msg) {
2      try {
3        var entradilla = new Object();
4        entradilla.url = window.location.href;
5        entradilla.url = ↩
             ↪ entradilla.url.replace('question_viewer_intro_only', ↩
             ↪ '');
6        entradilla.url = entradilla.url + ↩
             ↪ 'static/Assets/videos/opening/' + msg + '.mp4';
7        if (msg == 2) {
8          entradilla.loop = true;
9        }else {
10         entradilla.loop = false;
11       }
12
13       let display = document.querySelector('#joker-video');
14       display.style.visibility = 'visible';
15       display.innerHTML='';
16
```

```
17        let video = document.createElement('video');
18        video.addEventListener('ended', finEntradilla, false);
19        video.setAttribute('src', entradilla.url);
20        video.loop = entradilla.loop;
21        video.autoplay = true;
22        display.appendChild(video);
23      } catch (error) {
24        console.error('Error en ↩
              ↪ question_viewer_change_video_entradilla');
25      }
26    });
```

En el código del listado 4.1 se puede ver cómo se trata este mensaje (nombrado como *msg*). El primer bloque de código, que va desde las líneas 3 a la 11, se encarga de configurar la entradilla, guardando si se trata de un bucle o no, así como la ruta al archivo necesario. El segundo (líneas 13 a 15) visibiliza el elemento encargado de estos recursos, además de eliminar cualquier posible recurso que hubiera antes. Por último, con el código que se encuentra entre las líneas 17 y 22 se carga el recurso nuevo, además de añadir un manejador de evento que se ejecutará cuando acabe el vídeo (este manejador simplemente vuelve a esconder el elemento y a borrar a sus nodos hijos).

De esta explicación se puede extraer que los nodos hijos se eliminan dos veces. Esto no es accidental, sino que permite que, incluso si hay un error a la hora de borrarlos en el primer intento, ese error no vaya a durar más de lo que se tarde en lanzar de nuevo la entradilla.

Otro detalle destacable es que todo el código está dentro de un bloque *try-catch*. De esta manera se pueden manejar los posibles errores y, lo que es más interesante, se puede ubicar fácilmente en qué método se encuentra dicho error.

En el caso de la introducción y de los comodines, sólo se reproduce una única vez, por lo que hay que añadir un evento para el momento en el que el vídeo termine, y un método que recoja dicho evento y oculte el vídeo.

4.1.2. Segunda sección: Pregunta

Esta sección está, como se ha mencionado antes, dividida en varias secciones, que también se abordarán de manera separada.

4.1.2.1. Recurso multimedia

Todas las preguntas van acompañadas de una imagen o vídeo para ilustrar la imagen, ya sea como simple decoración o porque la pregunta hace referencia a la misma. La implementación de este recurso es bastante sencilla en comparación con la de otros elementos, pues no necesita más que un elemento HTML. Sin embargo, al haber tanto imágenes como vídeos no se puede implantar directamente en la estructura del árbol.

Debido a esto, en el contenedor de toda esta porción se creó otro contenedor secundario, específico de este recurso. Dicho contenedor permanecerá vacío hasta que se reciba un mensaje para crear la página. Como este mensaje incluye un archivo JSON con los datos necesarios, al leerlo puede comprobarse si se trata de una imagen o un vídeo, y utilizar el elemento HTML correspondiente. También conviene mencionar que el mensaje recibido no incluye el archivo multimedia como tal, sino que simplemente proporciona una ruta para que el propio documento HTML pueda establecerla como fuente.

El formato es la única diferencia que debe haber entre vídeos e imágenes, por lo que a la hora de aplicar un estilo en CSS se ha creado un selector general para ambos (incluye todos aquellos elementos “hijos” del contenedor).

El comportamiento del programa, sin embargo, sí debe ser distinto en los dos casos (pues se debe crear un elemento de tipo diferente). Es por eso que en los métodos encargados de cargar los datos se

comprueban los campos para ver de qué tipo es. Además, para el caso de los vídeos deben existir las opciones de pausar, reanudar, reiniciar o saltar al último frame. Estos métodos, para evitar errores, comprueban que el elemento se trata de un vídeo, y no hacen ninguna acción en caso negativo.

Listado 4.2: Método encargado de cargar los recursos multimedias y incrustar el elemento necesario en el HTML

```
let media = document.querySelector('#media');
      if(obj.data_pregunta.img_url) {
        let aux = document.createElement('img');
        media.appendChild(aux);
        let top = getComputedStyle(aux).getPropertyValue('top');
        if(top =='') top = 0;
        aux.remove();

        let img = document.createElement('img');
        img.setAttribute('src', obj.data_pregunta.img_url);
        media.appendChild(img);
        img.style.top = '-100vh';
        setTimeout(()=>{
          img.style.top = top;
        }, 250);
      } else if(obj.data_pregunta.video_url != ''){
        let aux = document.createElement('video');
        media.appendChild(aux);
        let top = getComputedStyle(aux).getPropertyValue('top');
        if(top =='') top = 0;
        aux.remove();

        let video = document.createElement('video');
        video.setAttribute('src', obj.data_pregunta.video_url);
        video.autoplay = true;
        media.appendChild(video);
        video.style.top = '-100vh';
        setTimeout(()=>{
          video.style.top = top;
        }, 250);
      }
```

Como se puede comprobar en el fragmento de código 4.2, la manera de comprobar el tipo de recurso es dependiendo de si en el mensaje aparece un campo *img_url* o *video_url*. Además, se puede comprobar que en un principio el recurso se crea "fuera" de la pantalla. De esta manera, usando la función *setTimeout* y el campo *transition* en CSS podremos crear una animación sencilla para que su aparición sea menos repentina y más agradable a la vista.

Por último, al igual que con los demás elementos, también se añadieron animaciones de entrada y salida. Para ello se sigue el mismo método que se ha seguido en todos ellos, estableciendo el formato de las transiciones a través de CSS, distinguiendo entre antiguos y nuevos componentes y moviendo cada uno a su posición (los antiguos fuera de la pantalla, y los nuevos dentro).

4.1.2.2. Pregunta

La generación del texto de la pregunta es probablemente el más sencillo, pues no necesita responder a ninguna interacción (menos el momento en el que se cambia de pregunta). Además, se mantiene estático durante todo el tiempo que esté en pantalla, y lo único que hay que hacer (en un principio) es recoger el texto del mismo archivo en el que se envían todos los datos y colocarlos.

Sin embargo, hay una complicación. El texto puede variar mucho en longitud, y es imposible saber cómo será de larga la siguiente pregunta. Por tanto, establecer un tamaño de letra fijo haría que algunas preguntas se vean demasiado pequeñas y con mucho espacio libre, mientras que otras podrían llegar incluso a desbordar su contenedor. No sólo eso, sino que comprobar las preguntas

con antelación para establecer el mejor tamaño de letra antes de la celebración del concurso sería demasiado tedioso y poco eficiente (habría que ir comprobando manualmente todas las preguntas, lo cual es precisamente una de las razones por las que se decidió iniciar este proyecto en primer lugar).

Para resolver este problema se vuelve especialmente útil la biblioteca TextFit. Esta biblioteca está hecha para poder ajustar el tamaño de letra al máximo posible sin desbordar el contenedor, y permite mucha personalización (por ejemplo, la posición del texto, tamaños mínimo y máximo, entre otros).

Esta biblioteca funciona de manera iterativa, comenzando por el tamaño de letra mínimo que se haya establecido, y aumentándolo hasta que se detecte un desbordamiento. Un inconveniente de TextFit es que no tiene en cuenta la posibilidad de hacer un salto de línea, algo que sí tiene HTML en cuenta. Por esa razón, aunque un texto parezca caber con mucho margen en la caja que lo contiene, de manera interna el navegador la considera una única línea, por lo que no sigue aumentando su tamaño al interpretar que va a haber (o ya ha habido con el tamaño mínimo) un desbordamiento.

La solución de este problema es sencilla. Se ha establecido un número de caracteres arbitrario, veinticinco. Cuando una línea tiene esa cantidad o más, el programa esperará al siguiente espacio en blanco e insertará un salto de línea HTML (
). De esta manera se divide el texto en varias líneas para que TextFit pueda aumentar su tamaño de forma correcta. No obstante, para evitar la creación de textos demasiado "verticales", se ha decidido limitar esta cantidad de líneas a seis.

Listado 4.3: Fragmento de código que introduce los saltos de línea cada veinticinco caracteres sin cortar las palabras.

```
let question = document.createElement('div');
      question_container.insertBefore(question, answers);
      question.style.left = '-100vw';
      question.classList.add('question');
      let questionText = obj.data_pregunta.pregunta;
      let charNumber = 0, magicNumber = 25, linesCount = 1;
      for(let i=0;(i<questionText.length && linesCount<=6);i++){
        if(questionText.charAt(i) == '␣' && ↩
            ↪ charNumber>=magicNumber){
          questionText = questionText.replaceAt(i,'<br>');
          linesCount++;
          charNumber = 0;
        }else{
          charNumber++;
        }
      }
```

4.1.2.3. Respuestas

Las respuestas son posiblemente una de las partes más complejas del visualizador de preguntas, pues es donde se encuentra la mayoría de la interacción. Estas respuestas deben ser capaces de responder a la selección de las mismas, a dos de los comodines y a la comprobación del resultado (tanto si es correcto como incorrecto), además de la revelación de distintas letras en el caso de las preguntas de ahorcado.

El archivo recibido a la hora de cargar una pregunta incluye cinco campos necesarios para las respuestas (seis si tenemos en cuenta el campo que define el tipo de pregunta). Estos campos son las 4 respuestas, y cuál es la correcta.

Esto también se aplica a las preguntas de ahorcado, en los que, de forma interna, la respuesta A siempre es la correcta, la B es una cadena vacía y las dos restantes son la cadena "H_A_N_G_M_A_N". Esto se debe a que antes se usaban estas respuestas para comprobar el tipo de la pregunta, mientras que el campo mencionado anteriormente que sirve para definir el tipo se empezó a usar hace menos tiempo. Por ello el programa comprueba tanto el campo "type" como el campo "C", de manera que aseguremos que las preguntas de ahorcado siempre se recogen de manera correcta.

Tras este tratamiento general, primero se explicará el caso de las respuestas cuádruples.

Estas preguntas van ordenadas de la A a la D, y por su formato forman una serie de elementos y contenedores. Las respuestas se almacenan en un contenedor general, de forma similar al texto de la pregunta y al recurso multimedia. Tras esto, hay cuatro subcontenedores, uno por respuesta. Dentro de ellos se encontrará el texto de la pregunta, la letra y un componente llamado "pertX" (siendo X la letra de la pregunta) que servirá para el comodín del público.

Al estar todas las respuestas en contenedores separados se pueden definir estilos propios para cada una, lo cual es importante pues las preguntas pueden estar hasta en cuatro estados distintos: inactivas, seleccionadas, incorrectas o correctas. Además, para darle más dinamismo a las animaciones de entrada y salida se ha hecho que no se muevan todas juntas, sino que haya un pequeño desfase entre ellas, lo que hace aún más necesaria esta separación.

Como estas preguntas pueden estar sujetas a varios tipos de eventos, se mencionarán en forma de lista para más claridad:

- **Selección de una respuesta**. Cuando se selecciona una respuesta (esto es haberla elegido sin haber comprobado todavía si es correcta o incorrecta) debe comprobarse primero que no se trata de una pregunta de ahorcado ya que, como se ha dicho antes, de manera interna no se distingue entre preguntas, sino que en el caso del ahorcado el encargado de gestionar el concurso marcará la respuesta A si es correcta o cualquier otra si es incorrecta. Tras esto se añade la clase "selected" a la pregunta seleccionada, que cambia su color a un tono naranja, y elimina esa misma clase de todas las otras respuestas que la tuvieran. En una ejecución normal sólo habría una respuesta más con ese atributo, pero al comprobarlas todas aseguramos que en el caso de que un error permitiera la selección de más de una en la siguiente selección quede corregido.
- **Marcar una respuesta como correcta**. En este caso también cambia el comportamiento dependiendo de si se trata de una pregunta de ahorcado o no. Más adelante se tratará el caso de las preguntas de ese tipo especial, pero en el caso básico lo que se hace es comprobar la letra del mensaje para retirar la clase "selected" y añadir la clase "correct", que volverá la respuesta de color verde.
- **Marcar una respuesta como incorrecta**. Este caso es prácticamente idéntico al anterior, pero en vez de añadir la clase "correct" se añade la clase "incorrect", que la volverá de color rojo.
- **Comodín 50/50**. Este comodín consiste en la eliminación de dos respuestas incorrectas. En un principio se escogió borrarlas de forma aleatoria, pero más tarde se reparó en que el mensaje que activaba este evento iba acompañado de un archivo que especificaba cuáles borrar, por lo que simplemente se seleccionan las respuestas en cuestión y se hacen desaparecer. Para ello se cambia el cambio de visibilidad del propio elemento, lo cual siempre sobrescribirá cualquier otra regla CSS. De esta manera se asegura que una respuesta siempre se va a borrar cuando sea necesario, si bien es cierto que técnicamente no se borran, sino que se esconden. La razón de esto es que de esta manera la estructura interna del documento no se altera y así se evita que, por ejemplo, al eliminar las respuestas A y C las dos respuestas restantes se muevan a la zona superior.
- **Comodín del público**. Si bien es cierto que el funcionamiento principal de este comodín se encuentra en otra sección de la página, el gráfico de barras puede no ser lo suficientemente claro. Es por ello que las respuestas también se alteran ligeramente para asegurar que se entiende la información recogida. El primer cambio es que las respuestas cambian su letra por el porcentaje de votos que la han seleccionado (por ejemplo, en vez de A pondrá "30 %"). En segundo lugar, se hace uso del componente llamado "pertX" mencionado anteriormente. Este componente está realmente vacío, pero tiene un color más oscuro y una opacidad reducida, de manera que al colocarse sobre las respuestas estas se convierten a su vez en un segundo

gráfico de barras, orientado horizontalmente, en el que cada respuesta presenta una silueta mayor o menor dependiendo de los votos que reciba.

Listado 4.4: Métodos encargados de los controles básicos de las respuestas.

```
socket.on('question_viewer_marcar_opcion', function (msg) {
  try {
    if(!document.querySelector('.hangman')){
      let markedAnswers = document.querySelectorAll('.selected');
      for (let i=0;i<markedAnswers.length;i++){
        markedAnswers[i].classList.remove('selected');
      }

      let answer = document.querySelector('#answer'+msg);
      answer.classList.add('selected');
    }
  } catch (error) {
    console.error('Error en question_viewer_marcar_opcion');
    console.error(error);
  }

});

socket.on('question_viewer_marcar_solucion_correcta', function (msg) {
  try {
    if(document.querySelectorAll('.hangman').length == 0){
      let answer = document.querySelector('#answer'+msg);
      answer.classList.remove('selected');
      answer.classList.add('correct');
    } else {
      let answerChars = document.querySelectorAll('.hangman>span');
      clearInterval(hangmanInterval);
      hangmanInterval = null;
      for(let i=0;i<answerChars.length;i++){
        answerChars[i].textContent = hangmanSolution.charAt(i);
      }
      count = null;
    }
  } catch (error) {
    console.error('Error en question_viewer_marcar_opcion_correcta');
    console.error(error);
  }
});

socket.on('question_viewer_marcar_solucion_incorrecta', function (msg) {
  try {
    if(document.querySelectorAll('.hangman').length == 0){
      let answer = document.querySelector('#answer'+msg);
      answer.classList.remove('selected');
      answer.classList.add('incorrect');
    }else {
      if(!hangmanInterval){
        hangmanInterval = setInterval(newLetter, miliHangman);
      }
    }
  } catch (error) {
    console.error('Error en question_viewer_marcar_opcion_incorrecta');
    console.error(error);
  }
```

```
55
56      });
```

Como se puede comprobar, el tratamiento varía en función de si la pregunta se de cuatro opciones o de tipo ahorcado. Por ejemplo, es imposible marcar una opción en las preguntas del segundo tipo (ya que estas "opciones" sólo existen de manera interna). Además, cuando se marca una solución correcta es necesario mostrar la palabra completa del ahorcado, y cuando se trata de una respuesta incorrecta se debe proseguir mostrando letras.

Se especifica la necesidad de seguir con el intervalo que muestra las letras porque el sistema permite parar el vídeo de una pregunta, para lo cual se decidió que sería un comportamiento deseable que ocurriera lo mismo con el proceso de desvelar la palabra.

4.1.2.4. Temporizador

El temporizador es de los pocos componentes del visualizador que funcionan de manera local. El servidor es capaz de pausarlo, reanudarlo, activar o desactivar y cambiar el tiempo restante, pero la cuenta regresiva se hace en el propio navegador.

Esto aumenta tanto la fiabilidad como el rendimiento, ya que un método que se ejecuta en el tiempo, reduce un número y cambia el texto de un componente consume menos recursos que enviar un mensaje cada segundo, procesarlo y aplicar el cambio.

Además, aprovechando que es uno de los únicos mensajes a los que sí podríamos anticiparnos, podemos evitar errores como, por ejemplo, que no llegue un mensaje en algún segundo, lo que provocaría que el temporizador se quedase congelado durante mínimo dos segundos. Con esta decisión, por tanto, se evitan errores y un consumo innecesarios.

Respecto al funcionamiento como tal es sencillo: se aprovecha la capacidad de establecer un intervalo en JS para ejecutar un método que reduce la cuenta de segundos restantes y aplica el cambio al texto. Además, dependiendo de los segundos restantes podría reproducirse un efecto de sonido para añadir dinamismo al concurso.

Cuando el programa recibe la señal de iniciar el contador hace que el temporizador sea visible, tras lo cual se inicia el intervalo. Además, como el intervalo comprueba una variable global para el programa, puede aumentarse o reducirse el tiempo a voluntad desde otros métodos. Para la funcionalidad de pausar o reanudar basta con manipular la ejecución de dicho intervalo, cancelándolo o iniciándolo según sea necesario.

4.1.3. Tercera sección: Participantes

En esta sección (la derecha) se encuentran los nombres de los tres participantes, ya sean equipos o jugadores individuales. Además, también deben verse las puntuaciones de cada uno y el turno en el que se encuentran.

La primera parte que se abordó en esta sección fue a los participantes y sus puntuaciones. Como los diferentes mensajes que se envían para configurar el visualizador y los puntos incluyen toda la información necesaria consiste en leerla y aplicarla a los campos necesarios.

Para ello se escogió una estructura basada en un contenedor para los tres nombres (de forma que a la hora de colocarlos no ocupen toda la pantalla). Dentro de él se han creado tres nuevos contenedores (uno por participante) en los cuales, con una disposición de tabla o "grid" se han ubicado los nombres, puntuaciones y el icono (invisible casi siempre) de penalización.

La segunda es más compleja, y se trata de mostrar el turno de un determinado jugador. Esto se hace de dos maneras:

- Por un lado, se cambia el color del nombre según el orden. Esta parte es bastante sencilla ya que, al igual que con las respuestas, se puede añadir una clase al elemento HTML dependiendo del turno que sea y, a través de CSS, aplicar el color necesario.

- Por otro lado, los números de las posiciones deben moverse de su lugar inactivo (en la zona superior) y colocarse a la izquierda del nombre a quien corresponda. En un primer momento fue bastante sencillo, pues de forma interna se trataban de componentes que aparecían y desaparecían dependiendo de la posición. Sin embargo, a la hora de animar el efecto no era el deseado (los números desaparecían de un sitio y aparecían en otro, mientras que se deseaba que se movieran de su lugar al punto que les correspondiera).

Para resolver la problemática de las posiciones se almacenaron la coordenadas originales de los números (de forma que pudieran volver a su lugar si fuese necesario) y se cambió el código encargado de colocar los números en su sitio para que, en vez de añadir o retirar la visibilidad de varios componentes, recogiera las coordenadas de un determinado participante y después mueva un único componente (correspondiente a la posición en cuestión) al lugar necesario.

Esto reveló un nuevo problema, ya que las coordenadas se almacenaban en píxeles. Aunque no es algo malo per se, la aplicación debe ser capaz de cambiar de tamaño ya que, al estar basada en un navegador, será necesario ponerlo en pantalla completa para que no se vea ningún otro componente de la interfaz del navegador. Debido a esto, se tuvo que añadir un nuevo método que actualizara estas posiciones iniciales en caso de un reescalado de ventana.

El principal punto débil de este enfoque es que puede dar problemas si hay números en su posición inicial y otros al lado de un participante en el momento del reescalado. Sin embargo, desde la empresa se consideró que era un error trivial y que no iba a llegarse a esa situación durante la celebración del concurso. Además, se puede arreglar rápidamente en plena ejecución (colocando todos los números en la misma situación, inactivos o en un participante, y reescalando dos veces), por lo que se entendió que, en caso de darse, sería durante la preparación del evento y podría resolverse fácilmente antes de comenzar. Por tanto, se decidió no abordar el error, o al menos darle una baja prioridad y continuar con otras fases más importantes del proyecto.

Como se puede comprobar en el fragmento de código 4.5, lo único que se hace en caso de reescalado es devolver los turnos a su posición original, recoger la referencia para su reposicionamiento, y dejarlos de nuevo en la posición que estuvieran.

Listado 4.5: Almacenamiento de las posiciones originales de los turnos y readaptación en caso de reescalado

```
 var turnNumbersOriginalPositions = {
  turn1: document.querySelector('#turn1').getBoundingClientRect(),
  turn2: document.querySelector('#turn2').getBoundingClientRect(),
  turn3: document.querySelector('#turn3').getBoundingClientRect(),
}

...

window.addEventListener('resize', (event) => {
  let top1 = document.querySelector('#turn1').style.top;
  let left1 = document.querySelector('#turn1').style.left;
  let top2 = document.querySelector('#turn2').style.top;
  let left2 = document.querySelector('#turn2').style.left;
  let top3 = document.querySelector('#turn3').style.top;
  let left3 = document.querySelector('#turn3').style.left;

  document.querySelector('#turn1').style.top = '0px';
  document.querySelector('#turn1').style.left = '0px';
  document.querySelector('#turn2').style.top = '0px';
  document.querySelector('#turn2').style.left = '0px';
  document.querySelector('#turn3').style.top = '0px';
  document.querySelector('#turn3').style.left = '0px';

  turnNumbersOriginalPositions.turn1= ↩
      ↪ document.querySelector('#turn1').getBoundingClientRect();
```

```
    turnNumbersOriginalPositions.turn2= document.querySelector('#turn2').getBoundingClientRect();
    turnNumbersOriginalPositions.turn3= document.querySelector('#turn3').getBoundingClientRect();

    document.querySelector('#turn1').style.top = top1;
    document.querySelector('#turn1').style.left = left1;
    document.querySelector('#turn2').style.top = top2;
    document.querySelector('#turn2').style.left = left2;
    document.querySelector('#turn3').style.top = top3;
    document.querySelector('#turn3').style.left = left3;
});
```

Por último, se añadieron tres iconos de castigo, uno a la derecha de cada participante, que mostrarán y ocultarán según el desarrollo del concurso lo vuelva necesario (además de ir acompañados de un efecto de sonido en el caso de mostrarlos).

4.1.4. Cuarta sección: Comodín del público

Este comodín fue el último en implementarse al ser el más complejo. Además, sufrió bastantes cambios hasta que se llegó a un resultado satisfactorio.

Además de mostrar las barras de votos en las respuestas, se deseaba crear un gráfico para ver con claridad la cantidad de votos de cada respuesta.

Para ello se utilizó la biblioteca Chart.js, una biblioteca de código abierto que permite mostrar datos estadísticos con una gran personalización y numerosas opciones y extensiones. Gracias a esta biblioteca se ahorró mucho trabajo, ya que no hizo falta desarrollar todo un método para crear el gráfico, sino que bastó con configurar el gráfico a través de objetos de JS y proporcionarle los datos cuando fuera necesario.

En un primer lugar se decidió hacer un gráfico de sectores, con las secciones en colores distintos y etiquetas que mostrase la respuesta a la que pertenecía cada sector, junto al porcentaje. Para ello se escogió utilizar la extensión DataLabels, que permitía una mayor personalización del texto y ubicación de las etiquetas.

Sin embargo, se acabó descartando la idea por que se consideraba que sería más legible e intuitivo un gráfico de barras. Esto volvió innecesaria la utilización de DataLabels, ya que podía mostrarse la etiqueta de cada barra y el porcentaje de votos sobre la misma utilizando sólo la configuración básica de Chart.js, por lo que se eliminó la extensión.

El último cambio fue el cambio de la paleta de colores, que dejó de tener un color para cada barra y empezó a presentar el mismo color en todas las barras, siendo este el mismo color de las respuestas para más cohesión.

Además del gráfico de barras, como se ha mencionado anteriormente y se puede comprobar en la figura 4.2, se utiliza un elemento para hacer de "gráfico auxiliar" sobre las respuestas, de manera que es más intuitivo y accesible. Además, para eliminar por completo la posibilidad de error de comprensión, se sustituye la letra de una respuesta por el porcentaje de votos que ha recibido.

4.1.5. Quinta sección: Ranking

Esta sección consiste en una lista ordenada de todos los participantes del público, en grupos de veinte personas.

Esta lista debe mostrar la posición, el nombre y la puntuación de cada participante. Para la representación de estos datos de manera consistente (ya que, por ejemplo, los nombres pueden variar mucho en longitud y se desea que las columnas de la tabla se mantengan ordenadas) se decidió utilizar un elemento HTML con la propiedad *display: grid*. Esta propiedad permite, precisamente,

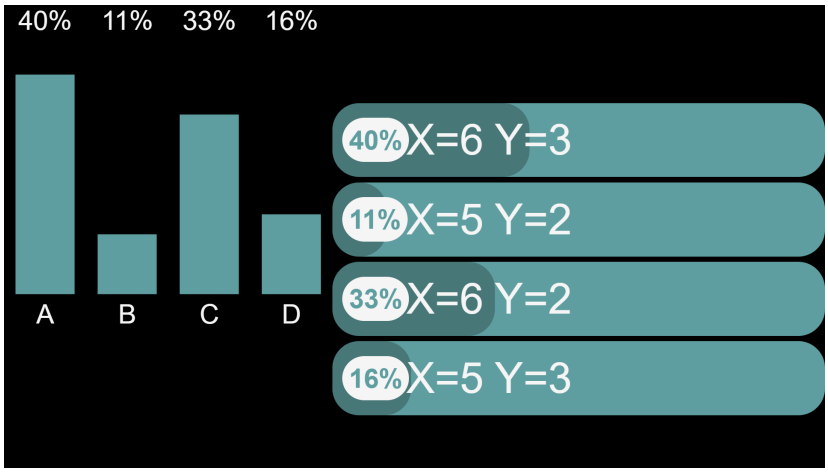


Figura 4.2: Aspecto del gráfico perteneciente al comodín del público, y la manera en la que afecta a las respuestas.

Ranking

Pos.	Nombre	Puntos	Pos.	Nombre	Puntos
1º	Juan Eduardo Vitoria García	343	11º	Iván Coello	270
2º	Daniel Almansa	327	12º	Sergio Casas Soria	262
3º	Alvaro Santos Martin Nieto	318	13º	Daniel Muñoz Fuentes	259
4º	David Sánchez Barrangán	313	14º	Santiago Navarro García	259
5º	Iván Rodrigo Alvaro	298	15º	Roberto Escudero Cruz	252
6º	Omar Moya Romero	297	16º	Julianjra	252
7º	Alvaro Fernández	289	17º	Víctor Sevilla Sánchez	249
8º	María Garrido	287	18º	Celia	247
9º	Malena Diez Viñas	284	19º	Manuel Aviles	243
10º	Elena Martín	278	20º	Sergio Sanchez Botey	240

Figura 4.3: Aspecto del ranking resultante.

crear estructuras similares a tablas, manteniendo una gran capacidad de personalización respecto a posiciones, tamaño de celdas, márgenes, etcétera.

Otro frente de esta sección fue la necesidad de las animaciones. Para volver estas animaciones más eficientes se decidió mover las filas en lugar de cada elemento por sí solo, lo cual reduce a un tercio la cantidad de movimientos y creaciones de elementos necesarias. Al igual que con el resto de elementos del visualizador, para la propia animación se utilizaron transiciones definidas a través de CSS en combinación con *timeouts*. Estos temporizadores se separaron en cien milisegundos, de manera que cuando se incluyen las filas lo hacen de forma ordenada, similar a una "escalera".

Por otra parte, la problemática más sencilla fue la capacidad de distinguir entre distintos tipos de participantes (por ejemplo, en el concurso utilizado para realizar las pruebas se distingue entre participantes de Bachillerato, Formación Profesional, etcétera). Para ello tan sólo hay que filtrar la lista de participantes entrante y cambiar el texto del título del ranking.

Por último, se añadieron tres pequeños elementos HTML para colocarse de fondo en las tres primeras posiciones, emulando medallas y aportando al atractivo visual del ranking.

4.1.6. Vistas separadas

Con la intención de facilitar la producción a través de diversas fuentes de vídeo, se consideró que sería especialmente útil contar con vistas separadas de los distintos elementos del visualizador. De esta manera se han creado ciertas rutas alternativas, que incluyen los siguientes elementos:

- Recursos multimedia.
- Texto de la pregunta.
- Respuestas y gráfico del comodín del público.
- Jugadores y sus puntuaciones.
- Ranking.

Para su inclusión bastó con reutilizar el mismo código que el del visualizador general, eliminando todos aquellos métodos, elementos e instrucciones que no afectaran a la sección que se está separando. Por último, se necesitó retocar también los archivos HTML y CSS, debido a que al ser vistas propias ya no era necesaria la inclusión de elementos contenedores, y por otro lado todas las partes que se quedarían en la vista necesitaban un reescalado para ocupar toda la pantalla.

4.2. SERVIDOR DE CONTROL DMX

Este módulo se volvió necesario debido a que la empresa no se encontraba cómoda utilizando el sistema anterior, que se basaba en el uso de la aplicación FreeStyler. Esta aplicación era demasiado engorrosa y poco intuitiva, y configurar las animaciones o movimientos de los focos podía ser una tarea bastante tediosa y con tendencia al error.

Se decidió utilizar de nuevo un servidor desarrollado en Flask, de manera que este componente mantuviera una cierta homogeneidad con el resto del sistema. Además, gracias a la potencia de dicho framework, así como del propio lenguaje Python, era posible desarrollar una pequeña aplicación que permitiera la creación de estas animaciones, de manera que se eliminara completamente la necesidad del software utilizado anteriormente.

Además, otra razón para escoger Python fue la existencia de una biblioteca para dicho lenguaje. Durante las investigaciones iniciales se encontró que la gran mayoría de bibliotecas estaban desarrolladas para Arduino, junto a unas pocas de Python. Se eligió el segundo lenguaje para poder darle uso a una tarjeta Raspberry Pi que se tenía en el momento, de manera que no fuera necesario adquirir una nueva tarjeta Arduino.

Para las pruebas se utilizaron los dos focos que se pueden ver en la figura 4.4. En esta figura se ve también una máquina de humo, y merece la pena aclarar que, aunque el servidor no tiene problemas de compatibilidad y es capaz de comunicarse con cualquier dispositivo DMX conectado correctamente, la máquina de humo quedó fuera de las pruebas debido a que estaba defectuosa.

4.2.1. DMX y ArtNet

Para el uso de ArtNet se ha escogido la biblioteca PyArtnet[1], que sirve para "traducir" ciertas órdenes y métodos a mensajes ArtNet. Esta biblioteca es de código abierto y se encuentra alojada en un repositorio de GitHub. Esta biblioteca está diseñada para Python y, pese a no tener apenas documentación, es bastante sencilla de usar.

4.2.2. Comunicación de órdenes al servidor, esquemas JSON

Se decidió que el servidor utilizaría archivos JSON para definir los mensajes y órdenes DMX. Esta decisión se basa en que el formato JSON puede ser bastante más intuitivo y fácil de crear o modificar (bastaría con un editor de texto sencillo), además de ser mucho más personalizable. Si bien es cierto que lo más cómodo sería una pequeña aplicación para ayudar a configurar los valores de cada parámetro, se decidió que no se haría en este proyecto debido a la carga de trabajo y al tiempo necesario para ello.

El formato de estos archivos sigue una especificación muy concreta a través de un esquema JSON, un lenguaje que permite definir la estructura que debe seguir un archivo de este tipo.

[1] https://pypi.org/project/pyartnet/

Figura 4.4: Conexión de los focos al controlador (el bloque azul de la derecha)

La estructura desarrollada se basa en un título, una descripción y la definición de uno o varios estados (es la manera en la que se va a llamar de ahora en adelante a un conjunto de valores de canales DMX). Estos estados, además, podrán ir acompañados de un valor que definirá el tiempo que debe durar la transición de un estado a otro. Sin embargo, uno de los campos más importantes es el tipo de orden. Para la creación del servidor se ha decidido contar con tres tipos de orden:

- **Orden DMX**. Se ha decidido poner este nombre al tipo de órdenes más simples que se han definido. Estas órdenes tienen un único estado definido, y no admiten la especificación de un tiempo, pues se presupondrá que es 0 (al darse este valor, la biblioteca calcula el movimiento más rápido para todos los dispositivos, teniendo en cuenta el *framerate* configurado en PyArtnet (25 por defecto). Este tipo de órdenes es útil cuando se trata de establecer un determinado estado de la manera más rápida posible.
- **Animación DMX**. Este tipo de orden tiene uno o más estados (en un principio se distinguió entre animaciones y animaciones múltiples, siendo estas últimas aquellas con dos o más estados, pero durante el desarrollo se determinó que era innecesario distinguirlos). Además, estos estados irán acompañados de un valor que defina el tiempo necesario para realizar la animación (siendo 0 el mínimo). Tras recibir un archivo de este tipo, PyArtNet realizará una interpolación lineal (aunque pueden configurarse otro tipo de interpolaciones) entre cada estado de manera secuencial y en el orden en el que se han especificado en el JSON. Una vez acaben todas las animaciones, los dispositivos se mantendrán en el último estado definido.
- **Loop DMX**. Estas órdenes son bucles, colecciones de dos o más estados (con valor de temporizador, al igual que las animaciones) que el servidor ejecutará de forma secuencial. Su funcionamiento es idéntico al de las animaciones normales, pero una vez se haya alcanzado el último estado se volverá al primero, ejecutándose toda la colección hasta que se reciba la orden de detenerse. Al igual que con las animaciones, en un principio se distinguió entre bucles simples (de dos estados) y bucles con más estados, pero se acabó decidiendo que era innecesario distinguirlos. Cabe destacar que el mínimo de dos estados se basó en la aparente imposibilidad de conocer de forma fiable el estado anterior de los dispositivos (si bien es cierto que al avanzar en el trabajo se encontró una manera de hacerlo, pero se decidió que no haría falta actualizar los bucles para incluir el estado anterior).

Entre estos tres tipos hay diversos campos, algunos de ellos comunes y otros propios de cada uno. Son los siguientes:

- ***Type.*** Este campo define de qué tipo de orden se trata este JSON, escogiendo entre los tres mencionados anteriormente.
- ***Description.*** Este campo permite describir la orden para que luego, desde la interfaz gráfica,

pueda conocerse su función.

- ***Devices.*** Este campo es único de las órdenes simples, y define todos los dispositivos que se desean utilizar. En ellos se almacenan objetos JSON anidados que almacenan el canal de inicio del dispositivo, cuántos canales ocupa y una colección de los valores de cada uno (estos campos se llaman *start*, *width* y *values*).
- ***Animations.*** Este campo se encuentra en el esquema para los bucles y las animaciones. Contiene varias instancias de *devices*, de manera que en cada paso de la animación puedan usarse dispositivos distintos. Además, incluye un campo *timer* para definir la velocidad en la que se debe ejecutar un paso. Este valor se encuentra dentro de *devices*, lo cual puede hacer que el uso de *animations* parezca redundante. La mayor diferencia entre ellas es una cuestión de concurrencia. Todos los estados definidos en un mismo campo *devices* se producen a la vez (aunque su valor de tiempo puede hacer que unos sean más rápidos que otros). Mientras tanto, en *animations* esto no ocurre, por lo que no se hayan terminado de realizarse todos los estados de un objeto *devices* no se pasará al siguiente. Otro detalle a valorar es que los bucles y las animaciones pueden parecer idénticos a nivel de formato, pero hay una diferencia: los bucles necesitan un mínimo de dos animaciones. Esto se debe a que una animación DMX se entiende como una transición entre dos estados, por lo que si se desea hacer un bucle debe poder volverse al estado anterior para repetirlo, de forma que en realidad lo que se hace es alternar un estado inicial y final (en el caso mínimo).

Listado 4.6: Esquema DMX para la definición de bucles, en los que se pueden comprobar los campos mencionados

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Loop DMX",
  "description": "JSON que define un bucle DMX con varios pasos secuenciales (mínimo 2)",
  "type": "object",
  "properties":{
    "type": {
      "description": "Identificador del tipo de JSON que se está enviando",
      "type": "string"
    },
    "title": {
      "description": "Título de la animación que se está enviando",
      "type": "string"
    },
    "description": {
      "description": "Descripción de la orden que se está enviando",
      "type": "string"
    },
    "color": {
      "description": "Color que se desea poner en el botón de la página.",
      "type": "number",
      "minimum": 0,
      "maximum": 15
    },
    "animations": {
      "description": "Lista de las distintas animaciones a ejecutar, en orden secuencial",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "devices": {
```

```
            "descripcion": "Colección de los distintos dispositivos a editar y los valores que se deben aplicar",
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "start": {
                  "description": "Canal DMX en el que comienza la colección de canales correspondientes al dispositivo",
                  "type": "number",
                  "minimum": 0,
                  "maximum": 255
                },
                "width": {
                  "description": "Número de canales DMX pertenecientes al dispositivo",
                  "type": "number",
                  "mimimum": 1,
                  "maximum": 255
                },
                "values": {
                  "description": "Valores que se deben aplicar a los distintos canales. DEBE TENER TANTOS VALORES COMO EL VALOR WIDTH",
                  "type": "array",
                  "items": {
                    "type": "number",
                    "minimum": -1,
                    "maximum": 255
                  },
                  "minItems": 1
                },
                "timer": {
                  "description": "Tiempo en milisegundos que debe durar la animación",
                  "type": "number",
                  "minimum": 0
                }
              },
              "required": ["start", "width", "values", "timer"]
            },
            "minItems": 1
          }
        },
        "required": ["devices"]
      },
      "minItems": 2
    }
  },
  "required": ["type", "animations", "title", "description"]
}
```

Cabe destacar que, aunque DMX soporta valores entre 0 y 255, se decidió que sería útil permitir un valor de -1 cuando no se quiera editar el valor de un canal. Esto puede ser importante cuando no se conozca exactamente el valor anterior, o cuando sólo se desee cambiar un valor concreto (por ejemplo, cambiar el color de los focos, estén en la posición que estén). Esta funcionalidad, por tanto, no es trivial, ya que si se sigue el ejemplo anterior, evita tener que crear una animación para cada posición posible en la que cambie sólo el color, y permite la creación de una única animación, lo que ahorra mucho tiempo, esfuerzo, y riesgo de errores.

Una vez creados los esquemas se comenzó con el desarrollo del servidor. Al igual que con el visualizador, se siguió una metodología iterativa e incremental, comenzando por fabricar un "esqueleto", un servidor con las rutas que fuesen a ser necesarias, pero devolviendo mensajes con el código HTTP 501 (mensajes "Not Implemented" para utilizar como "placeholders" a la vez que se comprobaba que el servidor era capaz de recibir y responder a las solicitudes).

El servidor, además, utiliza un objeto denominado DMX_CONTROLLER, que es el encargado de utilizar PyArtNet para enviar mensajes al propio controlador DMX, un dispositivo que procesa los mensajes ArtNet, los transforma en mensajes DMX y los envía por el universo necesario.

Una vez se había generado una estructura general, se fue abordando el tratamiento de cada tipo de orden, así como la comprobación de los esquemas JSON a través de la biblioteca para Python "jsonschema". El tipo más problemático fueron los bucles, ya que el servidor no respondía hasta haber acabado la orden, siendo de una duración desconocida y normalmente larga en este caso. Es por ello que se decidió recurrir a Asyncio y Threading, dos bibliotecas orientadas a hacer programas asíncronos y concurrentes, resolviendo así el problema (ahora el servidor responde cuando ha comprobado que la orden es ejecutable, siendo así cuando el formato del archivo JSON es correcto, sin esperar a acabar la propia orden).

También cabe destacar que, en las fases iniciales del proyecto, no se utilizó el navegador para enviar mensajes al servidor, sino que se utilizó IPython junto con la biblioteca Requests como una manera sencilla de enviar solicitudes y archivos de manera rápida, lo cual es muy útil en una situación en la que se necesitaba probar los cambios sobre la marcha.

Como se puede ver en el código del listado 4.7, siempre que se recibe una orden se comienza haciendo una breve comprobación para evitar posibles errores más adelante (ya que, cuando el ancho de un dispositivo y el número de canales no coincide la biblioteca lanza una excepción tras una serie de procesos que podríamos ahorrarnos con esta comprobación).

Tras ello, se inicializa el objeto que representa el dispositivo controlador de DMX. Esto es así en todos los métodos, de manera que se asegure que, independientemente de cuál se realice primero o de si ha habido errores en la creación, todos los métodos puedan funcionar por sí mismos. Se puede reparar también en una variable *self.aux*. Esta variable permite parar cualquier animación o bucle en el acto, ya que a través de otro método su valor puede colocarse en cero, deteniendo el bloque *while*.

Una vez se ha hecho esto, se realizan una serie de bucles anidados entre ellos, tratando en primer lugar cada animación de forma separada, y dentro de las mismas cada "paso". El proceso de las órdenes como tal es sencillo. Se recogen los valores de los distintos canales, se aplican, y se guarda cuál es el canal más lento. Esto último se hace porque si le pidiéramos al nodo que "espere" a otro canal, la animación puede detenerse sin que el canal que más tiempo necesita haya acabado, de manera que, por ejemplo, un foco pueda quedarse "colgado" a mitad de su movimiento por haber cambiado de color más rápidamente.

Listado 4.7: Ejemplo de código encargado de tratar las órdenes de bucles

```
async def loop(self, loop):
    '''Implementacion del bucle con una o varias animaciones DMX'''
    for animation in loop['animations']:
      for device in animation['devices']:
        if device['width'] != len(device['values']):
          raise Exception('El ancho del dispositivo y el numero ↩
              ↪ de valores deben de ser iguales')

    await self.node.start()
    if not self.universe:
      self.universe = self.node.add_universe(15)

    values = []
    self.aux = 1
```

```
    max_timer = -1

    while self.aux > 0:
      for animation in loop['animations']:
        for device in animation['devices']:
          try:
            channel = self.universe.get_channel ↩
                ↪ (str(device['start'])+'/'+str(device['width']))
          except:
            channel = self.universe.add_channel ↩
                ↪ (start=device['start'], width=device['width'])

          for i in range(0, len(device['values'])):
            if device['values'][i] == -1:
              values.append(channel.get_channel_values()[i])
            else:
              values.append(device['values'][i])

          if device['timer'] > max_timer:
            max_timer = device['timer']
            slowest_channel = channel
          channel.add_fade(values, device['timer'])

          values = []

        await slowest_channel.wait_till_fade_complete()
        max_timer = -1
    self.aux = 0
    await self.node.stop()
```

4.2.3. Control a través de una página web

Tras el desarrollo de todos los tipos de órdenes (como se ha mencionado anteriormente, fueron 5 en un principio y en las últimas fases se rebajaron a 3), se trabajó en la página web. Esta página tenía como requisitos ser lo más clara e intuitiva posible, permitiendo con un vistazo conocer cuáles son las opciones disponibles, de forma que el servidor pueda utilizarse con agilidad y rapidez durante la celebración del concurso.

También se decidió que las órdenes debían poder guardarse en el lado del servidor, de manera que no haya que ocupar tiempo y recursos en seleccionar archivos y enviarlos (aunque esta opción se encuentra existente para órdenes improvisadas o menos comunes). De esta forma, al cargarse la página el navegador solicitará la lista de órdenes guardadas en el servidor y, a la hora de ejecutarlas, simplemente enviará una solicitud con el nombre de la orden a ejecutar. No obstante, la presencia de muchas órdenes podía reducir la usabilidad de la página al hacerla menos intuitiva por la gran cantidad de títulos y botones, ordenados por orden alfabético por defecto.

Es por esta razón que se decidió añadir una nueva variable al esquema que definiera un color. Este color no tiene ningún tipo de influencia en el servidor, pero permite hacer una clasificación visual en la página web, además de alterar el orden de las órdenes para clasificarse por colores, aumentando la sencillez de su uso.

A estas órdenes se les añadieron dos botones: el que activa la orden y el botón de información, que solicita al servidor que le devuelva el texto de la descripción de un archivo. Esta información se muestra a la derecha de la página, a través de la biblioteca Toastr.

Para estas órdenes se creó un elemento HTML personalizado, que sirve como plantilla para que todas las opciones de ejecución se muestren de manera encapsulada, con todos los campos y botones necesarios. Este elemento podrá replicarse tantas veces como sea necesario (concretamente, tantas veces como órdenes prefabricadas tenga el servidor). De esta forma la creación de estos bloques es

bastante más sencilla y eficiente, ya que no hace falta dedicar muchas líneas de código a la creación de cada bloque con todos sus elementos interiores.

Listado 4.8: Código para la creación de los elementos personalizados destinados a las órdenes DMX

```
class Campo_orden extends HTMLElement{
 constructor(){
   super();
   let template = document.querySelector('#campo-orden');
   let clone = template.content.cloneNode(true);
   let shadowRoot = this.attachShadow({
     mode: 'open'
   });
   shadowRoot.appendChild(clone);
 }

 connectedCallback(){
   this.title = this.hasAttribute('data-title')? ↩
       ↪ this.getAttribute('data-title'):'TITULO␣DESCONOCIDO';
   this.shadowRoot.querySelector('h2').textContent = this.title;
   this.shadowRoot.querySelector('.order'). ↩
       ↪ addEventListener('click', ↩
       ↪ this.orderButtonListener.bind(this));
   this.shadowRoot.querySelector('.info'). ↩
       ↪ addEventListener('click', ↩
       ↪ this.infoButtonListener.bind(this), false);
 }

 async orderButtonListener(event){
   toastr.options.timeOut = 60000;
   toastr.options.extendedTimeOut = 60000;
   try{
     const url = '/prefab-json';
     const cabeceras = {
       'Content-Type': 'application/json'
     };
     const payload = {
       title: this.title,
     };
       const request = {
           method: 'POST',
           headers: cabeceras,
       body: JSON.stringify(payload),
       };
         let response = await fetch(url, request);
     if(response.status != 200){
       let responseText = await response.text();
       toastr.error('Ha␣habido␣un␣error␣inesperado.␣Consulta␣↩
           ↪ la␣consola␣para␣más␣información.', this.title, {
         position: 'top-right',
       });
       throw new Error (responseText);
     }
     toastr.success('Orden␣ejecutada␣con␣éxito', this.title, {
       position: 'top-right',
     });
     } catch(e){
         console.error(e);
     }
 }

 async infoButtonListener(event){
   toastr.options.timeOut = 60000;
   toastr.options.extendedTimeOut = 60000;
```

```
54      try{
55        const url = '/order-info/'+this.title;
56        const cabeceras = {
57          'Content-Type': 'application/json'
58        };
59          const request = {
60              method: 'GET',
61              headers: cabeceras
62          };
63            let response = await fetch(url, request);
64        let responseText = await response.text();
65        if(response.status != 200){
66          toastr.error('Ha habido un error inesperado. Consulta ↩
              ↪ la consola para más información.', this.title, {
67            position: 'top-right',
68          });
69          throw new Error (responseText);
70        }
71        toastr.info(responseText, this.title, {
72          position: 'top-right',
73        });
74        } catch(e){
75            console.error(e);
76        }
77    }
78  }
79  customElements.define('campo-orden', Campo_orden);
```

En el listado 4.8 se puede comprobar la creación de un elemento de este tipo. Los dos primeros métodos serán los encargados de crear el elemento como tal y de introducir los datos pertinentes, así como de enlazar a los botones los manejadores de evento necesarios.

Los otros dos métodos son los encargados de hacer las peticiones al servidor DMX utilizando únicamente el nombre de cada orden. Más tarde, el servidor DMX tratará estos nombres para buscar el archivo correspondiente y realizar la tarea que se le ha solicitado.

Además de las órdenes presentes y sus respectivos botones, se añadieron cuatro opciones más generales para poder responder a un mayor número de casos de uso:

- **Detener bucle**. Este botón enviará un mensaje al servidor que le indicará que se desea detener un bucle. El resultado de esto será que el servidor terminará la transición entre estados que estaba ejecutando en ese momento y no continuará enviando órdenes DMX al controlador. Aunque útil, es ligeramente impredecible, pues dependiendo del momento del bucle en el que se pulse el resultado será distinto, quedando los aparatos en el último estado que recibieron. Es por ello que en muchos casos se recomienda hacer uso de otra orden DMX (por ejemplo, se desarrolló una orden llamada "reset" que devolvía todos los canales al valor 0), que también interrumpirá el bucle y cuyo estado final es el mismo en todos los casos.
- **Validar JSON**. Este botón solicitará un archivo JSON al ser pulsado, y lo enviará al servidor. En vez de ejecutarlo como si fuera una orden, el servidor web responderá si el archivo cumple con el esquema definido para el tipo de orden que sea. Es útil porque no interrumpe otras órdenes ni comienza a ejecutarla en el caso de cumplir el esquema, lo cual es importante sobre todo ante la ausencia de una aplicación de alto nivel que asegure el correcto formato del archivo JSON (por el momento estos archivos se escriben manualmente, aunque se sigue considerando preferible a la tecnología utilizada anteriormente).
- **Enviar JSON personalizado**. Este botón es similar al anterior, pues el navegador solicita un archivo JSON para ser enviado al servidor. Una vez que se ha comprobado que cumple con el esquema, el servidor lo ejecutará sin guardar el archivo en su propio directorio (es decir, la orden se ejecutará una vez y luego habrá que mandar el JSON de nuevo). Su utilidad es limitada

en un concurso en directo, pero sirve para comprobar que el comportamiento de una orden determinada es el deseado antes de guardarlo en el servidor.

- **Añadir JSON prefabricado**. El comportamiento de este botón es idéntico al mencionado anteriormente, pero con la diferencia de que esta vez el servidor sí guardará el archivo en su lado. De esta manera, el archivo enviado se podrá activar como el resto de órdenes prefabricadas, desde la sección principal de la página. Este botón sirve también para sobrescribir órdenes ya existentes.

4.2.4. Últimas consideraciones

Merece la pena insistir en que los cuatro botones mencionados anteriormente no tienen un comportamiento especialmente dinámico o que permita una interacción rápida (por ejemplo, tres de ellos requieren la búsqueda de un archivo en el sistema de directorios del ordenador). Esto no incumple los requisitos derivados de la celebración en directo de los concursos, ya que su uso no está pensado para hacerse durante el propio directo sino para los ensayos o fases de preparación del concurso, en los que se tiene más tiempo para comprobar la información y no se requiere tanto dinamismo (aunque la página debe mantenerse en una velocidad igualmente razonable, pues si fuera demasiado lenta afectaría a estas fases de la preparación del evento).

Otro detalle notable es que no se ha implementado una manera de borrar archivos prefabricados. Aunque la solución habría sido bastante simple (bastaría con añadir un botón similar al de información que borrase el prefabricado), desde la empresa se decidió que no sería necesario, pues pueden simplemente sobrescribir los archivos necesarios o utilizar la línea de comandos de la Raspberry Pi de forma remota a través de SSH (*Secure SHell*) para borrarlos directamente. Curiosamente, esto resalta también lo mencionado en el párrafo anterior, pues muy difícilmente sería necesario borrar un archivo durante la celebración de un concurso, por lo que no se requiere de la velocidad que otorgaría la interfaz web comparada con el uso de SSH.

Una última consideración es que se puede comprobar la presencia de una rama aparte, en la que se han hecho tres *commits* antes de fusionarla de nuevo con la original. Esto fue debido a que se hizo una reestructuración de los ficheros y configuración del repositorio, y se llevó a cabo desde otra rama para evitar poder hacer algún cambio erróneo irreversible. En esta rama se ordenaron los directorios y ficheros del proyecto, se añadió un archivo *.gitignore* para ignorar ciertos archivos que no debían seguirse desde el punto de vista del repositorio (archivos sobre el entorno virtual, temporales, etcétera) y se añadió un fichero con los requisitos del programa, de forma que puedan instalarse automáticamente al descargar el repositorio en una máquina. Por último, se añadió un fichero README.md en el que se detallan los pasos para la instalación y ejecución del servidor, así como la instalación de las dependencias.

4.3. CONFIGURACIÓN DEL DISPOSITIVO ATEM

Esta última fase del proyecto podría considerarse más un trabajo de investigación y configuración que de desarrollo y programación, siendo el objetivo de esta el de poder hacer la producción audiovisual en directo de los concursos desde el propio dispositivo ATEM (un mezclador con múltiples fuentes de vídeo) en vez de depender de programas de terceros (como, por ejemplo, OBS). Esta decisión está basada en los múltiples problemas que se han encontrado con estos programas, llegando a considerarse demasiado poco fiables para el entorno de un directo profesional.

Por esta razón se ha considerado que, al utilizar el propio ATEM para este fin, se eliminan factores que podrían conducir a estos errores, simplificando la cadena de implicados en la producción del contenido audiovisual. Igualmente, para mayor precaución, se decidió también seguir usando estos programas de forma secundaria, como *backup* en el caso de fallo del ATEM.

Tras la primera reunión se acordaron los siguientes objetivos:

- Realizar diversas macros (serie secuencial de comandos) para poder cambiar de fuente de vídeo. Realizar ciertas animaciones y transiciones de tipo *Stinger* para estos cambios de fuente.
- Investigar la manera de poder activar dichas macros desde una botonera Stream Deck.
- Investigar la manera de incluir varias fuentes de vídeo en una única salida, pudiendo posicionarlas y escalarlas como fuera necesario.
- Incluir un fondo animado a dicha producción.
- Animar diversos *layouts* para poder mover las diversas fuentes de vídeo en la vista múltiple mencionada anteriormente.
- Escribir un breve manual en la documentación interna de la empresa, con ciertos pasos a seguir o consejos para los futuros encargados de configurar y utilizar estos dispositivos en los concursos.

El primer paso fue familiarizarse con el ATEM y con el software que el propio proveedor (BlackMagic) proporciona para su uso. Estas aplicaciones se conectan con el dispositivo y permiten un uso mucho más extenso de las funcionalidades que ofrece el aparato. Para conectarlo se puede usar un cable USB de tipo B o un cable Ethernet para configurarlo a través de una red LAN. Esta es la opción más recomendada, ya que el uso se vuelve más sencillo al no tener que estar constantemente accediendo al panel trasero del ATEM. Además, en el caso de este proyecto el uso de la red es obligatorio, ya que la aplicación encargada de configurar el Stream Deck se conecta al dispositivo a través de la red (este tema será tratado más adelante).

La manera principal de investigar y aprender sobre el uso de estos aparatos ha sido a través del propio manual disponible de forma gratuita en la página oficial de BlackMagic, y numerosos foros y tutoriales de la comunidad de usuarios. Una de las ventajas de este ATEM que pueden pasar desapercibidas a primera vista es que esta marca es una de las más utilizadas en la industria, por lo que, incluso si el fabricante no provee suficiente información o documentación, siempre habrá una gran cantidad de usuarios que han llegado a sus propias soluciones y están dispuestos a compartirlas.

Tras conectar las cámaras y la red LAN, se utilizó la primera aplicación (*ATEM Software Setup*) para configurar el dispositivo y actualizar sus *drivers*. Entre las opciones de configuración más importantes se encuentran las destinadas a especificar la dirección IP y la red a la que pertenece.

4.3.1. ATEM Software Control

Después de haberlo configurado se puede acceder a la siguiente aplicación, ATEM Software Control (ASC de ahora en adelante). Esta aplicación es la que se usa durante la gran mayoría del tiempo, y simula los numerosos botones del ATEM. En el caso concreto del modelo utilizado es especialmente útil ya que estos botones no están disponibles físicamente, por lo que este programa pasa de ser recomendable a obligatorio. En la figura 4.5 se puede ver cómo es la interfaz de la aplicación (es necesario matizar que en la ventana se ve nombrado un modelo distinto al utilizado en el trabajo. Esto se debe a que iba a ser el modelo utilizado originalmente, pero debido a problemas de funcionamiento se recurrió al mencionado en el capítulo de Metodología).

ASC cuenta con numerosas opciones y funcionalidades, divididas en secciones más generales que lo mantienen todo organizado de manera más lógica e intuitiva. Para el primer objetivo del trabajo, hay una herramienta que permite crear dichas macros, así como cambiar su nombre y descripción. La manera de crearlas es "grabando" los botones pulsados, que más tarde se pulsarán en el mismo orden durante su ejecución. También es importante destacar que estos botones se pulsan uno detrás de otro y sin esperas (por ejemplo, no sirve de nada esperar un minuto durante la grabación, ya que esa espera no se va a registrar), aunque ASC cuenta con la opción de añadir manualmente esperas y configurar su duración.

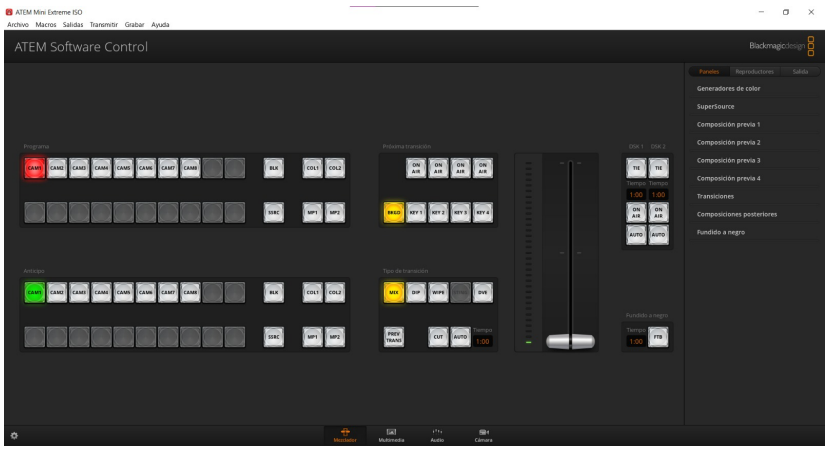


Figura 4.5: Interfaz de la aplicación ATEM Software Control

4.3.2. Vistas

Antes de comenzar con las animaciones es necesario elaborar un listado con las vistas que van a necesitarse, de manera que se pueda configurar todo de manera ordenada y siempre con un objetivo claro:

- **Vista 1**. Concursantes en grande, junto con el presentador y la pregunta.
- **Vista 2**. Presentador y pregunta.
- **Vista 3**. Concursantes y pregunta.
- **Vista 4**. Presentador y concursantes con el mismo tamaño, más la pregunta.
- **Vista 5**. Sólo presentador.
- **Vista 6**. Sólo concursantes.
- **Vista 7**. Sólo pregunta.

Como se puede notar, las cuatro primeras vistas incluyen varias fuentes de vídeo. Hay una vista, llamada *SuperSource* o SSC, que puede contener hasta cuatro entradas con el tamaño y posicionamiento que se quiera, lo que resuelve esta problemática.

Para la creación de los stingers se descargó un vídeo apropiado[2] y se editó cada fotograma para ser transparente en los casos que fueran necesarios. Por parte de ASC, las animaciones de este tipo están soportadas, por lo que sólo fue necesario importar los fotogramas editados (este programa no acepta vídeos puros, sino colecciones de *frames*) y configurar el momento de la animación en la que la pantalla queda completamente "cubierta", de forma que se haga la transición en ese momento. La ejecución de estas transiciones es muy sencilla, ya que basta con especificar la entrada a la que se va a transicionar en la escena de "preview" (aquella que no se ve en el directo y sirve para comprobar el estado final de una transición) y pulsar el botón de las transiciones de tipo *Stinger*.

También es de esta forma como se arregló el objetivo del fondo animado, ya que SSC permite definir un fondo que se verá en todos aquellos puntos que no haya una entrada ocultándolo. Si se añaden una serie de fotogramas puede configurarse para reproducirse en bucle, proporcionando el fondo animado.

Un punto destacable es el tratamiento de las vistas múltiples y cómo interactúan con estas transiciones. El mayor contratiempo es que SSC se trata como una entrada por sí misma, por lo que la interacción con las transiciones *Stinger* no es la adecuada. Si bien es cierto que es posible transicionar sobre la misma vista, los cambios entre distintas configuraciones del SSC son instantáneos, de manera que el *Stinger* no los tapa. La solución pasa por añadir una pausa en la macro, como se ha mencionado

[2] https://www.videezy.com/abstract/44865-liquid-transitions-pack

antes, con exactamente la misma duración que el tiempo que tarda el *Stinger* en ocultar la pantalla. De esta manera, para las vistas con SSC la transición no es únicamente a la vista, sino que a mitad de la transición se configura la posición y tamaño de las cuatro fuentes de vídeo.

Tras hacer las investigaciones y configuraciones necesarias, podemos ver que sólo con ASC se han resuelto tres de los seis puntos que se acordaron como objetivos. Los tres siguientes no podrán realizarse sobre la aplicación original, ya que se trata de opciones que no existen (o, al menos, no se pueden realizar a un ritmo razonable).

4.3.3. Animaciones en SuperSource

Además de las transiciones *Stinger*, se desea que haya animaciones para el caso de transiciones entre vistas múltiples. De esta manera, la cámara del público, por ejemplo, se movería y se haría más pequeña al cambiar de vista, lo cual proporciona una sensación de transición muy limpia y dinámica.

La manera de hacer este tipo de animaciones es también a través de las macros, pero se presentan varios problemas:

- Las macros no tienen soporte para este tipo de animaciones, por lo que los "movimientos"serían cambios instantáneos. La manera de arreglar esto sería utilizando las pausas mencionadas en los párrafos anteriores, de manera que no se definen dos estados, sino un gran número de ellos (uno por fotograma) y estableciendo una espera de un único *frame* entre cada uno.
- Las macros tampoco presentan ningún tipo de adaptabilidad. Por tanto, no es tan sencillo como reconocer la posición de las ventanas y moverlas de acuerdo al objetivo. Es necesario establecer un punto inicial, uno final y todos los intermedios, por lo que para las tres vistas múltiples será necesario hacer seis animaciones en total, de manera que queden cubiertas todas las posibilidades de vista inicial y final. Además, esto es claramente muy poco escalable.

La solución para el primer inconveniente pasa por YATM (*Your Animated Transition Macro*). Esta aplicación, que es de código abierto y se encuentra alojada en un repositorio de *GitHub* recoge los parámetros iniciales y finales de cada una de las cuatro entradas posibles en el SSC y permite establecer un tiempo para la animación. En pocas palabras, automatiza la tarea de intercalar esperas con pequeños movimientos para dar apariencia de movimiento. Gracias a esta aplicación se puede ahorrar mucho tiempo y esfuerzo, además de evitar el enorme riesgo de errores humanos.

El inconveniente de esta aplicación es que está hecha pensando en otro dispositivo ATEM, por lo que a la hora de escribir el archivo XML para poder importar la macro es necesario editar los campos que especifican la versión del ASC y el propio modelo de ATEM manualmente. Esto tampoco es completamente obligatorio, pues si se intenta importar sin editar el archivo el programa lo aceptará tras advertirnos de los posibles conflictos y problemas de compatibilidad que pueden surgir. No obstante, este aviso no es preocupante, ya que SSC y las órdenes especificadas en dichos archivos son completamente compatibles en ambos ATEMs.

Para el segundo problema el planteamiento ha pasado más bien por tratar de hacer lo más intuitiva posible dicha situación. No hay manera de hacerlo interactivo o con una mínima capacidad de respuesta a las distintas situaciones. Un programa que se encargara de hacerlo sobre la marcha sería innecesariamente complejo y consumiría mucho tiempo para el poco beneficio que otorgaría. Por lo tanto, se han hecho las seis animaciones y se han guardado las macros en el Stream Deck. Sin embargo, para evitar errores humanos y hacerlo más intuitivo, se ha utilizado la opción de añadir *feedback* en los botones, provocando un cambio de color que señale aquellas animaciones cuyo estado inicial es la vista múltiple actual. Por ejemplo, si la vista actual es la número uno se iluminarán las animaciones “1=>2” y “1=>3”. Esto también se aplica a las transiciones *Stinger*, ya que si se activa un cambio para pasar a la vista tres no tendría sentido no actualizar los botones (y reduciría mucho la usabilidad).

Figura 4.6: Ejemplo de conexión de todos los dispositivos al ATEM una vez configurados

4.3.4. BitFocus Companion

BitFocus Companion (BFC de ahora en adelante) se trata de una aplicación desarrollada para actuar como intermediario entre una botonera Stream Deck y un "switcher", un aparato usado para cambiar de cámaras y vistas (siendo el ATEM un dispositivo de este tipo). Esta aplicación es muy útil para este trabajo, ya que ahorra mucho tiempo y costes, siendo una herramienta bastante útil.

La manera de funcionar de esta aplicación es a través de un servidor web (al lanzar la aplicación permite configurar en qué interfaz debe escuchar, lo cual permite incluso que el servidor esté alojado en otra máquina distinta al ordenador que utiliza la aplicación como tal.

Una vez se ha lanzado el servidor, desde el navegador puede conectarse con el ATEM (está automáticamente soportado por BFC) especificando la dirección IP. El resto de configuración se hace automáticamente. Además, debe conectarse el Stream Deck a través de un puerto USB. Una vez se hayan cumplido estos pasos, la aplicación está preparada para usarse.

BFC cuenta con una sección dedicada por completo a configurar la botonera, de forma que, dependiendo del "switcher" que se esté usando, pueden colocarse ciertas instrucciones prefabricadas en cualquier botón del Stream Deck (por ejemplo, cambiar a la fuente número dos), incluyendo las macros que tenga el ATEM guardadas. Además, se pueden personalizar los botones (ya sea su aspecto visual o su comportamiento), lo cual permite mucha personalización.

El Stream Deck cuenta con 99 páginas, cada una con doce botones libres. Se decidió que lo más rápido e intuitivo sería colocar ciertos botones simples (como aquellos destinados a cambiar de fuente) en la primera página y comenzar a poner las macros a partir de la 99, continuando en sentido descendiente (ya que, si se está en la primera página y se sigue retrocediendo, se llega a la última directamente). En la página 99 se configuraron todas las macros correspondientes a las siete transiciones *Stinger*, mientras que en la 98 se colocaron aquellas destinadas a las animaciones entre vistas del SSC.

Como se puede comprobar en la figura 4.6, los únicos dispositivos necesarios son el ATEM, el Stream deck y un portátil u ordenador capaz de ejecutar Bitfocus. De nuevo, no está de más mencionar que este ATEM no es el modelo que se ha utilizado para hacer el trabajo, sino que se utilizó más adelante (una vez reparado) para las últimas comprobaciones y la realización de esta fotografía.

4.3.5. Manual de uso

Tras realizar todas las configuraciones e investigaciones necesarias, se ha escrito un breve manual básico en el *Notion* de la empresa.

En este documento se mencionan los puntos que se han tratado en este proyecto, así como enlazar

todas las páginas de interés (el manual del ATEM, las páginas de las aplicaciones externas utilizadas, etcétera).

También se han resumido alguno de los puntos que se han considerado más importantes del manual, para permitir una mayor rapidez y claridad a la hora de consultar información que puede ser usada más a menudo. Además, se han señalado algunas páginas concretas del manual de *BlackMagic*, como por ejemplo la que trata específicamente el caso de las transiciones de tipo *Stinger*.

CAPÍTULO 5
Conclusiones

A continuación se analizará si los resultados del proyecto han cumplido con los objetivos del mismo.

5.1. OBJETIVOS GENERALES

- **Los sistemas deberán ser fiables.** Para cumplir con los requisitos de fiabilidad, tanto en el visualizador de preguntas como en el servidor DMX se ha realizado un correcto tratamiento de errores, de manera que se evite un bloqueo completo en el caso de haber un fallo. Por otra parte, en el caso del ATEM se han utilizado únicamente configuraciones disponibles en el software original de la empresa proveedora, lo cual otorga ciertas garantías de seguridad y compatibilidad (aunque las animaciones se realicen a través de un software externo, su adaptación es sencilla y su aplicación se realiza siempre a través del software original). Además, en el caso de Bitfocus Companion, la aplicación es de código abierto y cuenta con una gran comunidad de usuarios y contribuidores. Esto da una sensación de seguridad, ya que en caso de ocurrir algún error se podrá recurrir a los foros o manuales disponibles para repararlo. Por último, se han realizado planes de contingencia para los casos en los que el ATEM deje de funcionar en directo (por ejemplo, utilizando OBS).
- **Las soluciones deberán ser usables.** El visualizador de preguntas no tiene uso por sí mismo, ya que lo que hace es presentar la información recibida a través de otros módulos. Sin embargo, es necesario considerar algunos aspectos de usabilidad que sí le afectan y cumple, como la velocidad de respuesta o un aspecto físico atractivo, conseguido a través del uso de elementos HTML y, sobre todo, la aplicación de diversos estilos a través de CSS. Por otro lado, el servidor DMX cuenta con un aspecto agradable y una gran cantidad de retroalimentación para el usuario gracias a las notificaciones de la biblioteca Toastr. Si bien es cierto que escribir los archivos JSON a mano puede ser un proceso tedioso y lento, en reuniones con la empresa se llegó a la conclusión de que seguía suponiendo una mejora con respecto al software utilizado anteriormente, además del hecho de que un programa completo para crear esos archivos podría suponer otro proyecto por si mismo. En último lugar, la solución para lanzar macros en el ATEM es intuitiva, mostrando el estado de la escena para evitar potenciales errores. No sólo eso, sino que cuenta con un manual que puede ser de gran ayuda a la hora de realizar un concurso real.
- **Los sistemas no deberán depender de la conexión a internet.** Ninguna de las soluciones funcionan con conexión a internet, aunque sí dependen de conexión a una red local respetando la arquitectura existente en el proyecto. Para cumplir con este objetivo, todas las bibliotecas utilizadas fueron descargadas de sus repositorios oficiales para poder ejecutarlas sin conexión, y más allá de esa cuestión ningún servicio depende de Internet.
- **Compatibilidad e integración.** El visualizador de preguntas anterior funcionaba como una aplicación de Unity alojada en la ventana de un navegador. En este sentido, el cambio ha sido completamente transparente al resto de la arquitectura, ya que se siguen recibiendo y procesando exactamente los mismos mensajes, con la única diferencia de que su tratamiento

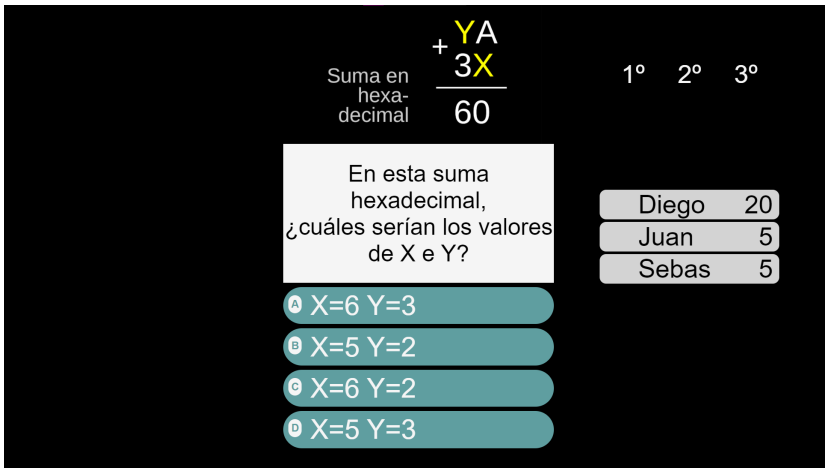


Figura 5.1: Aspecto final del visualizador de preguntas

ahora es en una aplicación web al uso y no en una aplicación de Unity. Por otro lado, aunque el cambio del servidor DMX fuera menos trivial, ocupa justamente el lugar de la aplicación anterior, y sigue conectando de manera similar con el controlador DMX que se encuentra conectado a la red LAN. En ambos casos se ha respetado el patrón de diseño "Observador" (en el caso del visualizador se trata de un "suscriptor" y en el caso del servidor DMX un "proveedor", aunque mientras se escribe este TFG ya se está actualizando para convertirlo también en suscriptor y que pueda reaccionar automáticamente a eventos del concurso). Respetando este patrón, mantener la compatibilidad ha sido sencillo al bastar con respetar el formato de los mensajes entrantes y salientes que existían antes. Por esta razón se considera que este cambio no ha supuesto ninguna consecuencia significativa en el resto del sistema. En último lugar, la configuración del ATEM se ha realizado utilizando tecnologías del propio fabricante o aplicaciones desarrolladas con estos dispositivos en mente, por lo que existe la certeza de que los problemas de compatibilidad serán inexistentes.

5.2. ARTEFACTOS RESULTANTES

5.2.1. Visualizador de preguntas

Los objetivos concretos del visualizador de preguntas se basaban en la búsqueda de una solución más flexible y eficiente que el sistema basado en Unity. Por ello se ha hecho uso de HTML, que permite la creación de "plantillas" de manera que la única preocupación a la hora de preparar el concurso sean las propias preguntas, sin ser necesario compilarlas de antemano en el visualizador que, en este caso, estará preparado para procesar cualquier entrada. Para completarlo, se ha utilizado CSS para mantener el aspecto visual del anterior visualizador (lo cual también permite editar el diseño de manera sencilla, tan sólo cambiando este archivo) y JavaScript para otorgar a la página de la interactividad propia de un concurso.

Otro objetivo era que se respetaran los estilos visuales, efectos y animaciones que existían anteriormente. Esto también se ha cumplido, llegando a utilizar exactamente los mismos archivos de sonido del sistema original, y asegurando que todos los eventos de importancia reciban su propia animación.

5.2.2. Servidor DMX

Para el servidor DMX el objetivo principal era la creación de una aplicación propia que permitiera controlar los dispositivos de este tipo de la manera más sencilla posible, además de poder definir efectos y animaciones predefinidas de manera sencilla.

Si bien es cierto que la solución de los archivos JSON no es precisamente una solución de alto

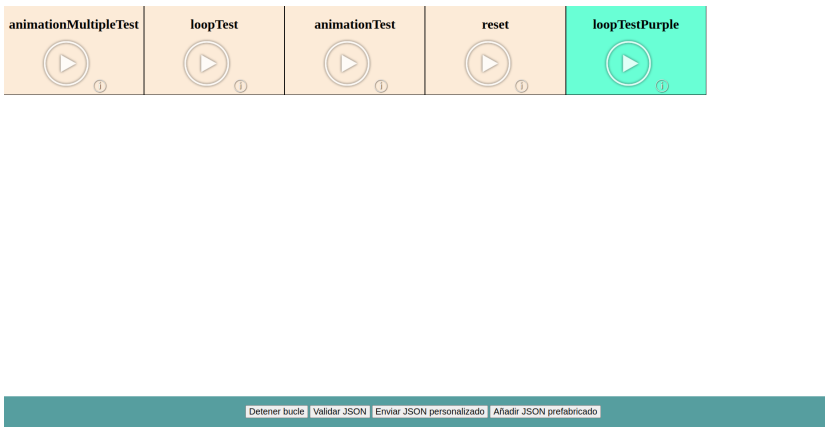


Figura 5.2: Aspecto final de la interfaz del servidor DMX

nivel al requerir de los valores concretos para cada canal, se consideró que la aplicación satisfacía el objetivo de la misma, ya que el control de los dispositivos una vez se han creado estas animaciones predefinidas es bastante sencillo e intuitivo. Estos archivos, además, muy seguramente tengan que crearse una única vez, quedándose guardados para los eventos siguientes y necesitando cambios tan sólo en el caso de cambiar de dispositivos DMX. La capacidad de leer estos archivos y la definición de esquemas concretos abre también la puerta a una aplicación de mayor nivel que pueda escribirlos automáticamente, por lo que se cumple el objetivo de compatibilidad.

Teniendo esto en cuenta, se ha podido hacer una interfaz de usuario sencilla e intuitiva gracias, de nuevo, a la combinación de HTML, CSS y JavaScript. Esto también provoca una gran libertad, pues puede utilizarse la aplicación desde cualquier dispositivo que cuente con un navegador compatible con las últimas versiones de estos lenguajes.

5.2.3. Configuración de dispositivo ATEM

Estos objetivos se han cumplido al completo, realizando todas las configuraciones necesarias en el propio dispositivo y utilizando una aplicación de terceros ya existente para la conexión con la botonera. Se ha generado un archivo XML con todas las macros, de manera que su almacenamiento no dependa del ATEM, y en caso de avería o simplemente cambio de aparato puedan importarse de nuevo sin más esfuerzo.

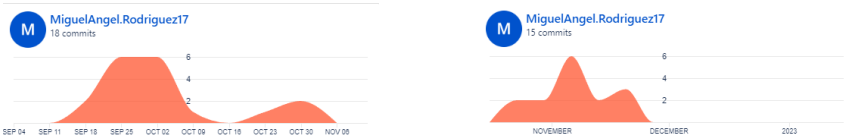
También se ha creado un manual de manera que en futuros proyectos este dispositivo pueda adaptarse a nuevos cambios o requisitos de forma rápida, asegurando que la solución seguirá siendo útil en el caso de cambiar de equipo o perder el estado guardado del dispositivo.

5.3. TRABAJO EN EL REPOSITORIO

Como se ha mencionado anteriormente, en los dos primeros subproyectos se ha trabajado sobre dos repositorios.

El primero ya estaba creado, y lo que se hizo fue trabajar en una nueva rama. Como se puede comprobar en los dos gráficos de la figura 5.3, el visualizador de preguntas cuenta con más commits y un trabajo más repartido en el tiempo. Esto se debe a que fue el desarrollo más complejo, y conllevó un largo periodo de distintos cambios y prototipado. El último pico que se ve en octubre se corresponde con el trabajo realizado para crear el ranking que, como se mencionó anteriormente, se añadió una vez estaba todo lo demás completado. A este pico también aportó la adición de las vistas específicas de los distintos elementos del visualizador, que se añadieron en el mismo periodo.

Por otra parte, el gran pico de actividad que se puede ver en el repositorio del servidor DMX se corresponde con el periodo en el que, además del desarrollo normal, se hicieron varios *commits*



(a) Repositorio del visualizador de preguntas **(b)** Repositorio del servidor DMX

Figura 5.3: Gráficos de los *commits* realizados en ambos repositorios)

a la rama separada que se ha mencionado, en la que se reestructuraron los paquetes y se limpió el repositorio de archivos innecesarios.

5.4. ARQUITECTURA RESULTANTE

Como se puede comprobar en la figura 5.4, un detalle notorio es que no hay prácticamente ningún cambio en el funcionamiento del sistema, y su diagrama se parece mucho al original (figura 1.2). Esto cumple también con los requisitos de integración, ya que al no haber realizado ningún cambio notorio en la aplicación, se puede asegurar la correcta integración y retrocompatibilidad con el resto de componentes.

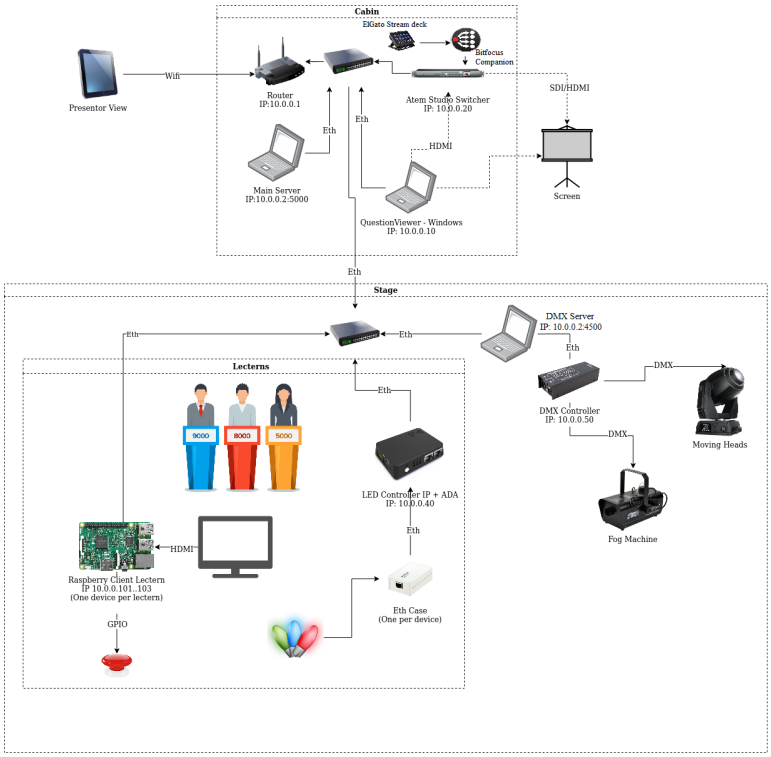


Figura 5.4: Arquitectura resultante

Los únicos cambios notables son la adición de un nuevo servidor, “DMX Server”, y de Bitfocus Companion junto con el Stream deck.

El servidor DMX actúa como intermediario entre la cabina y el controlador DMX. Este servidor es, como su nombre indica, el resultado del segundo subproyecto, que traducirá solicitudes HTTP a mensajes del protocolo ArtNet para que el controlador pueda llevarlas a cabo.

Se puede comprobar que su dirección IP es la misma (a excepción del puerto) que la del servidor principal. Esto no es un error, sino que el diagrama se ha colocado de esta manera con fines meramente ilustrativos. En realidad, el servidor DMX se ejecuta en la misma máquina que el servidor principal y el visualizador de preguntas, pero se encuentra en otro puerto, trabajando de manera completamente dependiente.

Por otra parte, Bitfocus Companion actuará como intermediario entre el Stream deck (que activará las macros necesarias) y el propio ATEM (que las ejecutará).

5.5. JUSTIFICACIÓN DE COMPETENCIAS ADQUIRIDAS

En el TFG se han aplicado las competencias correspondientes a la Tecnología Específica de *Tecnologías de la Información*:

TI1: *[Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones].* El desarrollo se ha realizado en estrecha comunicación con la empresa que necesitaba de estos servicios, y en todo momento se ha mantenido una retroalimentación en pos de asegurar el cumplimiento de sus requisitos y necesidades. Por tanto, se puede asegurar que estas soluciones cumplen adecuadamente con dichas necesidades, que se encuentran enumeradas en los párrafos anteriores.

TI2: *[Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados.].* Las constantes reuniones y retroalimentación por parte de la empresa han asegurado el cumplimiento constante de todos los requisitos de calidad y usabilidad, y no se han requerido inversiones por parte de la empresa más allá del salario de las prácticas. Todos los elementos utilizados eran ya propiedad de la empresa, por lo que lo único que se ha hecho ha sido refactorizar los elementos existentes o introducir nuevos elementos gratuitos y de código abierto. Por lo tanto, se considera que la mejoría del sistema en términos generales ha sido muy alta con un coste bajo.

TI3: *[Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas].* Se ha seguido una metodología de tipo RAD, de forma que un objetivo central del desarrollo ha sido la opinión de la empresa y los requisitos que pudieran tener. Así mismo, el desarrollo de las soluciones ha tenido la usabilidad, la ergonomía y la robustez de los programas, ya que es vital un uso sencillo y rápido en el contexto de uso del sistema (eventos en directo).

TI5: *[Capacidad para seleccionar, desplegar, integrar y gestionar sistemas de información que satisfagan las necesidades de la organización, con los criterios de coste y calidad identificados].* Aunque el sistema de visualización de preguntas no es nuevo, sino que se ha actualizado parcialmente, el subsistema correspondiente al servidor DMX no tenía precedentes en el sistema general, introduciendo un componente que antes no existía en el sistema, y siempre cumpliendo con los requisitos y sin provocar problemas de compatibilidad con el resto de componentes. Por tanto, toda esa plataforma ha sido creada, desplegada e integrada desde cero de forma satisfactoria.

Bibliografía

[1] Pervaiz K Ahmed and Mohammed Rafiq. *Internal marketing*. Routledge, 2013.

[2] Julian Berman. jsonschema. URL: https://python-jsonschema.readthedocs.io/en/stable/#about.

[3] Blackmagic. Manual de uso del ATEM 2 M/E Production Studio. URL: https://documents.blackmagicdesign.com/UserManuals/ATEM_Production_Studio_Switchers_Manual.pdf?_v=1663225210000.

[4] Chart.js. Chart.js. URL: https://www.chartjs.org/.

[5] Yu-kai Chou. *Actionable gamification: Beyond points, badges, and leaderboards*. Packt Publishing Ltd, 2019.

[6] CodeSeven. toastr. URL: https://github.com/CodeSeven/toastr.

[7] ECMA. ECMA-404, The JSON Data Interchange Syntax. https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf, 2017.

[8] ESTA. ANSI E1.11. https://tsp.esta.org/tsp/documents/docs/ANSI-ESTA_E1-11_2008R2018.pdf, 2018.

[9] Python Software Foundation. Python 3.11.1 documentation. URL: https://docs.python.org/3/.

[10] jazix. Your Animated Transition Macro. URL: https://github.com/jazix/YATM-YourAnimatedTransitionMacro/tree/V1.001.

[11] Javier Silvestre Madrid. Récords de audiencia en Antena 3 gracias al bote de 'Pasapalabra' de Pablo Díaz. URL:https://www.lavanguardia.com/television/20210702/7572901/audiencia-antena-3-bote-pasapalabra-pablo-diaz-supervivientes.html, 2021. Último acceso: feb. 2023.

[12] MDN. CSS: Cascading style sheets. URL: https://developer.mozilla.org/en-US/docs/Web/CSS.

[13] MDN. HTML: HyperText Markup Language. URL: https://developer.mozilla.org/en-US/docs/Web/HTML.

[14] MDN. Javascript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[15] Fakhroddin Noorbehbahani, Fereshteh Salehi, and Reza Jafar Zadeh. A systematic mapping study on gamification applied to e-marketing. *Journal of Research in Interactive Marketing*, 13(3):392–410, 2019.

[16] Pallets. Flask. URL: https://flask.palletsprojects.com/en/2.2.x/.

[17] Dayana Sofia Barros Pozo and Ricardo Patricio Medina Chicaiza. Gamificación: Reflexiones teóricas desde el enfoque empresarial. *Religación: Revista de Ciencias Sociales y Humanidades*, 6(27):197–210, 2021.

[18] Samuel Reed. Textfit. URL: https://github.com/STRML/textFit.

[19] Jesús Salido. Plantilla LaTeX para Trabajo Fin de Estudios en Ingeniería Informática - UCLM (España). Zenodo, DOI: https://doi.org/10.5281/zenodo.4561708, 2019.

[20] spacemanspiff2007. Pyartnet. URL: https://pypi.org/project/pyartnet/.