Universidad de Alcalá
Escuela Politécnica Superior

Grado en Sistemas de Información

Trabajo Fin de Grado

Desarrollo de aplicación SPA en REACT, apoyada en API RESTFull para la escucha de emisoras de radio online

Autor: Fernando Robledo Cabrerizo

Tutor/es: Ana Castillo Martínez

2019

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Sistemas de Información

Trabajo Fin de Grado
Desarrollo de aplicación SPA en REACT, apoyada en API RESTFull para la escucha de emisoras de radio online

Autor: Fernando Robledo Cabrerizo

Tutor/es: Ana Castillo Martínez

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

CALIFICACIÓN: ……..

FECHA: ……..

Agradecimientos

Agradezco a todas las personas que me han ayudado de una manera u otra a llegar hasta aquí, compañeros de estudios y de trabajo, así como a los profesionales docentes de la Universidad de Alcalá.

Un agradecimiento especial a mis padres que siempre me ayudaron y dieron apoyo en todo momento, también a Laura por su tenacidad y compañía a lo largo de este viaje.

Y por supuesto a mi tutora Ana Castillo, por la ayuda y facilidades prestadas durante la realización de este trabajo.

Índice Resumido

Índice Detallado

Índice de Imágenes

Índice de tablas

Resumen

En este proyecto se va a desarrollar una página web de tipo SPA que permita escuchar stream de emisoras de radio online. Usando como base la librería Javascript React y apoyada en un API REST desarrollado con PHP y el framework Symfony 4. También se desarrollará una librería que permita recuperar metadatos del audio de servidores de stream de tipo Shoutcast y ICECast.

Palabras Clave

React, Shoutcast, IceCast, Symfony, Redux

Abstract

In this project, we are going to develop a Single Page Application or SPA that allows user to listen online radio streams. The base of this project will be the React Javascript library and a REST API programmed with PHP and the Symfony 4 framework. We also create an audio metadata reading library for the Shoutcast and ICECAST stream servers.

Key words

React, Shoutcast, IceCast, Symfony, Redux

1. Introducción

En los últimos años, en el mundo del desarrollo web, las llamadas webapp o SPA (Single Page Application) o aplicaciones de navegador se han ido imponiendo como estándar de facto reemplazando así a los sitios web multi página o MPA.

La principal característica de estas aplicaciones reside en que las acciones de navegación no producen una recarga completa del contenido mostrado en el navegador, como sí ocurre en el modelo clásico. En las SPA tanto las acciones de navegación, como la carga de recursos, flujos de recepción y envío de datos, se producen de manera parcial, asíncrona y en un segundo plano, dando como resultado una experiencia de uso fluida y mucho más similar a la de una aplicación nativa, de ahí el nombre de aplicación web.

Este nuevo paradigma viene apoyado por las nuevas tecnologías que han ido trayendo de mano los navegadores. HTML5 [1], CSS3 [2] y las nuevas versiones de JavaScript [3] como ES6 [4], son la punta de lanza.

Estas nuevas características van siendo incluidas en los principales navegadores a buen ritmo, pero no todos dan la misma cobertura [5], siendo esta fragmentación [6] y heterogeneidad uno de los principales problemas a los que se enfrentan los desarrolladores a la hora de crear una aplicación web, pues en función de la versión y tipo de navegador usado, tendrá un set de funcionalidad diferente. Para mitigar este problema han surgido en el ecosistema de desarrollo diferentes librerías y frameworks que aseguran compatibilidad sobre gran número de navegadores, abstrayendo así al desarrollador de esa tarea, a día de hoy los principales son Angular [7], React [8] y VUE [9].

Estas aplicaciones, al estar alojadas en la nube y permitir un acceso ubicuo, se enfrentan a nuevos retos como por ejemplo la persistencia de datos, ya que el navegador solo permite almacenar un pequeño volumen de datos de forma local. Conjuntamente es normal que una parte importante de los datos que vaya a mostrar o manejar la aplicación sean de carácter dinámico, por lo que la mayor parte de las webapps, a excepción de las muy simples, trabajan conjuntamente con un back-end. Lo más habitual es que la comunicación se produzca bajo el estándar de API REST [10] (Representational State Transfer) y la transferencia de datos se realice en formato JSON [11] (Javascript Object Notation).

Este documento se encuentra dividido de la siguiente forma: En la sección dos, se encuentran los objetivos del trabajo a realizar y el marco de trabajo. La siguiente sección analiza el estado del arte donde se introducen algunos conceptos y tecnologías utilizadas para la realización del trabajo. La cuarta sección se centra en el desarrollo realizado en el proyecto. La siguiente sección muestra el presupuesto del trabajo desarrollado donde se muestra un análisis de los costes necesarios para la realización del proyecto. Para finalizar, la última sección muestra las conclusiones extraídas tras la realización de este trabajo.

2. Objetivo

El objetivo central de este trabajo es la realización de una web app, que ofrezca al usuario la funcionalidad de escuchar entre diferentes emisoras de radio online, usando un navegador web.

Para la realización del mismo nos apoyaremos en la librería Javascript REACT para la parte del front-end y PHP 7 [12] con el framework Symfony 4 [13] para la creación de un api REST como infraestructura back-end que de soporte al frontal web.

Para lograrlo se llevarán a cabo los siguientes objetivos durante la realización de este trabajo:

- Desarrollar una interfaz de usuario ligera y sencilla, basada en principios de diseño UX/UI, orientada a la usabilidad. Se aplicarán principios de diseño mobile first [14] y responsive [15], para la correcta visualización en diferentes tipos de dispositivos.

- Construir los diferentes elementos con un diseño modular y reutilizable basado en componentes.

- Estudio del protocolo Icecast [16] y Shoutcast [17] utilizado para la transmisión de streams de audio y desarrollo de una librería capaz de leer la meta-información transmitida como parte de los streams.

El API se diseñará bajo arquitectura stateless REST, usando mensajes JSON, las peticiones usarán los diferentes verbos HTTP y códigos de respuesta apropiados.

El código se desarrollará bajo los principios SOLID [18], clean code, DRY [19], utilizando los patrones de diseño que se consideren más apropiados para cada elemento.

Tanto el código de la parte frontal como del back-end dispondrá de pruebas, que garanticen el correcto funcionamiento de la aplicación.

En la imagen bajo estas líneas, se representa la arquitectura del sistema a crear, con sus distintos componentes y flujos de navegación, todo ello bajo modelo SPA [20].

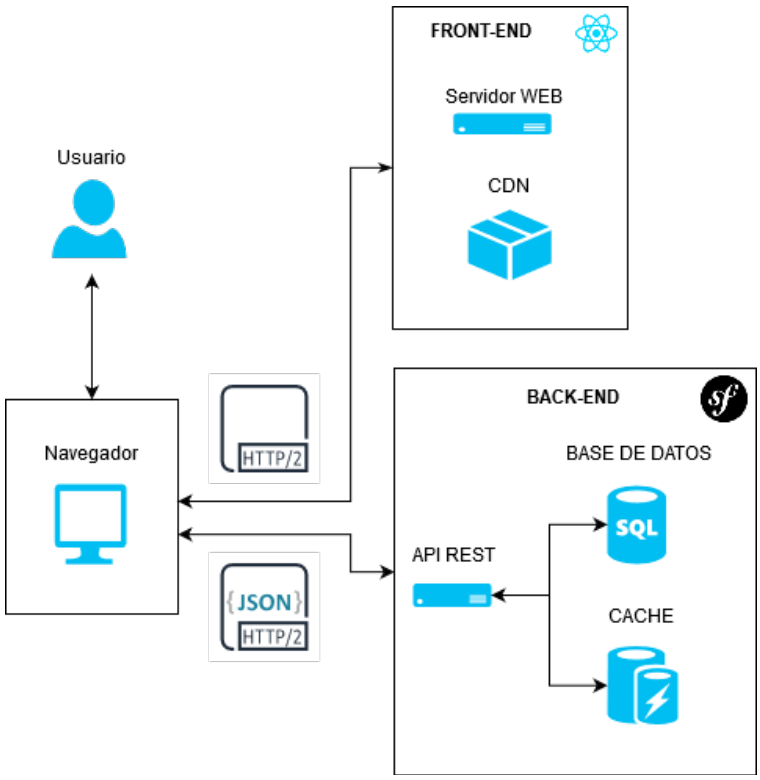

Imagen 1 - Arquitectura de nuestra aplicación SPA

3. Estado del arte

Para intentar obtener una visión lo más global y completa posible sobre el estado actual del desarrollo de aplicaciones web ha realizado un estudio completo desde diferentes puntos:

- En primer lugar, se analizan las arquitecturas web existentes y sus principales características.

- En segundo lugar, se analizará el estado de las tecnologías más demandadas en el mundo SPA, dando un repaso al ecosistema del lenguaje Javascript, viendo cuales son los principales frameworks y librerías junto a sus características y sus tendencias en el mercado.

- En tercer lugar, se realiza un benchmarking sobre las principales webs para la escucha de radio online, obteniendo datos que serán útiles para el desarrollo del proyecto.

- Por último, se realiza un repaso a las soluciones software para el streaming de audio en el navegador

3.1. Patrones de arquitectura Web

En el mundo del desarrollo web, el patrón SPA surgido en los primeros años del siglo XXI y popularizado en la última década, ha ido desplazando al clásico MPA, este nuevo modelo introduce cambios y desafíos técnicos para los desarrolladores. Ambos modelos no son excluyentes y es habitual encontrar sistemas que integran ambos SPA y MPA, o que realizan la evolución entre un modelo u otro de forma orgánica.

En los siguientes puntos se explican las principales características de cada uno de ellos.

3.1.1. Arquitectura MPA

Esta arquitectura ha sido la forma clásica de construir sitios y aplicaciones web, en este modelo el cliente solicita cada una de las páginas al servidor. Podemos ver un esquema de su funcionamiento en la imagen dos.

Las web multi página, en adelante MPA, han sido por muchos años las reinas en el desarrollo web. Bajo arquitectura MPA cada página de una web es recuperada del servidor y repintada por el navegador. Esta arquitectura plantea menos complejidad en la gestión de eventos generados por el usuario, su funcionamiento usa el navegador tal y como fue diseñado inicialmente, dejándole a él la responsabilidad de gestionar los eventos asociados a la navegación, las operaciones con ficheros, la comunicación con el servidor, etc.

Otra ventaja de las MPA es la gestión de motores de búsqueda o SEO, bajo modelo MPA la página se envía ya renderizada por el servidor, por lo que estos motores de búsqueda pueden interpretarla e indexarla correctamente.

Las MPA por sí mismas, no exigen soporte de Javascript en el navegador, lo que puede ser una ventaja para páginas web sencillas o que se ejecuten en entornos con limitaciones, en este modelo el navegador recibe código HTML generado por el servidor web, que puede ser

representado directamente. A pesar de ello, hoy día se presupone que las aplicaciones web serán ejecutadas en un navegador con soporte para Javascript.

En el modelo MPA, es habitual encontrar un mayor acoplamiento entre las partes que componen un sistema web, siendo frecuente en arquitecturas web monolíticas, en las que todas las partes de una aplicación web conviven dentro del mismo proyecto.

Un punto negativo en esta arquitectura es la velocidad. Esto se debe principalmente a que, en cada navegación, tendremos que iniciar la conexión y recuperar las páginas y los recursos asociados desde el servidor, por ello la ventana del navegador tendrá que ser repintada por completo, con lo que la percepción de velocidad será menor y menos cercana a la de aplicaciones nativas.

Con la popularización de las XMLHttpRequest o peticiones AJAX [21], y el imparable crecimiento del ecosistema Javascript, la arquitectura MPA está siendo dejada atrás y siendo considerada obsoleta.

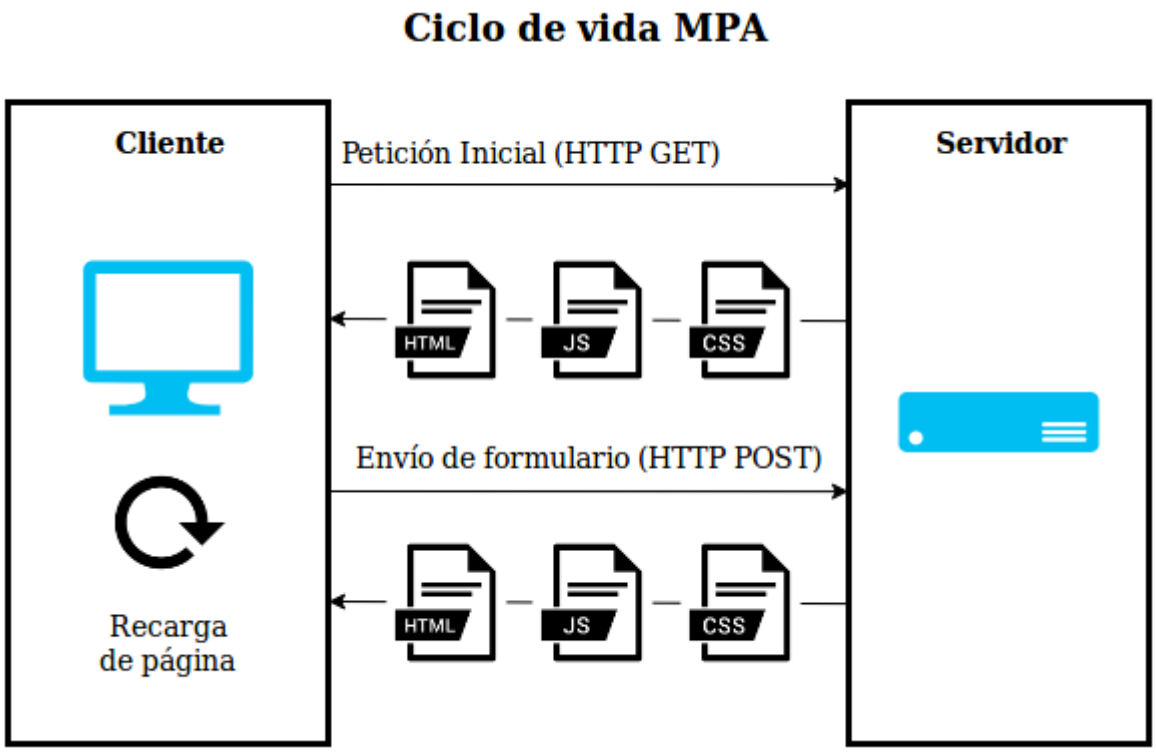

Imagen 2 - Ciclo de vida aplicación MPA

3.1.2. Arquitectura SPA

La arquitectura SPA en contraposición a la MPA clásica, descarga la aplicación completa durante primera navegación a la página web, realizando el resto de las cargas y operaciones de datos, de manera asíncrona. El foco principal de esta arquitectura es mejorar la experiencia de navegación del usuario haciéndola más cercana a la que ofrecen las aplicaciones nativas. En la imagen tres, se puede ver un diagrama del ciclo de vida típico en una web SPA.

La preferencia por las webs de tipo SPA viene dada por varios factores.

Un punto clave es la velocidad. En arquitectura SPA, a excepción de la carga inicial en la que se han de descargar todos los recursos principales de la aplicación y realizar el proceso de ensamblaje de la página en el navegador, lo cual suele resultar más pesado que en sus homólogos SPA, el resto de las acciones o navegación solo causa una recarga parcial de los recursos o zonas de la página web afectadas, como resultado se obtiene una experiencia más fluida.

Esta arquitectura permite un bajo acoplamiento [22], con una mejor separación de las capas que componen la misma, pudiendo tratar las otras partes del sistema como cajas negras, la aplicación tan solo necesita conocer las interfaces para poder interactuar con ellas.

Otra característica de las SPA es que, dada su naturaleza, para su carga tan solo necesitan ficheros estáticos, que son ejecutados en el navegador, y estos pueden servirse desde un CDN [23] con un menor coste y más fácilmente gestionables en caché. Además, existen herramientas avanzadas que analizan estos ficheros estáticos, con código Javascript y CSS, y los dividen en ficheros parciales llamados chunks, lo cual acelera aún más la carga.

A diferencia que las MPA las SPA una vez iniciadas toman el control de todos los eventos que suceden en el navegador y son responsables de responder y gestionar todos ellos, como navegación, renderizado y actualización de contenido, comunicación con sistemas back-end, operaciones de carga y descarga de ficheros, etc.

Uno de los mayores retos a los que se enfrentan las SPA son el SEO y la analítica. Los motores SEO no son capaces de interpretar correctamente las aplicaciones SPA, ya que el Document Object Model, es construido en tiempo de ejecución en el navegador, haciendo que las páginas no se indexen correctamente. El Document Object Model más conocido como DOM [24], es como se conoce a la interfaz estándar usada por los navegadores a la hora de procesar y representar internamente documentos HTML o XML, esta interfaz permite interactuar con el documento para su consulta o modificación, el documento se representa en forma de árbol, siendo cada elemento un nodo de este. La analítica plantea problemas a la hora de capturar las interacciones del usuario, en una página SPA es el programador el encargado de disparar los eventos lo que hace que la mayoría de las integraciones con librerías de analítica de terceros, no sean instalar y listo. Para mitigar este problema junto con el de la velocidad del primer renderizado de existen herramientas SSR, que permiten enviar una versión pre-renderizada al cliente en la primera petición, estas herramientas no son triviales de instalar y plantean retos para el desarrollo cuanto mayor sea la complejidad de la aplicación web.

Un factor importante es que las SPA dependen totalmente del motor Javascript del navegador en el que son ejecutadas y el soporte de versión de este. Esto es crucial, ya que no todos los navegadores incorporan las últimas funcionalidades de Javascript a igual ritmo, por suerte existen herramientas de software como babel para facilitar el desarrollo.

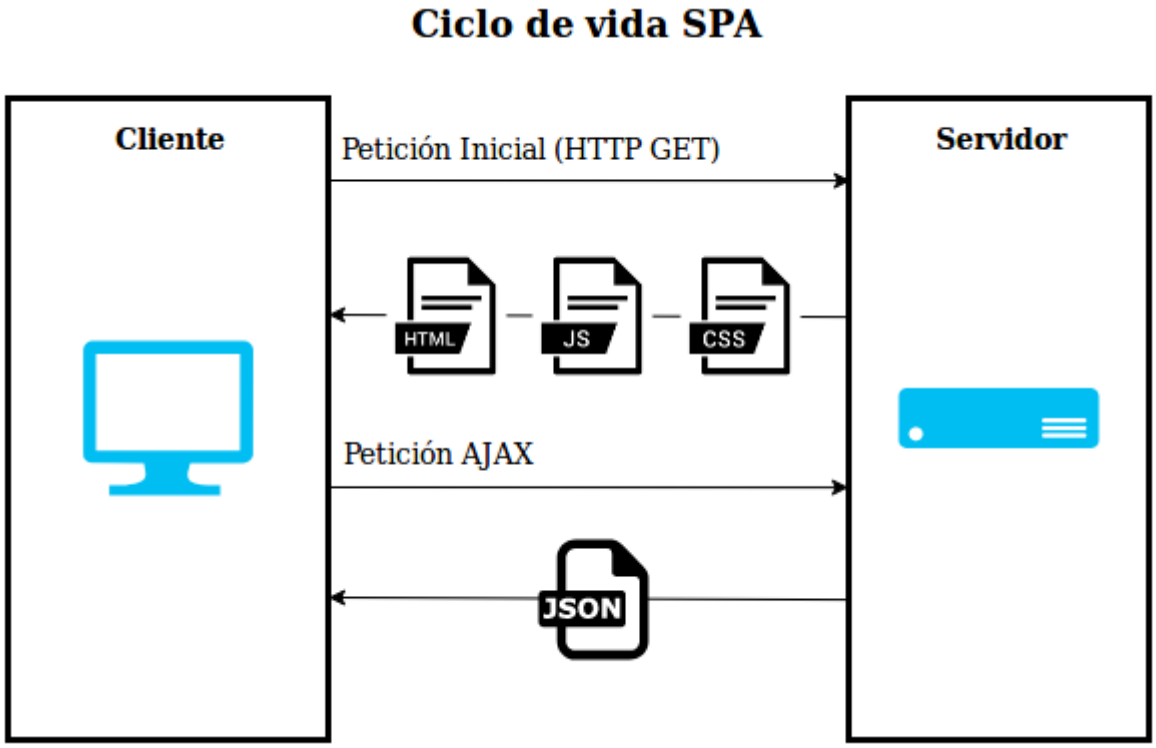

Imagen 3 - Ciclo de vida aplicación SPA

3.2. Tecnologías más demandadas

Para la creación de webs basadas en SPA a pesar de que todas ellas utilizan Javascript existen una gran infinidad de librerías y frameworks, que han ido variando y siendo sustituidos por otros más modernos, para acotar nos ceñiremos a los tres más populares la a hora de dar vida a este tipo de aplicaciones. A día de hoy, estos son Angular, React y VUE.

3.2.1. Angular

Angular es el nombre que recibe la segunda versión del framework Javascript para desarrollo web creado por Google, su primera versión se llamó AngularJS y fue lanzado en octubre del año 2010. La segunda versión lanzada en septiembre de 2016 y fue renombrada a Angular, supuso una reescritura completa del proyecto AngularJS para suplir las carencias del mismo y un cambio de enfoque.

3.2.2. React

React es una librería Javascript creada por Facebook y que fue lanzada en marzo del 2013, el origen de la librería fue un prototipo creada por un trabajador de Facebook llamado Jordan Walke, que más tarde se usaría para solucionar los problemas de gestión del servicio Facebook Ad, lo que daría origen a la primera versión de la librería.

3.2.3. Vue

Vue es un framework Javascript creado como proyecto personal por un trabajador de Google llamado Evan You, tras haber trabajado en la creación del framework AngularJS, fue publicada en febrero del 2014.

3.2.4. Características técnicas

Las 3 herramientas vistas anteriormente son capaces crear una webapp y comparten muchas de sus características, en este punto veremos cuáles son sus principales diferencias técnicas.

Una importante diferencia es la naturaleza de los mismos. Angular y Vue son frameworks mientras que React es una librería. Esto quiere decir que cuando usamos Angular o Vue estaremos obligados en mayor medida a trabajar de una determinada manera que si lo hiciéramos con React, dejando este último mayor libertad de elección al desarrollador. Angular además es especialmente estricto en este sentido por lo que presenta una curva de aprendizaje mayor. Esta flexibilidad que ofrece React, es un punto muy positivo ofreciendo una menor barrera de entrada.

Un punto clave a la hora de comprarlos es el rendimiento. Este punto no es sencillo de medir y depende en gran medida de las características de la aplicación a desarrollar. Los datos de velocidad que se obtienen tras enfrentar los tres a test de rendimiento sintéticos [25] sitúan como ganador a Vue seguido muy de cerca por React, estando en último lugar Angular. La diferencia principal de rendimiento viene marcada por la forma en la que estos frameworks interactúan con el DOM**.** Angular trabaja directamente con el DOM nativo del navegador, haciendo actualizaciones selectivas sobre las partes modificadas, por el contrario React y Vue utilizan un sistema de DOM virtual, este DOM virtual crea una copia en memoria, más simple y de menor peso, sobre la que se calculan los cambios a realizar, este sistema mejora en notablemente el rendimiento, reduciendo el número de operaciones finales efectuadas sobre el árbol de DOM nativo.

En cuanto a arquitectura, Angular [26] usa un sistema de módulos que distingue entre componentes, servicios y vistas, con la inyección de dependencias como eje central de la interoperabilidad y reusabilidad, a la hora de crear templates utiliza HTML con interpolación y un sistema propio de directivas. React se basa también en componentes, pero al contrario que react no hace distinciones entre estos, diferenciándolos solamente entre stateless o stateful, también implementa inversión de control [28] pero no dispone de un contenedor de servicios como en Angular, para el templating React ofrece la opción de usar JSX, un lenguaje mezcla de XML y Javascript y permite interpolar de forma sencilla Javascript. VUE [27] por su parte al igual que React basa su arquitectura en componentes y como sistema de templates utiliza HTML con interpolación y directivas propias, opcionalmente permite usar JSX.

A la hora de realizar el enlace de datos o Data Binding, la forma en que la vista y los modelos de datos se comunican. Angular y Vue utilizan un modelo de enlace doble, mientras que React usa uno simple. En el patrón doble, o two-way, se crea un enlace entre el valor del modelo y la representación del mismo en la vista, esto hace que cualquier modificación independientemente del origen, modelo o vista, actualice ambos valores. En el patrón simple, o one-way, solo hay una fuente de verdad, el modelo, los cambios de valor en la vista, en lugar de modificar el valor en el modelo, emitirán señales de notificación o eventos, y será responsabilidad del modelo actuar ante esas señales y decidir cuándo actualizar el valor.

Otro punto importante es el tamaño de las aplicaciones generadas y la propia librería. Todos ellos disponen de bundlers o "paquetizadores", que reducen y optimizan el tamaño del código final, y hacen uso de técnicas tree-shakin [29] que detectan código no usado y lo eliminan y consiguiendo reducir el peso del código generado.

Al crear código, React y Vue permiten hacerlo directamente en lenguaje Javascript, ofreciendo opcionalmente la opción de integrar herramientas como flow [30] o Typescript [31]. Por el contrario, Angular ha de ser programado directamente en Typescript, lo que ofrece otra barrera de entrada de cara a los desarrolladores que no conozcan este lenguaje. Typescript es un superset o ampliación del lenguaje Javascript, creado por Microsoft que añade características no nativas de Javascript, como el tipado fuerte, que ayuda a reducir errores durante la programación.

	Angular	**React**	**Vue**
Tipo	Framework	Librería	Framework
Creador	Google	Facebook	Evan You
Facilidad de aprendizaje	Media	Media / fácil	Fácil
Flexibilidad	Baja	Alta	Alta
Lenguaje	Typescript	Javascript / JSX	Javascript
Modelo DOM	Nativo	Virtual	Virtual
Año	Sept 2016	Marzo 2013	Feb 2014
Reducer	NgRX	Redux	Vuex

Tabla 1 - Comparativa entre Angular, React y Vue

Casos de uso:

- React: FB, Airbnb, Uber, Netflix, Instagram, Twitter, Pinterest, Reddit
- Angular: Google, Udemy, Invision, Mit
- Vue: 9gag, behance, Nintendo, Gitlab, Wizzair, Font Awesome, Laravel

3.2.5. Estado y tendencia del mercado

Actualmente Angular, React y VUE son las tecnologías que lideran el mundo de desarrollo Javascript para aplicaciones web. Para tener una visión de su evolución y adopción actual, se usarán los datos ofrecidos por la herramienta Google trends [32]. Estos datos nos ofrecerán una visión sobre la popularidad de estas tecnologías en el motor de búsqueda de Google.

Hemos incluido en la búsqueda la primera versión de Angular, llamada AngularJs, a pesar de que a día de hoy se considera obsoleta, para tener una mejor visión de conjunto sobre ese framework y la transición de una versión a otra.

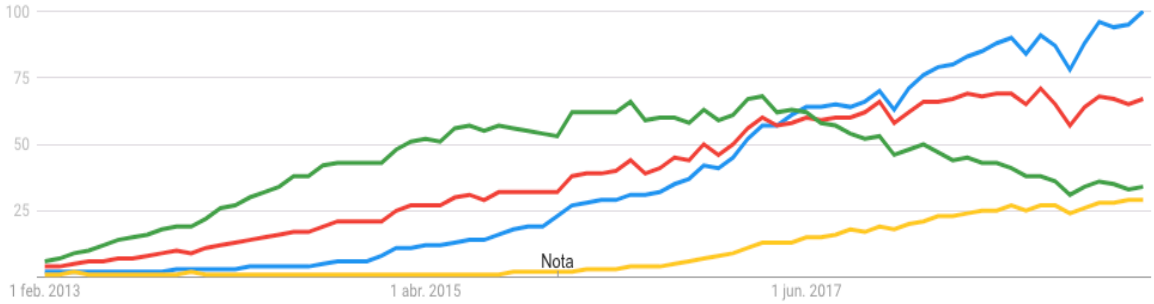

Imagen 4 - Tendencia de búsqueda en Google Trends

Porcentajes relativos de búsqueda

React	**100%**
Angular	**67%**
AngularJS	**34%**
VUE	**29%**

Tabla 2 - Porcentaje de búsqueda relativo en Google Trends

Analizando la gráfica, de la imagen 4, se observa como en abril del 2017 el interés React tomó la delantera al hasta entonces líder Angular (en sus dos versiones). A partir de ese punto se aprecia una clara tendencia al alza en el número de búsquedas relacionadas con React, a su vez las búsquedas para la primera versión de Angular comienzan a disminuir notablemente y se mantienen para la segunda versión.

Las tendencias en la gráfica de búsqueda, mantienen una relación con el orden cronológico de la aparición de cada uno de los frameworks, en primer lugar en octubre de 2010, Google lanzó la primera versión de Angular, llamada AngularJs, a esta le siguió la primera versión de React, lanzada por Facebook en mayo de 2013, el siguiente en aparecer en el mercado fue Vue, liberado por Evan You en febrero del 2014, por último en diciembre de 2015, Google lanzó la beta de la segunda versión del framework Angular, incompatible con la primera, la primera versión estable de este apareció en septiembre de 2016.

Para contrastar los datos de Google trends, podemos analizar métricas de sus repositorios de github, por tal en el cual se aloja el código de los tres proyectos y muestra el feedback y las interacciones por parte de la comunidad de desarrolladores hacia los proyectos. Tendremos en cuenta el número de estrellas y número de issues combinadas con el número de descargas a través de npm, que es la herramienta usada para la instalación de las librerías. Estos datos son un buen indicador del interés suscitado en la comunidad.

Proyecto	Estrellas	Issues	Pull Request	Descargas semanales (npm)
angular.js	59.523	386	75	448.246
angular	48.174	2.420	391	1.038.504
react	129.267	502	163	5.604.403
vue	138.781	201	80	985.022

Tabla 3 - Métricas de proyectos en Github y npm

Analizando los datos obtenidos de GitHub, resumidos en la tabla 3, atendiendo por las estrellas otorgadas, como representación del "cariño" de los desarrolladores a los diferentes proyectos, este dato sitúa en primera posición a Vue con 139.781, en segunda posición a react 129.267 y en última posición a Angular (v1 + v2) 107.697. Guiándonos por otras métricas como el número de "issues", vemos que Angular tiene un número mucho mayor que el resto, este dato es un arma de doble filo, y puede ser tanto un buen como mal indicador, ya que las "issues" pueden ser aportaciones de la comunidad ampliando o mejorando funcionalidad o reportes de fallos. En cuanto a descargas semanales a través de la herramienta npm, gana React con una enorme ventaja, seguido por Angular y Vue, con una pequeña diferencia de volumen entre estos dos últimos.

Usando la herramienta github-stars-history [33] podemos ver cuál ha sido la evolución de las estrellas otorgadas por los usuarios a los proyectos a lo largo del tiempo. En la gráfica de la imagen 5 vemos, al igual que en Google Trends, como a lo largo del año 2015 la nueva versión de angular reemplaza la anterior, y como la pendiente para React y Vue aumenta enormemente a partir del año 2016, superando a Angular. De esta tabla podemos concluir que actualmente el interés de la comunidad por React y VUE crece muy por encima del de Angular.

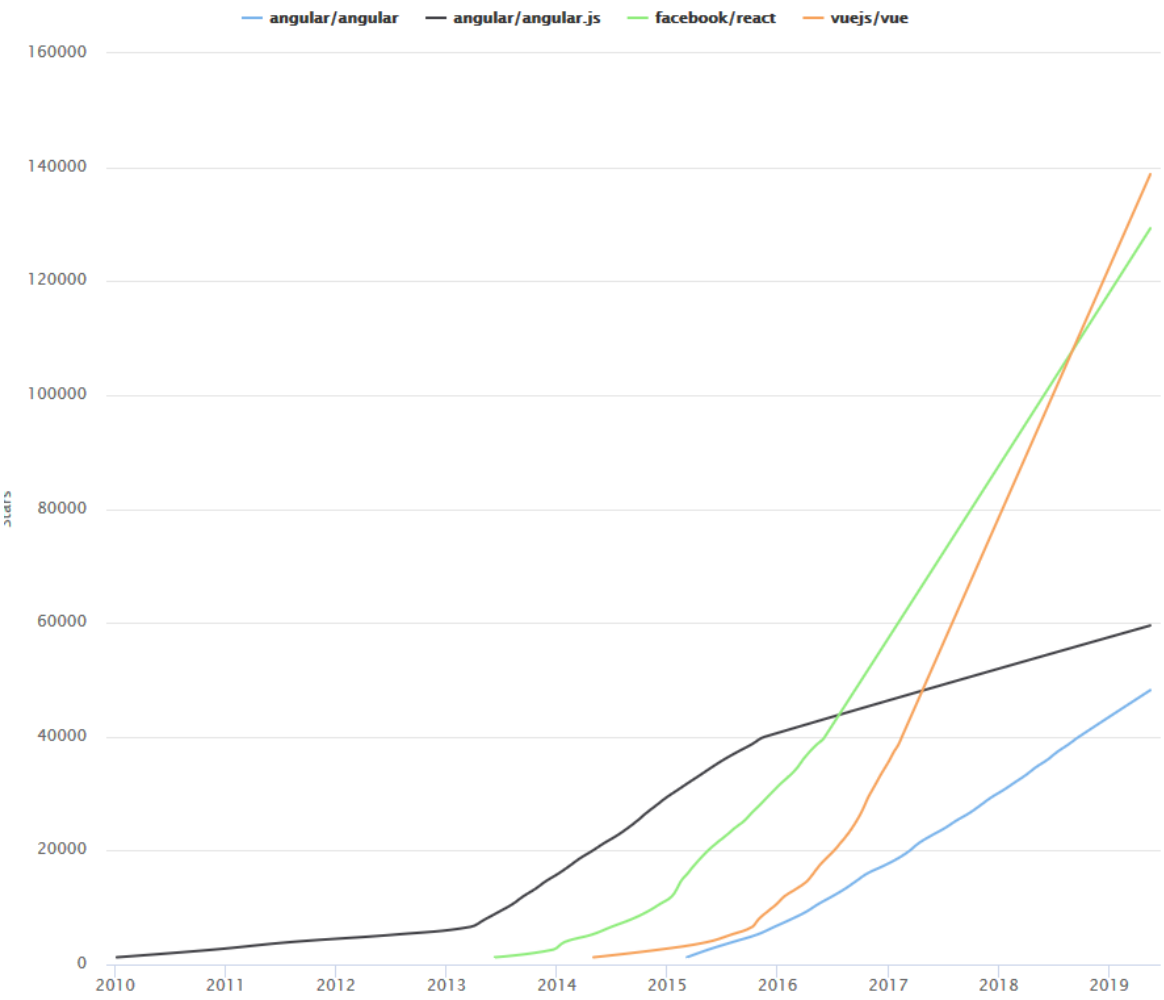

Imagen 5 - Evolución de estrellas otorgadas en Github

3.3. Benchmarking sobre webs de escucha radio online (nacionales)

A continuación, en la tabla 4, se comparan las principales webs para la escucha de radio online en España. Los sitios web han sido obtenidos del resultado de búsqueda orgánica en Google (incluir fecha), usando el término “escuchar radio online”.

	Responsive	Arquitectura	Stack	# Ranking
http://www.radio-espana.es/	No	MPA	jQuery	3232
https://emisora.org.es/	No	MPA	jQuery SoundManager	421
https://www.espana.fm/	No	SPA*	jQuery jQuery UI	10680
http://radio-espana.com/	Si	MPA	jQuery	8994

			SoundManager Drupal	
https://www.radio.es/	No	MPA	AngularJs Handlebars jQuery	1742
https://es.radioonline.fm/	No	SPA*	jQuery jQuery UI	13632
https://onlineradiobox.com/es/	Si	MPA	jQuery	2004 (brasil*)

Tabla 4 - Comparativa principales webs de escucha de radio online

De este análisis podemos obtener algunas conclusiones:

La gran mayoría de sitios no tiene en cuenta a los dispositivos móviles, no ofrecen una versión adaptada para la visualización en estos dispositivos, por ello la experiencia del usuario en estos dispositivos es muy deficiente.

La gran mayoría son aplicaciones MPA, que causan una recarga completa de la página cuando realizamos alguna acción, como por ejemplo cambiar de emisora. Algunas de ellas tienen un comportamiento más propio de las SPA, permitiendo cambiar de emisora sin tener que recargar la página, pero realmente carecen de muchas de las características que deberían ofrecer como SPA, por ejemplo, un sistema de routing, que mediante una URL permita volver al contexto actual.

Algo común en todas ellas es el uso generalizado de la librería jQuery, a día de hoy sigue siendo una gran herramienta, pero sin duda está en desuso y cada vez es más frecuente no encontrarla en proyectos con un stack tecnológico moderno.

3.4. Streaming de Audio en el navegador

Para poder escuchar emisoras de radio en el navegador son necesarias varias piezas trabajando en conjunto. Por un lado, necesitamos que el navegador sea capaz de reproducir el flujo de audio que recibe desde internet, a día de hoy cualquier navegador moderno soporta la reproducción y control de audio. Esto es posible gracias a la Web Audio API [34]. Esta especificación fue creada por el W3C en el año 2011, para dotar y estandarizar en el navegador de las funciones Javascript necesarias para el manejo avanzado de sonido. Más tarde el estándar HTML5 introdujo la etiqueta o tag <audio>, que también permite reproducir streams de audio sin tener que programar nada en Javascript, eso sí, ofreciendo solo un set de funcionalidad reducido respecto al ofrecido por Web Audio API.

Al otro lado del navegador son necesarias otras piezas de software, como un servidor de streaming, encargado de enviar los flujos de datos con audio codificado a los navegadores conectados. Estos servidores utilizan alguno de los protocolos de codificación de audio estándar que define el Web Audio API, siendo el más común el MP3.

Existen en el mercado diversas soluciones software para crear un servidor de streaming de audio, los más extendidos son Icecast [35], Shoutcast [36]. Estos servidores usan las cabeceras de

respuesta HTTP para ofrecernos metadatos sobre el audio que están transmitiendo, como el bitrate (calidad del audio), el género musical, el nombre de la emisora, etc. No todos los servidores de stream ofrecen estos datos, pero si algunos de ellos. También es habitual que, junto con los datos de audio, en el stream viaje incrustado otra información, como el nombre de la canción que está sonando. Mediante el uso de estándar Web Audio API, no es posible acceder a esta información directamente, por lo que necesitaremos ayuda de otros sistemas si queremos mostrar esta información en el navegador.

4. Desarrollo

En esta sección detallaremos como ha sido el proceso de desarrollo, con la etapa de prototipado e implementación, para cada una de las partes de la aplicación.

4.1. Prototipado

Previamente a la fase de implementación de la aplicación, se hará una fase de prototipado, cuyo objetivo es el de detectar y bocetar los elementos que serán necesarios más adelante durante la fase de implementación.

El prototipado se va a realizar en dos partes relacionadas pero independientes. La primera parte analizará los componentes necesarios en el lado del back-end de la aplicación. Se han de definir el modelo de datos y las APIs necesarias para alimentar la parte frontal.

En la segunda parte, se crearán los bocetos de las diferentes vistas que tendrá la aplicación, tanto en su versión móvil como de escritorio, incluyendo los elementos mínimos necesarios.

4.1.1. Prototipado del Backend

En la parte de back-end necesitamos un API REST de la que podamos recuperar la información referente a las emisoras de radio. Para ello definiremos las emisoras como recursos y les asignaremos el nombre de `station`, en la versión inicial tan solo ofreceremos endpoint para la lectura de los datos, mediante el verbo o método GET, más adelante se pueden incluir el resto de verbos HTTP [37] para habilitar el CRUD [38] sobre estos recursos.

URI	VERBO	Descripción
/sation	GET	Devuelve la lista completa de estaciones de radio
/station/{name}	GET	Devuelve la información específica de una emisora concreta identificada por el valor {name}

Tabla 5 - Puntos de conexión en nuestra API

Lo siguiente es definir el modelo de datos, este ha de recopilar toda la información relevante de los recursos de tipo emisora. Para la persistencia usaremos una base de datos relacional de tipo SQL.

El modelo estimado dispone de los siguientes campos

Campo	Tipo	Descripción
name	texto	Nombre de la emisora (ID de recurso externa)
url	texto	URL de streaming

frontname	texto	Nombre de la emisora a mostrar
type	texto	Tipo de emisora de streaming
metadata	booleano	Indica si el streaming contiene metadatos
image	texto	URL con la imagen de la emisora
genere	texto	Género musical

Tabla 6 - Modelo para almacenar información de emisoras

Con el modelo de datos y los endpoint definidos, podemos prototipar las respuestas devueltas por el API para cada uno de los endpoint.

Respuesta con lista de emisoras: **`GET /station`**

```
[
  {
    "name": string,
    "url": string",
    "frontname": string,
    "type": string,
    "metadata": bool,
    "image": string,
    "genere": string
  },
  {
    ...
  }
]
```

Información de emisora específica: **`GET /station/{name}`**

```
{
  "name": string,
  "url": string,
  "frontname": string,
  "type": string,
  "metadata": bool,
  "songname": string,
  "bitrate": int,
  "image": string,
  "genere": string
}
```

En la respuesta de emisora individual, aparecen dos nuevos campos que no aparecen en el listado global, estos son, ***songname*** y ***bitrate***, estos campos estarán solo disponibles en las emisoras que tengan habilitados los metadatos en su streaming, y nos informarán del nombre de la canción emitida y el bitrate del audio respectivamente. El contenido de esos datos será dinámico y no será recuperado de la capa de persistencia. En la parte de implementación veremos las alternativas de cómo resolver este punto y la opción tomada.

4.1.2. Prototipado del Front-end

En la parte frontal, vamos a bocetar las interfaces que se le mostrarán al usuario en la aplicación.

La aplicación cuenta con dos vistas principales, la galería y el reproductor.
En las imágenes podemos ver los bocetos para las versiones móviles.

La primera vista corresponde con la galería de emisoras, y como elementos principales incluye, en la parte superior el logotipo de la aplicación, y debajo de este el listado de emisoras disponible, este listado será formado por miniaturas de emisoras en formato vertical, e incluirá información relevante, como el nombre y el logotipo de la emisora. La galería organizada por una rejilla, el número de componentes mostrados en esta rejilla dependerá del ancho disponible en pantalla.

En la siguiente vista, se muestran los elementos de la vista de reproducción. De arriba abajo, esta vista incluirá un botón de acceso a menú, que veremos en la vista 3. A continuación el logotipo de la aplicación. Y por último el componente de reproductor, este incluirá toda la información relevante de la emisora, como logotipo, nombre, metadatos si están disponibles, y permitirá controlar la reproducción y el volumen.

La tercera vista, representa la segunda con el menú abierto, el menú será lateral, con origen en la parte izquierda de la pantalla, y ocupará el total de la altura disponible en pantalla. En este menú se mostrarán la galería de emisoras, en este caso el diseño de miniaturas será en modo apaisado.

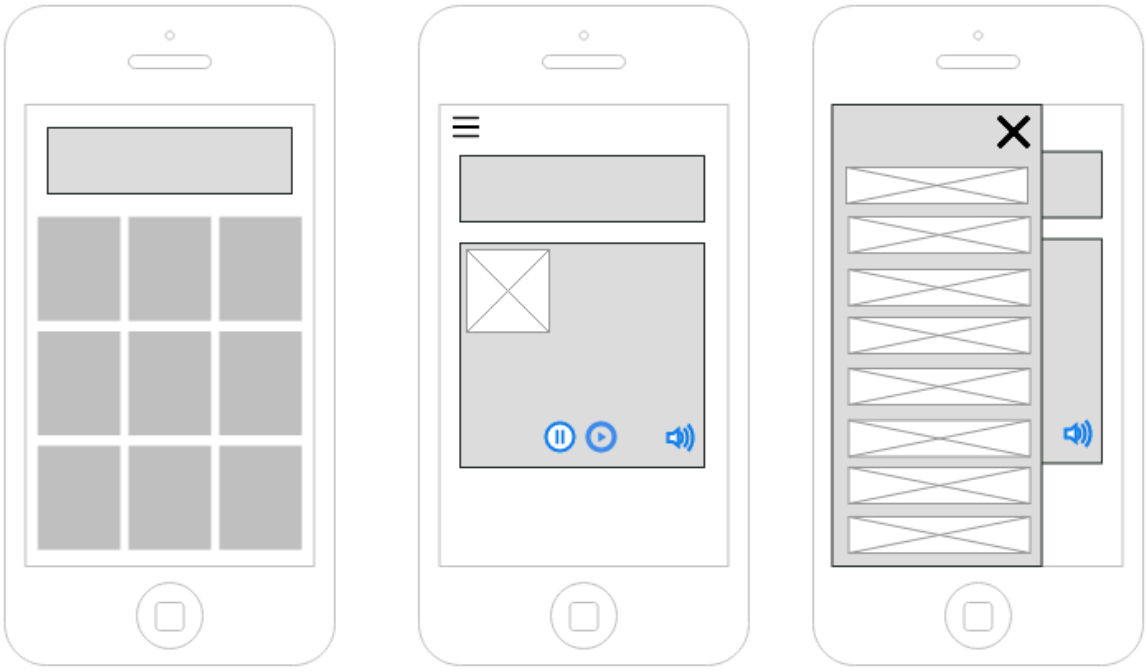

Imagen 6 - Bocetos de las diferentes vistas de la aplicación en versión móvil

Para la versión de escritorio, se usará un diseño muy similar, pero en este caso se eliminará la tercera vista de la versión móvil, ya que el espacio horizontal extra que otorga una resolución de escritorio permite mantener abierta de manera constante el listado de emisoras en el lado izquierdo en la vista del reproductor. Las vistas irán envueltas en un contenedor que limitará el tamaño horizontal máximo, el límite lo definiremos en la parte de implementación.

La vista de galería de emisoras, al igual que su versión móvil, usará la misma rejilla y diseño de miniaturas, mostrando hasta un máximo de 4 elementos por línea.

En la vista de reproducción, como ya se ha comentado, se divide en dos zonas, la izquierda y la derecha, la izquierda con proporción ¼ y la derecha con proporción ¾. En la parte izquierda se incluye un listado vertical de emisoras, la parte central y derecha de la pantalla lo ocupa el reproductor.

Imagen 7 - Bocetos de las diferentes vistas de la aplicación en versión escritorio

Como ya dijimos anteriormente usaremos un diseño “responsive” o sensible, que se adaptará dinámicamente a las diferentes resoluciones, permitiéndonos usar los mismos componentes para toda la gama de dispositivos.

4.2. Implementación de la aplicación

Con la fase de prototipado ya finalizada, tenemos una idea más o menos clara de que necesitamos desarrollar, en esta sección del documento realizaremos una memoria del proceso de implementación d

4.2.1. Implementación del Back-end

Dentro de este apartado se resume la implementación realizada para desarrollar el back-end de la aplicación y las decisiones tomadas durante el proceso.

En primer lugar, presentaremos el software usado, y las principales características que nos brinda.

En el siguiente apartado, hablaremos de la implementación real de nuestro prototipo con un resumen del flujo de funcionamiento.

Por último, un inciso en cómo se recupera la información de audio de determinados servidores de stream detallando el proceso de extracción de los metadatos.

4.2.1.1. Software usado

Para construir el software que dará vida al back-end de nuestra aplicación usaremos el lenguaje de programación PHP con orientación a objetos en su versión 7.2, junto con el framework PHP Symfony [39] en su versión 4.3. Gracias a las herramientas que incorpora este framework permite la creación de APIs REST de forma sólida, rápida y sencilla.

Symfony se basa en el modelo MVC [40]. Este modelo separa la lógica de la aplicación en tres capas, modelo, vista y controlador y de ahí recibe su nombre.

En nuestro proyecto, el modelo será el que definimos en la parte de prototipado, y lo usaremos como contenedor para la información de las emisoras.

La vista, al ser una API REST no es una vista clásica en sentido estricto, está vista será la representación de los datos devueltos por el API a los clientes externos y se hará en formato JSON, que es el estándar de facto para las API REST, para esta tarea nos ayudaremos del componente serializador [41] que incluye Symfony de cual hablaremos más adelante.

El controlador será el encargado de enrutar las peticiones y orquestar las operaciones necesarias para generar una respuesta.

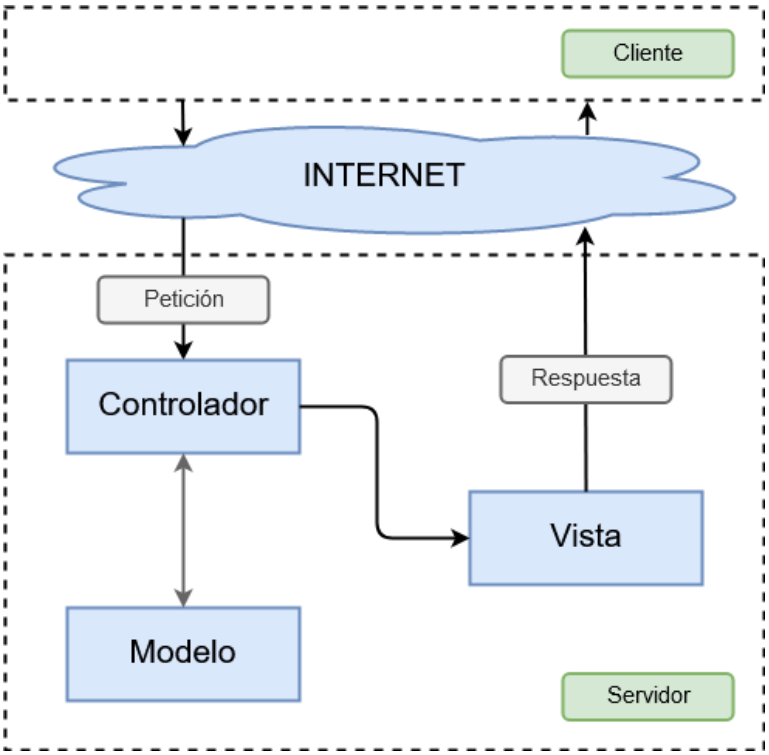

Imagen 8 - Esquema MVC

Symfony a su vez extiende este modelo MVC, añadiendo más capas de indirección, como por ejemplo el uso de un controlador frontal. Este controlador frontal, representa el punto único de entrada a la aplicación, y se encarga de gestionar todas las peticiones. Esta arquitectura

simplifica tareas como la gestión de peticiones, seguridad, etcétera. Por debajo de este controlador frontal, definiremos otros sub controladores o controladores lógicos, que contendrán métodos que serán los encargados de responder a cada una de las rutas de nuestra API, en Symfony se estos métodos se denominan *actions*.

Para trabajar con los modelos, vamos a usar Doctrine [42] que es un Mapeador Objeto Relacional u ORM, de hora en adelante, por sus siglas en inglés. Este ORM permite al desarrollador abstraerse de los detalles concretos implementación del y comunicación con el sistema de persistencia usado, creando así código agnóstico. Esto permite crear código desacoplado que no está ligado a ningún sistema de base de datos concreto, pudiendo migrar el día de mañana de uno a otra de manera sencilla, por ejemplo, de MySQL a Postgres o cualquiera de los motores soportados por Doctrine. Para permitir esto, Doctrine, al igual que otros ORM, tiene su propio lenguaje de consulta, denominado Doctrine Query Language o DQL [43], el cual usaremos en nuestra aplicación.

Para la definición de los modelos Doctrine basa sus modelos en clases anémicas PHP, con anotaciones sobre cada una de sus propiedades, las propiedades estarán encapsuladas mediante getters y setters.

En Doctrine los modelos no han de contener lógica, ya que se basa en el patrón Data Mapper [44], definido por Martin Fowler en el libro Patterns of Enterprise Application Architecture. Bajo este patrón las operaciones que incluyan lógica relacionada con la persistencia las haremos a través de un repositorio, siendo el modelo un simple contenedor para de datos cargados en memoria.

En la capa de representación o vista, la exposición de datos al mundo exterior se hará en formato JSON. Para esta tarea usaremos el componente serializador de Symfony. Este componente permite realizar la conversión de objetos de PHP a diferentes formatos de datos como JSON, XML o CVS. El serializador puede convertir datos en ambos sentidos. En el proceso de serialización existe una capa intermedia, en la que los datos son convertidos al formato de array relacional de PHP. El paso de objeto a array se denomina normalización y de array a formato de salida codificación. Symfony incluye de serie normalizadores para los tipos de objetos más comunes, y codificadores para los formatos de datos estándar más extendidos. Para nuestra aplicación, usaremos el proceso de serialización para convertir los objetos PHP devueltos por Doctrine, al formato de salida JSON.

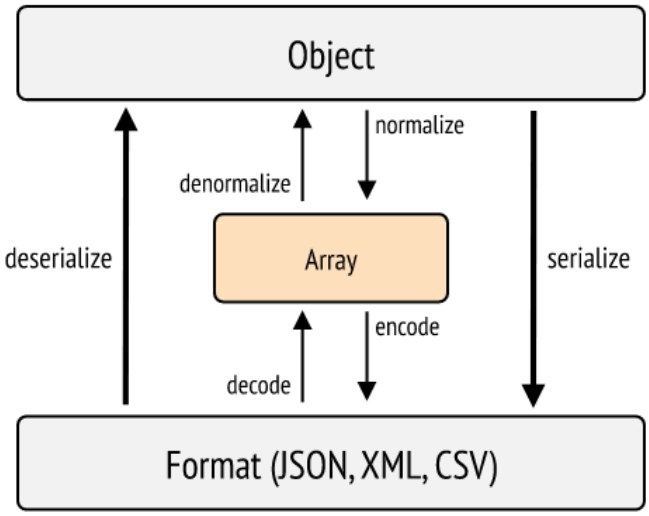

Imagen 9 - Diagrama de funcionamiento del serializador

Vamos a añadir también un sistema de caché a la aplicación. Esto ayudará a reducir el número de operaciones sobre la base de datos y el tiempo de respuesta de las peticiones.
Para esta tarea usaremos el software REDIS [45]. REDIS es un sistema de caché en memoria, y por ello permite unos tiempos de respuesta muy reducidos al recuperar datos almacenados en él.

Para la comunicación de nuestra aplicación con REDIS Symfony brinda soporte y ofrece un servicio de caché que hace uso de la interfaz estándar de PHP PSR-6.

El estándar PSR-6, definido por el PHP-Framework Interop Group o PHP-FIG [46], un conglomerado de personas y proyectos relevantes en el entorno de PHP que se encarga de definir estándares que maximicen la interoperabilidad entre distintos los diferentes frameworks y librerías PHP. El estándar PSR-6 [47] define una serie de interfaces que permiten, aislar el código, del sistema de caché específico y así poder cambiar este en un futuro sin tener que alterar nuestro código.

Como hemos visto, gracias al ORM y la interfaz PSR-6 podremos alterar las piezas externas de nuestra aplicación como el sistema de persistencia y de caché, simplemente con pequeños ajustes de configuración, sin tocar una línea de código.

Symfony dispone también de un sistema de inyección de dependencias, esto junto con el uso de interfaces estándar como PSR-3 para loggin o PSR-6 para caché, nos facilita desarrollar nuestro código siguiendo el principio de inversión de control o IoC, de esta manera todas las dependencias de nuestras clases son pasadas como parámetros, normalmente mediante su constructor. Este principio de programación permite desacoplar el código y simplifica su testeo.

A la hora de escribir el código PHP usaremos el estándar PSR-2, que presenta una guía de estilo de codificación con el objetivo mejorar la legibilidad de código, y la interoperabilidad entre desarrolladores.

Como gestor de dependencias para nuestro proyecto usaremos Composer [48], que es el gestor estándar de PHP, a través de él instalaremos todo el software anteriormente citado y mantendremos la lista de dependencias de nuestro proyecto.

4.2.1.2. Desarrollo del Back-end

Visto el software sobre el que se desarrollará la aplicación y las principales características de este, vamos a explicar cómo se han construido las diferentes partes del back-end.

Comenzamos creando los controladores, como punto de entrada a la aplicación. La aplicación dispondrá de un controlador con nombre `StationController`, este a su vez tendrá dos métodos o actions, uno para cada cada endpoint definido para el API en el apartado de prototipado.

Para estos métodos se usarán la nomenclatura estándar, definida en la guía de buenas prácticas de Symfony, formada por *verbo http + recurso*. Esto da lugar al método `cgetStation`, que devolverá el listado de emisoras, y a `getStation`, que devolverá información de una estación concreta. El carácter '*c*' delante del nombre del primer método, nos indica que nos va a devolver una colección de objetos del tipo *station*.

Definiremos las rutas que ya elegimos en la etapa de prototipado usando anotaciones en cada uno de los métodos creados en el controlador. En PHP las anotaciones son comentarios incluidos dentro del bloque de documentación de una clase, método o propiedad.

A la hora de implementar el modelo, Doctrine trabaja con anotaciones, las usaremos sobre las propiedades de nuestra clase Station para definir la longitud y tipo de los campos. En los campos alfanuméricos usaremos una longitud máxima de 255 caracteres para todos ellos menos para el campo url, que será del doble, 512. Añadimos también un nuevo campo con nombre 'ID' de tipo numérico, que será la clave primaria o PK, este campo será de uso interno y no se expondrá en las respuestas públicas. Mediante anotaciones crearemos también un índice sobre el campo name, ya que este será el identificador externo del recurso en nuestra API y de esa forma aceleramos las búsquedas.

Campo	Tipo Doctrine	Longitud
id	integer	0-4294967295
name	string	255
url	string	512
frontname	string	255
type	string	255
metadata	booleano	N/A
image	string	255
genere	texto	255

Tabla 7 - Modelado del recurso emisora "station" en Doctrine

A la hora de programar la lógica vamos a usar la filosofía llamada *thin controller fat service*, en este paradigma se intenta mantener los controladores "delgados" de forma que tengan el mínimo de líneas posibles, y que al verlos con un simple vistazo quede claro la

función que realizan. La lógica de negocio la trasladamos el código a clases de servicios, estas clases serán independientes, reutilizables y más mantenibles al no estar acopladas, y podrán ser usadas desde un controlador o cualquier otro lugar. En nuestro caso, a estas clases de servicio las denominaremos Managers.

De esta forma crearemos un servicio con nombre *`StationManager`*, con los métodos públicos *`getStations()`* y *`getStation(name)`*, que serán llamados respectivamente por cada una de las acciones del controlador. El flujo dentro de estos métodos será el siguiente.

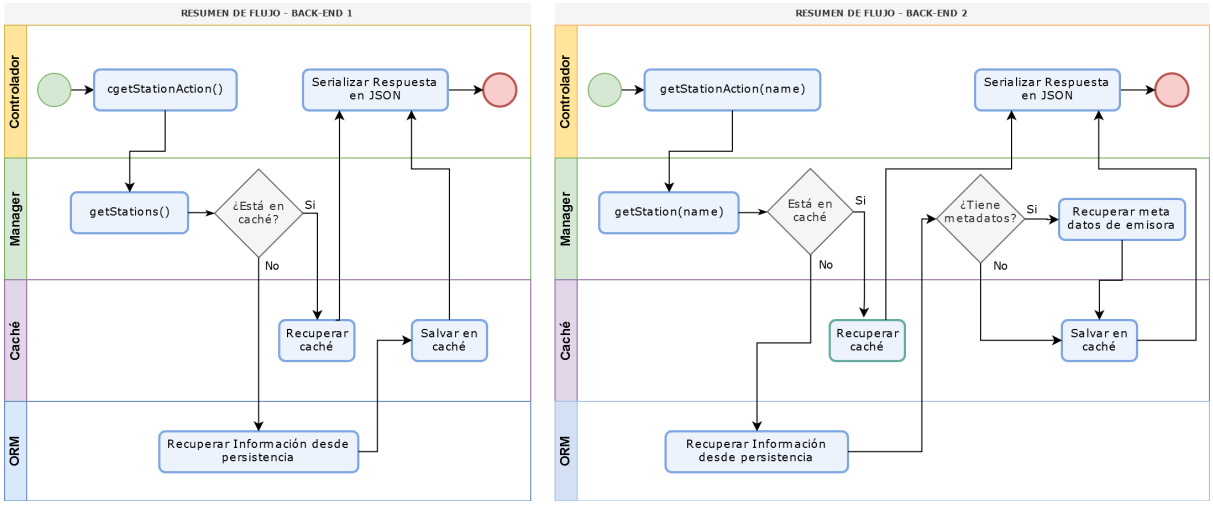

Imagen 10 - Diagrama de flujo para cada endpoint del API

Siguiendo el diagrama de flujo, vemos que el proceso es sencillo. Inicialmente el controlador se encarga de llamar al manager, el manager por su parte orquesta la persistencia y la caché, preguntando primero al servicio de caché si allí se encuentran los datos buscados, y en caso de no encontrarlos invocando al ORM para recuperarlos y salvarlos a caché para que la siguiente vez que se pregunten por esos datos, la respuesta sea servida rápidamente. En la etapa final el manager devuelve los Objetos con los datos solicitados y el controlador se encarga de serializarlos en el formato de salida correcto.

En la recuperación de una estación concreta, vemos una llamada extra, a un proceso que recupera los metadatos del stream de la emisora, si esta lo soporta. Para poder recuperar esos datos hemos de leer tanto las cabeceras HTTP como los datos en bruto del stream de audio. Sobre este proceso hablaremos en detalle en el siguiente apartado.

La gestión de errores se realiza mediante excepciones y su correspondiente mapeo a códigos de error HTTP. Por ejemplo, si intentamos recuperar información de una emisora no existente, la aplicación lanzará internamente una excepción de dominio que será interceptada y devolverá un mensaje de error en formato JSON y el código HTTP 404. De la misma manera sucederá con otras posibles excepciones. Para automatizar la gestión de excepciones estas implementarán la interface de Symfony HttpExceptionInterface.

4.2.1.3. Recuperación de la metainformación del stream

Como ya hemos comentado, algunos de los streams de audio contienen información embebida dentro del flujo de datos del stream.

En la aplicación daremos soporte a los metadatos devueltos por los servidores de tipo Shoutcast e Icecast [16], ya que los ambos servidores funcionan de manera muy similar y son los más usados. Shoutcast usa el protocolo Icy [17], este nombre se debe a que en sus inicios el proyecto se denominó ICYcast pasando más tarde a llamarse ShoutCast.

Para recuperar esta información debemos enviar una petición HTTP GET a la URL del stream, añadiendo la cabecera `Icy-MetaData: 1`. Si el servidor lo soporta y tiene habilitada la información, la respuesta incluirá todas o algunas de las siguientes cabeceras. En función de cuales detectemos sabremos si se trata de un servidor de tipo Icy o Icecast.

Icy	Icecast	Descripción
icy-pub	ice-public	Indica si la estación es pública
icy-name	ice-name	El nombre del stream
icy-notice	ice-description	Descripción del stream
icy-url	ice-url	URL del stream
icy-genere	ice-genere	Género del audio
icy-br	ice-bitrate	Bitrate
icy-metaint		Frecuencia de los metadatos*

Tabla 8 - Cabeceras para servidores Icy e Icecast

De estas cabeceras es de especial interés `icy-metaint`, este dato nos dará la información de la frecuencia con la que aparecen los metadatos embebidos en el flujo de audio. Este valor viene representado en bytes, por ejemplo un valor de 16000 indica que cada 16k la meta información es incluida en la señal de audio.

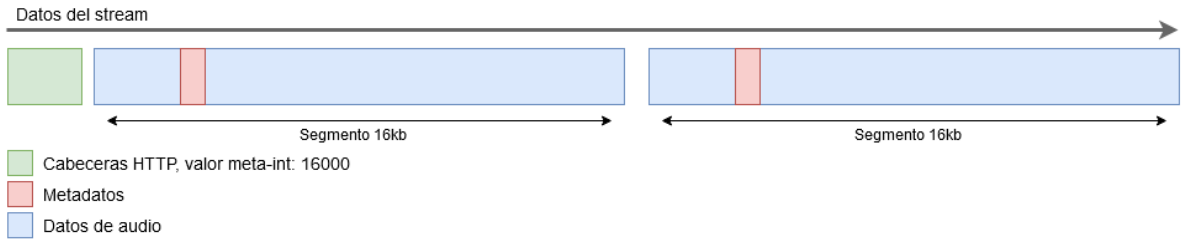

Imagen 11 - Diagrama de datos embebidos en el flujo de audio

En la imagen anterior se puede ver cómo cada fragmento de longitud N bytes indicados por el valor de `icy-metaint` se incluyen los metadatos. En verde se representa la información de cabeceras HTTP, en azul el flujo de datos, y en rojo la metainformación. Para localizar estos metadatos dentro de la fracción de audio delimitada por el valor de `icy-metaint`, existe una marca de inicio y final, con los literales que se observan en la siguiente imagen, dentro de las comillas simples tendremos la metainformación que necesitamos.

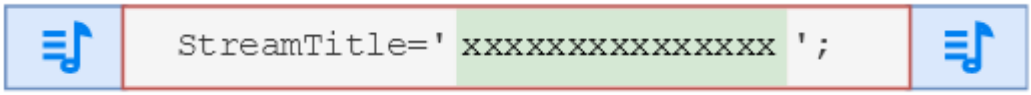

Imagen 12 - Etiquetas de inicio y fin dentro de los metadatos embebidos en el audio

Los datos embebidos normalmente corresponden con el nombre de la canción reproducida, aunque a veces son usadas para incluir otro tipo de información.

Sería lógico pensar en realizar este proceso en el cliente y trasladar esta tarea al front-end, liberando así al back-end de la carga extra, pero debido a las restricciones de seguridad que presenta la capa XMLHttpRequest API de los navegadores, no es posible realizar la lectura de las cabeceras de la respuesta y tampoco una lectura parcial del stream, para recuperar solamente el trozo de respuesta que contiene los metadatos. Tampoco es posible recuperar la información con el API Web Audio, mientras el stream es reproducido. Todo ello nos obliga a realizarlo en el back-end.

Se han buscado librerías que realicen esta tarea, y a pesar de que las hay para otros lenguajes como Python o Javascript (en lado de servidor), no se ha encontrado ninguna para PHP. Por ello hemos construido nuestra propia librería, como componente independiente del proyecto y añadido como dependencia, esto nos permite evolucionar ambos proyectos de manera independiente. La librería realiza todas las tareas anteriormente citadas, leyendo las cabeceras y los metadatos embebidos en aquellos stream que estén disponibles, mediante el uso de la librería curl [49].

4.2.2. Implementación del Front-end

En este punto, se resumen los pasos seguidos y el software usado para la creación de la parte frontal de la aplicación. Primero veremos el software y librerías usadas y cuáles son las características que ofrecen. En el siguiente punto, realizaremos la implementación del prototipo anteriormente definido.

4.2.2.1. Software usado

El front-end de nuestra aplicación se va a desarrollar usando como base la librería Javascript React en su versión 16.8. Además de React usaremos otras librerías compatibles que extienden su funcionalidad. Para manejar el estado de la aplicación haremos uso de la librería Redux [50]. En la implementación del enrutado y el renderizado en lado de servidor o SSR usaremos Next.js [51] en versión 9. Y para dar estilo a los componentes en React nos ayudaremos de la librería Styled Components [52].

React se basa en la creación y uso de componentes. En React cada componente maneja sus propiedades y estados de manera aislada. El flujo de comunicación está diseñado en un sólo sentido, de padres a hijos, en referencia a la jerarquía entre componentes. Para comunicar componentes entre sí, tendremos que hacer uso de funciones *callback* como propiedades del componente hijo o ayudarnos de manejadores de estado globales como Redux, del cual hablaremos más adelante.

Al usar un componente React esté sigue un flujo de ciclo de vida definido [53]. Es vital conocer y entender correctamente este ciclo antes de construir una aplicación. El ciclo de vida tiene cuatro fases, **montaje**, **actualización**, **desmontaje** y **gestión de errores**.

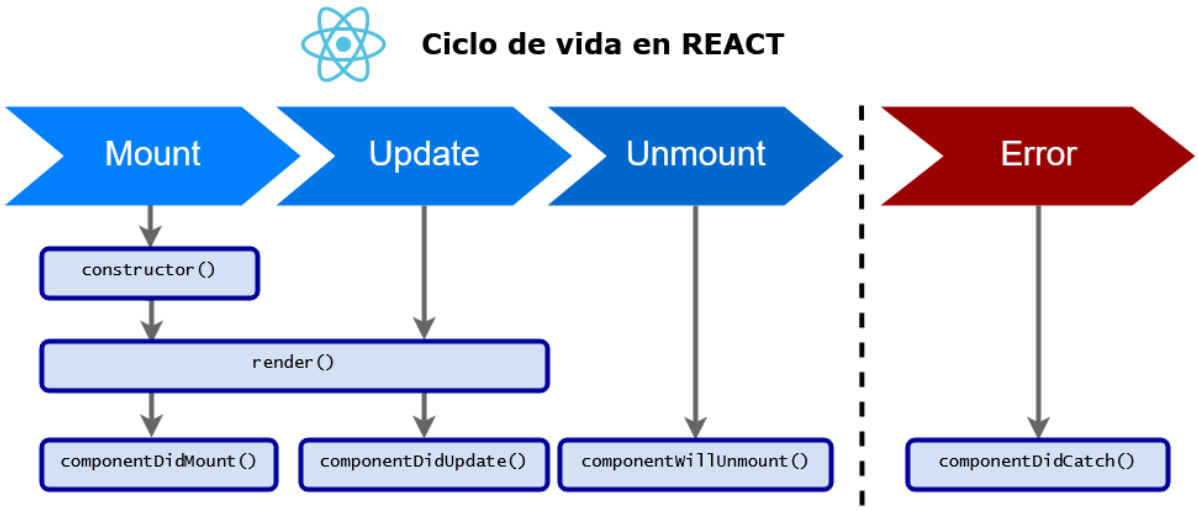

Imagen 13 - Ciclo de vida del componente en React

Comenzamos por la fase de montaje del componente. El montaje lo entendemos como todo el proceso al final del cual el componente estará listo e insertado como nodo en el árbol DOM. En esta fase inicialmente se invoca al **constructor** de la clase del componente, aquí es donde se debe inicializar el estado. Seguidamente se llama al método ***render()*** encargado de devolver la representación del componente. En último lugar, una vez el componente ha completado su proceso de ensamblado, se invoca el método ***componenteDidMount()***, en este punto tendremos ya disponible el contexto del navegador, y se podrán hacer suscripciones a eventos, definir intervalos, llamadas AJAX, etc.

La siguiente fase es la de actualización, esta fase es la encargada de actualizar la representación del componente cuando sus propiedades o estado cambian. Inicialmente se llama al método ***render()*** que al igual que en la fase de montaje retornará la representación del componente para su estado y propiedades actuales. Seguidamente ya con el DOM actualizado, se invocará el método ***componentDidUpdate(`prevProps`)*** en la variable de entrada `prevProps` tendremos el valor anterior de las propiedades del componente. Es importante tener cuidado al actualizar el estado dentro de este método, pues si no establecemos una condición de guardia, podemos causar un bucle de renderización infinito.

Cuando el componente ha terminado de usarse y se procede a eliminar la instancia del mismo, comienza la fase de desmontaje. En esta fase se invoca el método ***componentWillUnmount()*** y es el lugar indicado para cancelar suscripciones, timers, etc que pudiéramos tener activos y puedan causar una pérdida de memoria.

Para la gestión de errores en el componente React dispone del método ***componentDidCatch(`error`, `info`)***. Cuando el componente intercepte un error causado por un componente hijo, este método será invocado. El método recibe dos parámetros, `error`, con la excepción que lanzó el componente descendiente e `info`, un objeto Javascript con información de contexto del componente que lanzó el error.

Los componentes React son clases ES6 que extienden la clase `React.Component` , la cual implementa las funciones de ciclo de vida comentadas, permitiendo sobrecargarlas para incluir nuestra lógica. Hay un subtipo de componentes especiales en React, se trata de los componentes que no disponen de estado propio, a estos componentes, que solo cuentan con lógica en la función render, se les conoce como componentes sin estado o *stateless*. Por su naturaleza más sencilla y no tener que hacer uso de las funciones de ciclo de vida, en lugar de con una clase, se recomienda definirlos mediante una función flecha de ES6.

En desarrollo web tradicional es normal usar HTML o algún lenguaje de plantillas para la creación de componentes web. React en su lugar utiliza la sintaxis Javascript XML o JSX. Este lenguaje nos permite crear componentes utilizando una sintaxis con tags de tipo XML, similares a los tags de HTML y hacer uso de estos tags HTML dentro del código Javascript. Podremos crear nuestros propios tags y referenciar otros componentes, podemos diferenciar, los componentes definidos en React ya que comenzarán por letra mayúscula, al contrario de los tags HTML que siempre comienzan por minúscula. Este lenguaje permite también interpolar código Javascript dentro de tags usando llaves `{ código JS }`.

Hemos visto que en React el estado es local para cada componente, este estado aislado hace complejo el compartir información entre componentes. Para ayudarnos a solucionar este problema usaremos la librería REDUX, redux es un manejador de estado, que actuará como proveedor único para aquellos valores que hubieran de ser globales en la aplicación.

Vamos a ver las características básicas del patrón redux y su ciclo de funcionamiento. En el diagrama vemos que existen tres componentes, el store, las acciones y el reducer.

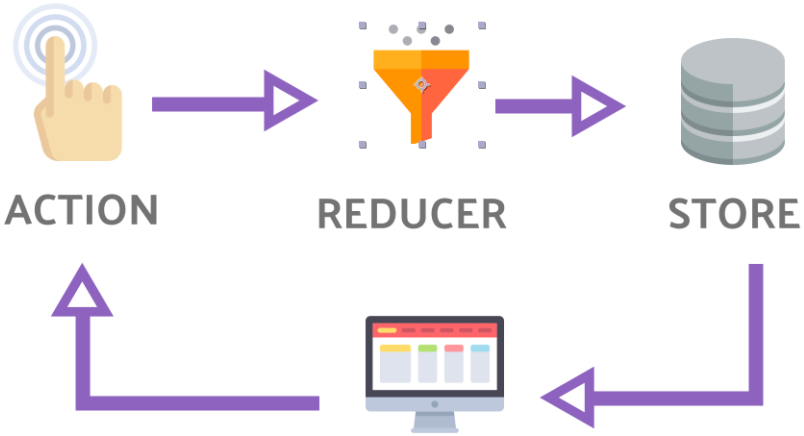

Imagen 14 - Flujo de funcionamiento en Redux

El *store*, es donde se almacena el estado, se trata de un objeto Javascript, esté *store* es de sólo lectura, y no podremos realizar operaciones de modificación directamente en él. Al igual que en React los datos viajan en una sola dirección.

Por otro lado, tendremos acciones, *actions* por su nombre en inglés, estas acciones son objetos que dispondrán de una clave llamada *type* para identificar el tipo de acción emitido y un valor o payload, con los datos asociados para generar el nuevo estado.

El siguiente componente es el reducer. El reducer es una función que recibe el estado actual y una acción, y como resultado devuelve el nuevo estado. El reducer ha de ser una función pura, una función pura es aquella en la que su valor de salida sólo depende de sus parámetros de entrada, y para un mismo valor de entrada siempre generará un mismo valor de salida predecible. Como dijimos el estado es de sólo lectura, y por ello inmutable, el reducer siempre devolverá una copia del objeto estado con los valores actualizados.

Para el soporte de enrutado de urls dinámicas se ha usado la librería Next.js, esta librería además de dar soporte al enrutado estático y dinámico, dota a la aplicación de funcionalidades de renderizado en el lado del servidor o SSR, bajo esta tecnología, la página no es servida de manera estática y ejecutada únicamente en el navegador del usuario, sino que es servida ya renderizada, ahorrando al navegador esa primera ejecución del código. Esta tecnología además

de ayudar a mejorar en rendimiento en esa primera carga, solucionar otros problemas relacionados con la indexación por motores de búsqueda y SEO.

Por último, para dar el estilo a los componentes, en lugar de usar las típicas hojas de estilo css y siguiendo las buenas prácticas de REACT y Nextjs usaremos CSS embebido en el Javascript de los componentes o CSS-in-JS [54], con ayuda de la librería Styled Components.
Las ventajas de Styled Components sobre REACT vainilla es que este permite el uso de propiedades, temas de estilo, extensión de estilos, referencias a componentes entre otras características.

4.2.2.2. Desarrollo del Front-end

Para comenzar en desarrollo vamos a deconstruir la interfaz diseñada en el apartado de boceto en los diferentes componentes que la forman. También detectaremos los componentes comunes de las distintas vistas.

Comenzaremos con la vista de galería, en la que se muestra el listado de emisoras, esta pantalla contiene dos elementos principales, la cabecera con el logotipo y la galería. Dentro de la galería vemos una rejilla que contiene una colección de elementos de tipo emisora, ese será otro de los componentes.

En la vista de reproducción, arriba observamos en la versión móvil una barra de menú, que permitirá desplegar el menú lateral. Bajo este, vemos de nuevo el componente de cabecera con el logotipo. Debajo tendremos el componente del reproductor de audio. Esta pantalla cuenta además con el menú lateral, que contendrá una lista de emisoras en formato columna, en la versión móvil, se aprecia este mismo componente siempre visible en pantalla.

En resumen, hemos detectado los siguientes componentes:

Componente	Descripción
Cabecera con logo	Contiene el logotipo y elementos de cabecera
Listado de emisoras (galería)	Muestra una rejilla con las estaciones de radio
Tarjeta de emisora (galería)	Cada elemento de la rejilla, información de la emisora
Barra de menú	Permite desplegar menú lateral en vista móvil
Reproductor de emisora	Información de emisora seleccionada y control de audio
Menú Lateral	Se despliega lateralmente en la vista móvil
Listado de emisoras (lateral)	Contiene un listado vertical de emisoras

Tarjeta de emisora (lateral)	Cada elemento del listado, información de emisora

Tabla 9 - Desglose del prototipo en componentes

En esta lista tenemos todos los componentes detectados en base al prototipo. Vemos que hay dos listados de emisoras y dos tarjetas de emisoras, estos componentes a priori parecen tener bastante en común ya que tan solo cambia el estilo visual con el que son mostrados, intentaremos juntarlos en un solo componente de galería o listado de emisoras y otro de tarjeta de emisora.

Dentro de la vista de galería, vamos a añadir un componente no detectado en la fase de prototipado, se trata de un componente de búsqueda, que nos permitirá hacer un filtrado en base a texto en el listado de emisoras.

Componente	Descripción
Caja de búsqueda	Permite filtrar en base a texto el listado de emisoras

Tabla 10 - Componente extra de búsqueda

Ahora que tenemos los componentes detectados, procederemos a definir el store que usaremos con Redux para manejar el estado de la aplicación. Los elementos que aparecen en la tabla siguiente, nos permitirán conocer en todo momento el estado completo de nuestra aplicación. Nuestra aplicación contará con un único store compartido.

Clave	Descripción	Valor inicial
`stations`	Listado de estaciones	nulo
`selectedStation`	Estación seleccionada	nulo
`menuOpened`	Estado de menu lateral	falso
`volume`	Volumen del audio	100
`muted`	Audio silenciado	falso
`filterText`	Filtro de búsqueda de emisoras	‘’

Tabla 11 - Claves del store Redux usada en la aplicación

En base a estos elementos, definimos las acciones que permiten modificar su valor ya que recordemos el store es inmutable. En la siguiente tabla se pueden observar todas las acciones que tendremos disponibles para gestionar en nuestro reducer y el valor que les podemos asociar.

Acción	Payload	Modificación de store
`STATIONS_LOADED`	listado de estaciones	stations

`STATION_SELECTED`	estación seleccionada	selectedStation
`MENU_TOGGLE`	estado del menú	menuOpened
`STATION_FILTER`	texto de búsqueda	filterText
`PLAYER_VOLUME`	valor del volumen	volume
`PLAYER_MUTE`	estado del silenciado de audio	muted

Tabla 12 - Acciones gestionadas por el reducer

En este punto se podría echar en falta los estados y acciones asociados al componente de reproductor de audio, pero durante este mismo análisis se ha detectado que este elemento no tiene necesidad de interactuar con los demás componentes ya que a ellos el estado actual de la reproducción (cargando, reproduciendo, en pausa, fin o error) no les es relevante. Por este motivo la gestión de esos estados se realizará de manera aislada y local dentro del componente reproductor en lugar de en el store global.

Antes de ponernos a crear componentes nos falta definir el layout global. Para esto crearemos un componente Layout que actuará de contenedor global de todos los componentes y definirá algunos estilos base. El contenedor tendrá un ancho máximo de 1335px. Para detectar la versión a mostrar, móvil o escritorio, usaremos media query de css, en este caso el punto de corte será un ancho máximo de 1024, cualquier cosa por debajo será gestionada como dispositivo móvil o tablet, y por encima de esa resolución se le tratará como si de un pc de escritorio se tratase.

Con todo esto ya estamos listos para dar vida a los diferentes componentes.

Componente de cabecera

Este componente es de los más sencillos, tan solo contiene el logotipo y a nivel funcional deberá llevarnos a la página de inicio. Este comportamiento lo conseguiremos con el uso de la acción `STATION_SELECTED` y el payload `null`. Que anulará cualquier posible selección de emisora actual y nos retornará a la vista de la galería.

Imagen 15- Logotipo de la aplicación

Barra de menú

De nuevo se trata de un componente sencillo, formado por un contenedor y una imagen con la típica hamburguesa que abrirá el menú lateral. Este componente sólo será visible en la versión móvil. Será uno de los encargados de gestionar el valor `menuOpened` del store, por ello

hará uso de la acción `MENU_TOGGLE` con el payload a true al pulsar sobre el icono de la hamburguesa.

Galería de emisoras

En este componente vamos a representar las emisoras obtenidas por la aplicación desde el back-end, para ello tendremos que hacer uso del valor stations del store, también consultaremos los valores `stationSelected` gracias a él sabremos si hemos de pintar una rejilla o una lista vertical y filterText que nos permitirá aplicar el filtrado sobre el listado de emisoras. A priori podría entenderse que desde aquí también se gestionará el evento de selección de emisora `STATION_SELECTED`, pero no será este, si no un componente hijo, la tarjeta de emisora, el encargado de ello.

Para la maquetación de este componente vamos a hacer uso de Flexbox de CSS, que nos permite simplificar el desarrollo. Como hemos detectado anteriormente, este componente tendrá dos representaciones, la vista de rejilla y la vista columna, que lo lograremos con la propiedad flex-flow.

Los componentes hijos, serán la caja de búsqueda y la colección de tarjetas de emisora.

Tarjeta de emisora

Este componente al igual que la galería tendrá dos visualizaciones, una en modo retrato y otro apaisado, en ambos mostrará la misma información, solo que con diferente distribución. De nuevo como en la galería conseguimos ambas visualizaciones con el mismo código JSX tan solo usando CSS.

Hace uso de la acción `STATION_SELECTED` que irá asociada al evento click sobre cualquier parte del componente. A continuación vemos los diseños para ambas vistas.

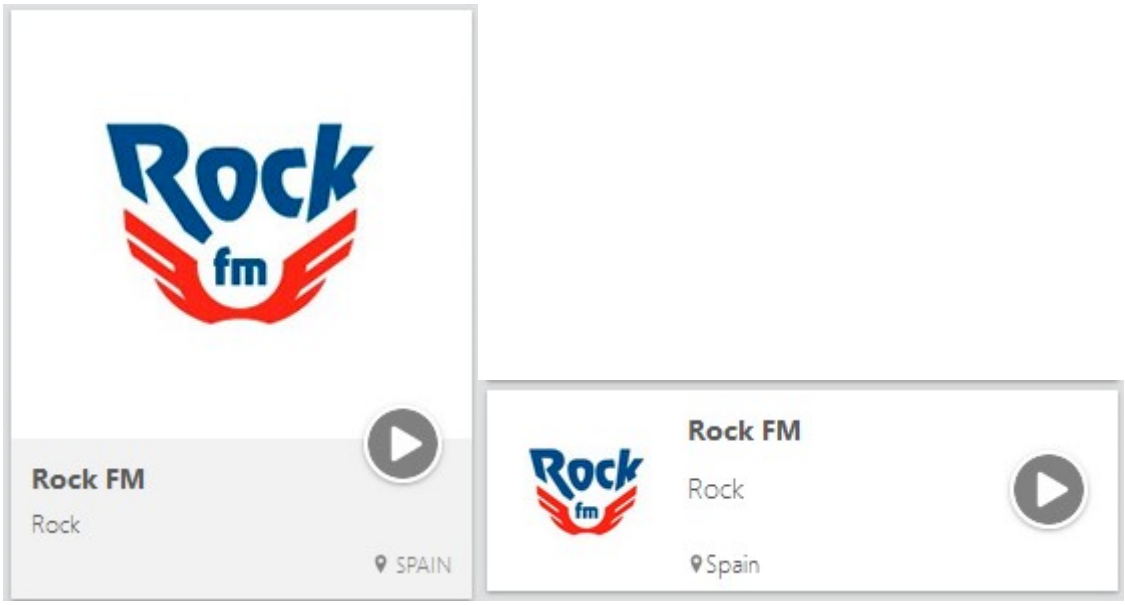

Imagen 16 - Diseños de tarjeta de emisora, retrato y apaisado

La tarjeta muestra información con el logotipo, el género musical, el país de la emisora y un pequeño botón que actúa "llamada a la acción" y nos invita a reproducir el audio.

Caja de búsqueda

De nuevo se trata de un componente sencillo, será el encargado de emitir la acción `STATION_FILTER` al reducer y facilitar la búsqueda de una emisora concreta dentro del listado, como payload usará el valor de la caja de texto. Se mostrará siempre encima del listado. Se compone de una imagen con la típica lupa y un input de texto.

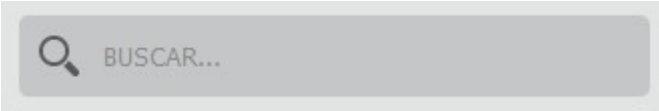

Imagen 17 - Caja de búsqueda

Menú lateral

Este componente se usará como contenedor o padre para el listado de las emisoras en la vista de reproductor. Para su funcionamiento hace uso del valor `menuOpened` del store, y de la acción `MENU_TOGGLE` con valor false que nos permitirá cerrarlo en la versión móvil.

Como ya hemos dicho lo usaremos como contenedor, por sí mismo solo contiene una imagen con el icono de cruz al que se asociará la acción de cierre de menú. En la versión móvil también mostrará un fondo translúcido, conocido como backdrop, sobre la parte visible del reproductor cuando el menú se encuentre abierto.

Como componente hijo tendrá el listado de estaciones en orientación columna.

Reproductor

En este componente haremos gestionaremos toda la lógica de reproducción. Hace uso de los valores `stationSelected`, para obtener la emisora seleccionada, `volume` y `muted` para el control del volumen. Gestiona también la emisión de las acciones `PLAYER_VOLUME` y `PLAYER_VOLUME`, para modificar el volumen o silenciar el audio.

Para controlar la reproducción dentro de este componente se ha usado la librería Javascript Howler [55] en la versión 2.1. El uso de esta librería simplifica el trabajo con diferentes navegadores y nos asegura la máxima compatibilidad. Para tener control sobre el estado de la reproducción el componente dispone de estado interno con las siguientes propiedades.

Clave de estado	Descripción y valores
`playerState`	estado de reproducción: `LOADING, PLAYING, PAUSED, ERROR`
`player`	Objeto howler del reproductor, null inicialmente
`meta`	Objeto que contendrá los metadatos de la emisora

`error`	En caso de error contendrá una descripción del mismo

Tabla 13 - Estados locales del componente reproductor

Este componente hace uso del ciclo de vida de react. En el `constructor` se crea el objeto player, en caso de que ya existiera se elimina el anterior y se crea uno nuevo, también se resetea la información de metadatos que pudiera existir y se intenta iniciar la reproducción del audio. En el evento `componentDidMount`, se comprueba si la emisora dispone de metadatos y en caso afirmativo se crea un intervalo que cada 60 segundos tratará de refrescar esos metadatos, haciendo una llamada al back-end. En la función `componentWillUnmount`, eliminamos el intervalo de refresco de metadatos en caso de existir y también la descarga del objeto de estado player, con esto evitamos posibles pérdidas de memoria.

En cuanto al apartado gráfico el reproductor cuenta con los siguientes elementos. (1) Imagen con el logotipo de la emisora seleccionada. (2) Una etiqueta con el bitrate reportado por la emisora. (3) Nombre de la emisora junto a una animación dinámica que cambiará si la estación se está cargando, reproduciendo o en pausa. (4) Los metadatos en caso de que estén disponibles aquí se mostrará también un posible mensaje de error. (5) Control de audio, con un icono que permite silenciar y una barra de desplazamiento horizontal que controla el volumen. (6) Y un botón de función dinámica que permite pausar o reanudar la reproducción.

Imagen 18 - Elementos del componente reproductor

Este último componente de la interfaz representa el estado del reproductor y modifica la imagen mostrada en función del estado del reproductor, valor de `playerState`.

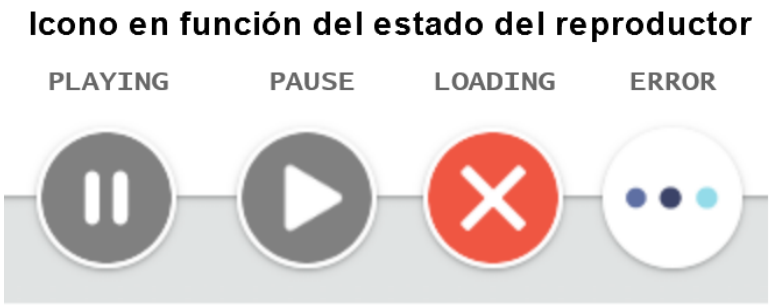

Imagen 19 - Imágenes en función del estado del reproductor

Vistas

Con los componentes definidos es hora de crear las dos vistas o páginas, para esta tarea usaremos Next.js. Gracias a esta librería podemos asociar rutas a las vistas y actualizar la URL durante la navegación. Además, esta gracias a esta librería como ya comentamos obtendremos renderización en el lado del servidor. En Next.js hemos de definir las páginas que serán diferentes puntos de entrada a la aplicación. Estás páginas serán a su vez componentes React, de un orden superior a los ya creados.

La vista principal o Index, de la aplicación será la responsable de realizar la primera carga de estaciones desde el API para poblar el `store` de Redux. Usaremos el método del ciclo de vida `componentDidMount()` para hacer esa llamada, siempre que el valor `stations` del `store` esté vacío. Esta página también será la encargada de pintar el mensaje de error en caso de no poder conectar con el API para recuperar las emisoras. Una vez recuperadas, se cargará dentro del layout los componentes de Cabecera y Listado de emisoras en modo galería.

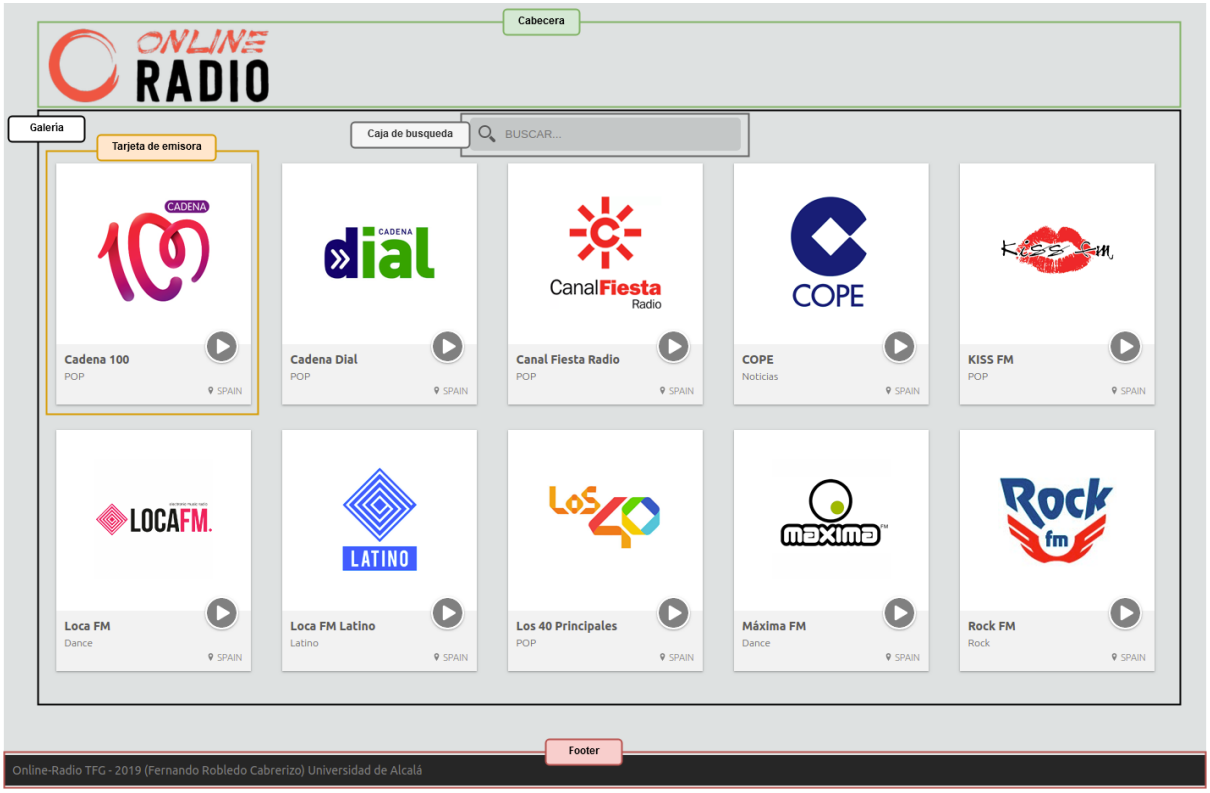

Imagen 20 - Vista de galería de la aplicación en desktop

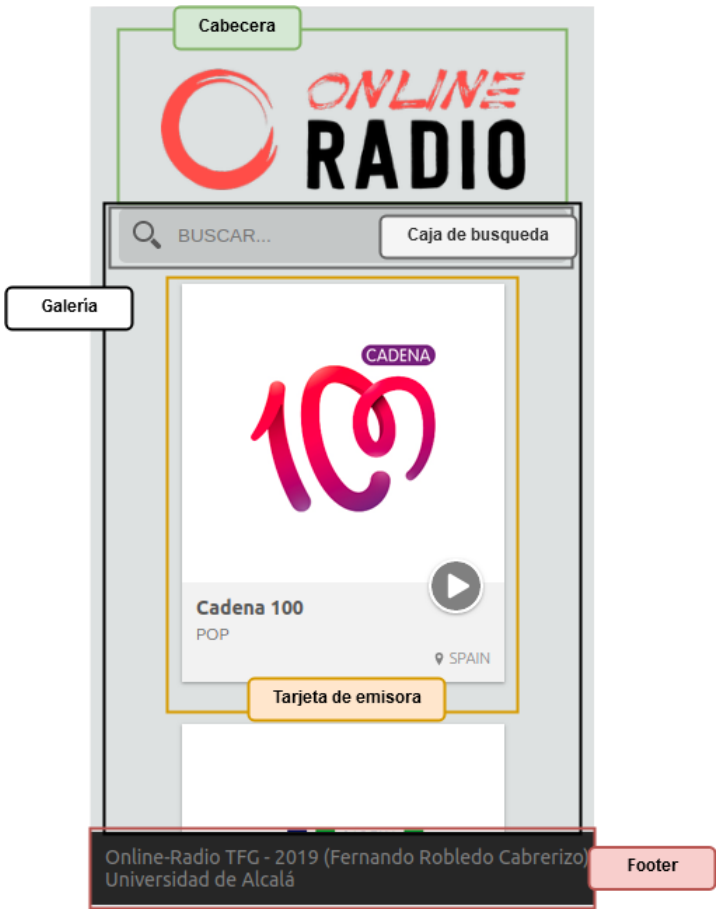

Imagen 21 - Vista de galería de la aplicación en móvil

En las imágenes 20 y 21, podemos ver los diferentes componentes que forman la vista del índice y su aspecto. Las tarjetas de las emisoras, al ser pulsadas, enlazan a la vista del reproductor. Para trabajar con Next.js en lugar del tradicional tag de HTML `<a>` para enlaces, usamos el componente `<Link>`. Este componente Link, evita que el navegador realice una navegación real, y nos asegura el correcto funcionamiento de nuestra SPA, ya se actualizará la barra de dirección del navegador y el historial, pero no se hará una recarga completa de página.

La segunda vista, corresponde al reproductor de audio. Usará la funcionalidad de crear páginas dinámicas [56] que ofrece Next.js. Como ya hemos comentado las páginas creadas con Next son componentes React, estos componentes reciben propiedades extra, como las diferentes partes de la URL, esta información la tendremos disponible dentro del componente y se podrá compartir con componentes hijos. En esta ruta dinámica capturaremos todo lo que nos llegue como parte la url, siguiendo el siguiente patrón `https://dominio-app.com/[nombre-emisora]` en una variable que llamaremos `station`, y si se trata de un nombre de emisora válido, cargaremos la vista del reproductor, en caso contrario se mostrará la vista de galería.

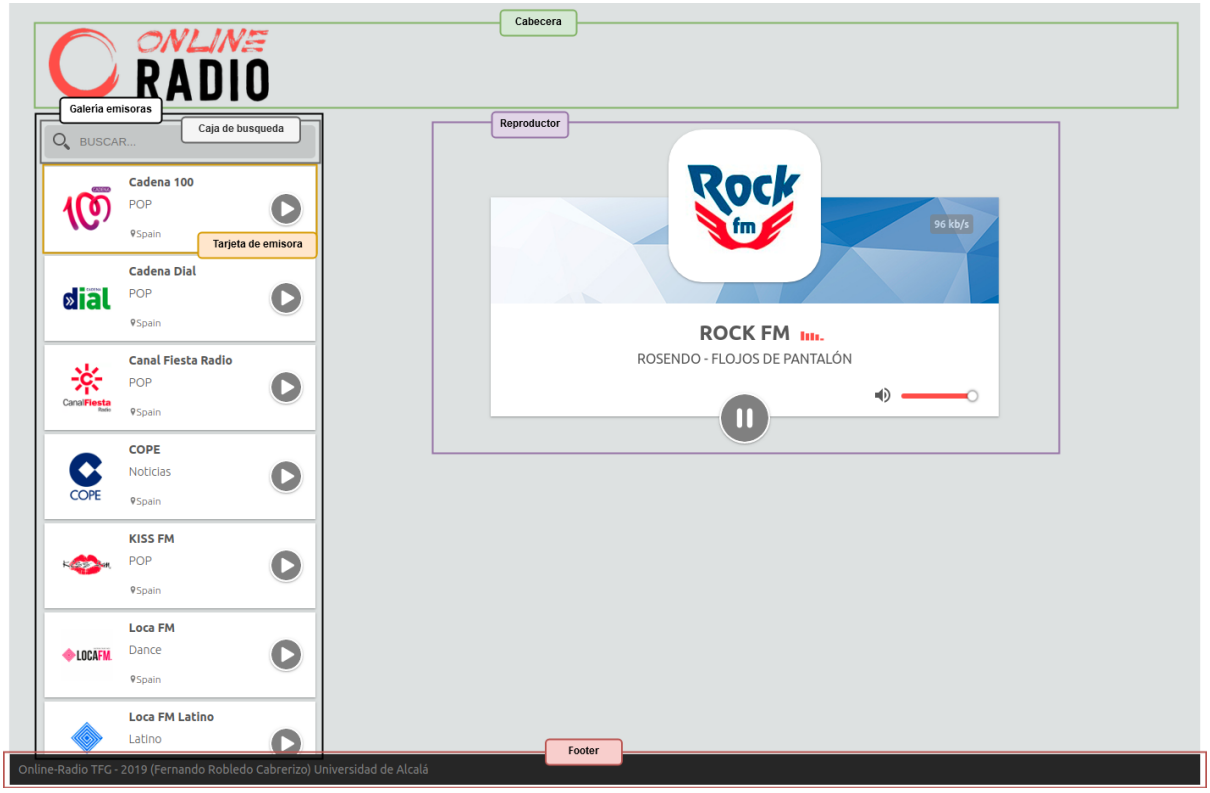

Imagen 22 - Vista de reproductor en escritorio

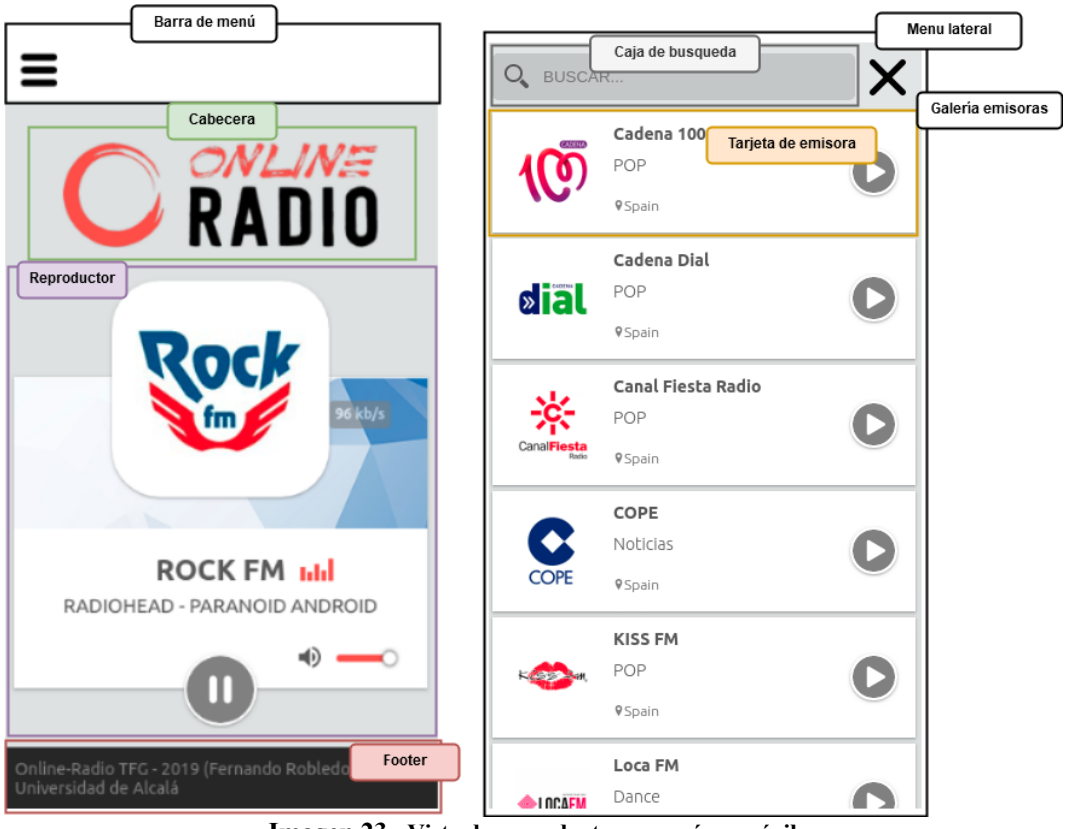

Imagen 23 - Vista de reproductor y menú en móvil

En las imágenes 22 y 23, se puede ver el aspecto final de la página en su versión escritorio y móvil respectivamente, así como la descomposición en componentes.

Gracias al routing dinámico y el uso del componente `<Link>` de Next.js tendremos navegación por URL, con el comportamiento propio de una SPA. Si cargamos una nueva ventana usando una URL del tipo `https://dominio-app.com/[nombre-emisora]` podremos recuperar el estado de la aplicación.

5. Presupuesto

En este punto procederemos a estimar los costes de la realización del proyecto.

En primer lugar, consideraremos los costes materiales asociados al proyecto.

Concepto	Precio	Amortización	Uso	Total
PC Desarrollo	2.500,00 €	36 meses	3 meses	208,33 €
Periféricos	600,00 €	36 meses	3 meses	50,00 €
Conexión Internet	35,00 €	36 meses	3 meses	105,00 €
				363,33 €

Tabla 14 - Costes materiales

Además de estos costes materiales, hay otros costes de infraestructura ligados a la puesta en vivo de la aplicación.

Servicio	Coste mensual	Unidades	Total
2x Cloud Engine (micro)	8,85 €	3 meses	26,65 €
Storage 10GB	0,21 €	3 meses	0,63 €
Cloud SQL	8,81 €	3 meses	26,43 €
Memory Store (REDIS)	33,60 €	3 meses	100,80 €
			154,41 €

Tabla 15 - Costes de infraestructura

El coste de personal del proyecto es el siguiente:

Perfil	Precio / Hora	Horas	Total
Desarrollador Back-end	30,00 €	100	3.000,00 €
Desarrollador Front-end	30,00 €	220	6.600,00 €
			9.600,00 €

Tabla 16 - Costes de personal

La suma de todos los totales anteriores nos da como resultado el coste de ejecución. Este coste asciende a 10,117.74 €. A esta cantidad hay que sumar el beneficio de empresa, que representa un 20% y el IVA con un tipo del 21%. Tras aplicarlos el coste total del proyecto asciende a **14.690,96 €**.

Concepto	%	Total
Coste de ejecución	-	10.481,08 €
Beneficio de industria	20,00 %	2.023,55 €
IVA	21,00 %	2.549,67 €
	Total IVA incl.	**14.690,96 €**

Tabla 17 -Costes totales

6. Conclusiones

La realización de este trabajo me ha permitido profundizar en el conocimiento de la librería REACT, la cual conocía de manera muy superficial.

La curva de aprendizaje ha sido progresiva. La aplicación ha tenido varias fases de desarrollo incrementales, en las que se ha ido variando la arquitectura de la misma incorporando nuevas librerías.

En la primera fase, se utilizó solamente React, y a pesar de ser una aplicación sencilla la tarea de comunicar valores de estado entre los diferentes componentes hacía que la complejidad aumentase notablemente y estos estuvieran acoplados.

En la siguiente fase, se introdujo la librería Redux, esto me permitió afianzar este patrón de gestión de estado global, al implementarlo la complejidad de la aplicación se redujo. La gestión del estado resultaba sencilla y era centralizada, pudiendo identificar qué componente hace uso de cada valor y cuando. Considero el uso de una librería de este tipo, para la gestión de estado, obligatoria para el desarrollo de aplicaciones front-end.

Sin duda la parte más compleja ha sido la última fase, con la inclusión de la librería Next.js de renderizado en lado del servidor (SSR). En la aplicación solamente se ha usado la función de manejo de rutas, a pesar de ello hubo que rehacer los puntos de entrada a la aplicación y crear componentes de orden superior. También hubo que modificar la forma de dar estilo a los componentes y migrar al uso de Styled Componentes.

En conclusión, en la tarea de crear una página web usando el paradigma SPA, React facilita enormemente la tarea, pero es en conjunción con otras librerías como Redux y NextJs cuando adquiere su mayor potencial.

El realizar una primera fase de análisis y prototipado me ha permitido tener claro que necesitaba crear y cómo podía hacerlo, a medida que los conocimientos del software usado aumentaban.

Durante la creación del back-end el uso de frameworks como Symfony, que en este caso ya conocía bien de antemano, reducen enormemente el tiempo de desarrollo y ofrecen unos cimientos sólidos para crear un API REST u otro tipo de aplicación.

El proyecto deja abiertas múltiples líneas de trabajo futuras.

En la parte de back-end:

- Implementar una capa de autentificación, preferiblemente basada en JSON Web Token (JWT).
- Con la autenticación ya implementada añadir nuevos endpoint para permitir las operaciones restantes CRUD (create, update y delete) sobre el recurso estaciones.

En el front-end:

- Hacer más uso del SSR, realizando las llamadas al API también desde el lado del servidor, sirviendo la página ya renderizada y con el estado hidratando en la primera carga, creando una aplicación isomórfica [57].
- Utilizar la capa de registro del back-end para permitir a los usuarios crearse un perfil y guardar sus estaciones favoritas.
- En estaciones con meta-datos, integrar APIs de terceros, como discogs [58] o lastfm [59], para recuperar las caratulas de canciones y álbumes, e incluir enlaces a tiendas de audio online como Amazon music, iTunes o Google music, obteniendo ingresos gracias a sus programas de afiliados. De igual manera permitir a los usuarios añadir las canciones a su lista de favoritos.
- Añadir capacidades sociales, como comentarios y enlaces de compartir, para generar interacción entre usuarios.
- Ofrecer distintos temas de estilo al usuario mediante el uso de la funcionalidad ThemeProvider de Styled Componentes.

7. Referencias

1. MDN web docs, Lenguaje HTML5, [Online] Disponible: https://developer.mozilla.org/es/docs/HTML/HTML5
2. MDN web docs, Hojas de estilo en cascada CSS, [Online] Disponible: https://developer.mozilla.org/es/docs/Web/CSS
3. WC3Schools, Versiones del lenguaje Javascript, [Online] Disponible: https://www.w3schools.com/js/js_versions.asp
4. Ralf S. Engelschall, Características de ES6, [Online] Disponible: http://es6-features.org/
5. Alexis Deveria, Can I Use, consulta de soporte de características soportadas por navegador, [Online] Disponible: https://caniuse.com/
6. StatCounter, Estadísticas de uso de navegadores, [Online] Disponible: https://gs.statcounter.com/browser-market-share
7. Google, Documentación oficial Angular, [Online] Disponible: https://angular.io/
8. Facebook, Documentación oficial React, [Online] Disponible: https://reactjs.org/
9. Evan You, Documentación oficial VUE, [Online] Disponible: https://vuejs.org/
10. Wikipedia, Transferencia de Estado Representacional, [Online] Disponible: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional
11. Wikipedia, Notación de objeto de JavaScript, [Online] Disponible: https://es.wikipedia.org/wiki/JSON
12. The PHP Group, Versiones PHP, [Online] Disponible: https://www.php.net/releases/index.php
13. Symfony Framwork, Symfony 4, [Online] Disponible: https://symfony.com/4
14. Riley Graham Longo, Mobile First: What Does It Mean?, [Online] Disponible: https://www.uxmatters.com/mt/archives/2012/03/mobile-first-what-does-it-mean.php
15. Wikipedia, Diseño web adaptable, [Online] Disponible: https://es.wikipedia.org/wiki/Diseño_web_adaptable
16. ePirat, Protocolo ICECast, [Online] Disponible: https://en.wikipedia.org/wiki/Icecast https://gist.github.com/ePirat/adc3b8ba00d85b7e3870
17. Chinmay V S, Protocolo Shoutcast, [Online] Disponible: https://thecodeartist.blogspot.com/2013/02/shoutcast-internet-radio-protocol.html
18. Wikipedia, Principios SOLID, [Online] Disponible: https://es.wikipedia.org/wiki/SOLID
19. Wikipedia, Princpio no te repitas (DRY), [Online] Disponible: https://es.wikipedia.org/wiki/No_te_repitas
20. Wikipedia, Arquitectura Single Page Application, [Online] Disponible: https://es.wikipedia.org/wiki/Single-page_application
21. Wikipedia, AJAX - Asynchronous JavaScript And XML, [Online] Disponible: https://es.wikipedia.org/wiki/AJAX
22. Wikipedia, Bajo acoplamiento, [Online] Disponible: https://en.wikipedia.org/wiki/Loose_coupling
23. Wikipedia, Red de distribución de contenidos, [Online] Disponible: https://es.wikipedia.org/wiki/Red_de_distribuci%C3%B3n_de_contenidos
24. Wikipedia, Document Object Model, [Online] Disponible: https://es.wikipedia.org/wiki/Document_Object_Model
25. Stefan Krause, Pruebas de rendimiento de diferentes frameworks Javascript, [Online] Disponible: https://stefankrause.net/js-frameworks-benchmark8/table.html

26. Google, Arquitectura Anguar, [Online] Disponible: https://angular.io/guide/architecture
27. Evan You, Comparación de VUE frente a otros frameworks: https://vuejs.org/v2/guide/comparison.html
28. Wikipedia, Inversión de control, [Online] Disponible: https://es.wikipedia.org/wiki/Inversi%C3%B3n_de_control
29. Webpack Team, Tree shaking, [Online] Disponible: https://webpack.js.org/guides/tree-shaking/
30. Facebook, Documentación oficial de Flow, [Online] Disponible: https://flow.org/en/docs/getting-started/
31. Microsoft, Documentación oficial de Typescript, [Online] Disponible: https://www.typescriptlang.org/docs/home.html
32. Google Trends, Comparativa de búsqueda entre React, Angular, AngularJS y VUE, [Online] Disponible: https://trends.google.es/trends/explore?date=2013-02-01%202019-05-18&q=%2Fm%2F012l1vxv,%2Fg%2F11c6w0ddw9,%2Fg%2F11c0vmgx5d,%2Fm%2F0j45p7w
33. Pnowy, Utilidad Github Starts History, [Online] Disponible: https://stars.przemeknowak.com/
34. MDN web docs, Web Audio API, [Online] Disponible: https://developer.mozilla.org/es/docs/Web_Audio_API
35. ICECast, página oficial, [Online] Disponible: http://icecast.org/
36. Shoutcast, página oficial, [Online] Disponible: https://www.shoutcast.com/
37. Métodos de petición HTTP, [Online] Disponible: https://developer.mozilla.org/es/docs/Web/HTTP/Methods
38. CURD, [Online] Disponible: https://es.wikipedia.org/wiki/CRUD
39. Symfony Framework, Documentación oficial Symfony, [Online] Disponible: https://symfony.com/doc/current/index.html
40. Wikipedia, Modelo Vista Controlador, [Online] Disponible: https://es.wikipedia.org/wiki/Modelo–vista–controlador
41. Symfony Framework, Componente serializador de Symfony, [Online] Disponible: https://symfony.com/doc/current/components/serializer.html
42. Doctrine Project, Documentación oficial Doctrine, [Online] Disponible: https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html
43. Doctrine Project, Lenguaje DQL, [Online] Disponible: https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/dql-doctrine-query-language.html
44. Martin Fowler, Patron Data Mapper, [Online] Disponible: https://martinfowler.com/eaaCatalog/dataMapper.html
45. Wikipedia, REDIS, [Online] Disponible: https://es.wikipedia.org/wiki/Redis
46. PHP FIG, página oficial, [Online] Disponible: https://www.php-fig.org/faqs/
47. PHP FIG, estándar para caché PSR-6, [Online] Disponible: https://www.php-fig.org/psr/psr-6/
48. Composer, Introducción a Composer, [Online] Disponible: https://getcomposer.org/doc/00-intro.md
49. Wikipedia, Librería cURL, [Online] Disponible: https://es.wikipedia.org/wiki/CURL
50. Dan Abramov y Andrew Clark, Librería Redux, [Online] Disponible: https://es.redux.js.org/
51. ZEIT, Caracteristicas de Next.js, [Online] Disponible: https://nextjs.org/#features
52. Styled Componentes, Documentación oficial Styled Components, [Online] Disponible: https://www.styled-components.com/docs

53. Facebook, Estado y ciclo de vida React, [Online] Disponible: https://es.reactjs.org/docs/state-and-lifecycle.html
54. Facebook, Estilo y CSS en REACT, [Online] Disponible: https://es.reactjs.org/docs/faq-styling.html
55. Howler, [Online] Disponible: https://howlerjs.com/
56. ZEIT, Routing dinámico en Next.js https://nextjs.org/docs#dynamic-routing
57. Wikipedia, Javascript Isomorfico, [Online] Disponible: https://en.wikipedia.org/wiki/Isomorphic_JavaScript
58. DiscDogs, API para desarrolladores, [Online] Disponible: https://www.discogs.com/developers
59. Last.fm, API para desarrolladores, [Online] Disponible: https://www.last.fm/api/

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá