Musikverein

Jose Serralvo, Ramón Soler, Raúl Cátedra y Verónica Velázquez

Agradecimientos

Gracias al equipo docente, mentores y al resto de grupos por su apoyo y asesoramiento durante el desarrollo del proyecto. Gracias también a los integrantes de este grupo, la colaboración y la comunicación han sido fundamentales para conseguir los objetivos y desarrollar este proyecto.

Introducción

En el presente proyecto se va a desarrollar una plataforma web de música basada en SoundCloud/Spotify. Para ello, se desarrollará un sistema cliente-servidor, empleando como tecnologías *NodeJS*, *Express*, *MongoDB*, *Firebase* y *Cloudinary* para la implementación del servidor y *ReactJS* con *Redux* para la implementación del cliente.

Objetivos

Los objetivos de este proyecto se dividen en objetivos generales y objetivos específicos. A continuación, se exponen los distintos objetivos generales junto con sus correspondientes objetivos específicos.

- Aprender a desarrollar un proyecto profesional en equipo, aplicando *SCRUM* como metodología *Agile*.

  Crear un proyecto usando *Github projects* en el que tener un *Kanban* con el que poder ver el estado del proyecto.
  Establecer las tareas de los cuatro *Sprints* para adaptar el desarrollo a la metodología planteada.
  Agendar reuniones semanales para la consecución de *Sprint reviews* y *Sprint plannings*.
  Agendar reuniones diarias para la consecución de los *Daily StandUps.*
  Agendar reuniones con los expertos cuando sean necesarias.
  Dividir las tareas entre los distintos integrantes del grupo.
  Mejorar las habilidades comunicativas de los integrantes del grupo.

- Profundizar y fortalecer los conocimientos de desarrollo de la parte de servidor con las tecnologías marcadas.

  Crear una *API REST* con *NodeJS, Express y MongoDB para el guardado y envío de datos del cliente.*
  Desarrollar los diferentes *end-points* para el correcto funcionamiento de los servicios del servidor.
  Establecer los distintos *middlewares* para la validación de autentificacion y datos que se reciben desde el cliente.
  Crear los distintos modelos para la consistencia en el guardado de datos en *MongoDB*.
  Establecer las comunicaciones con los distintos servicios externos como *Firebase* o *ReCaptcha.*
  Desarrollar una capa de abstracción para facilitar el cambio entre tecnologías de bases de datos.
  Crear test para la verificación de código.

- Profundizar y fortalecer los conocimientos en el desarrollo del cliente con las tecnologías marcadas.

  Crear un cliente con *Redux* y ReactJS.
  Desarrollar una arquitectura robusta de *Redux* para el correcto funcionamiento del cliente.
  Crear las diferentes páginas y vistas necesarias para el correcto funcionamiento del cliente.
  Establecer los componentes necesarios.
  Desarrollar las conexiones con los distintos servicios, tanto propios como externos.
  Crear la capa de abstracción de conexiones con el servidor para poder cambiar con facilidad entre servidores en caso de ser necesario.
  Dar estilos para una correcta visualización de la aplicación.
  Aprender y profundizar en el uso de librerías para atajar problemas comunes.

- Enriquecer los conocimientos y el *Portfolio* personal para reflejar la calidad del aprendizaje tecnológico y de *Soft-Skills*.

  Trabajar en equipo para la consecución de los mejores resultados posibles.
  Asistir a las reuniones marcadas para tratar el avance del proyecto.
  Cumplir con la metodología *Agile*.
  Tomar decisiones de forma democrática entre todos los integrantes del grupo.
  Desarrollar una documentación que acompañe y explique el proyecto.

Metodología

A continuación, se expone el conjunto de procedimientos que se deben seguir para la consecución de los objetivos antes definidos.

- **Metodología *Agile***

  Se define como metodología a utilizar *Scrum*:

  > *Scrum* es un framework de trabajo que permite mejorar el trabajo colaborativo en equipo y, a su vez, una metodología dentro de la mentalidad *Agile*. En *Scrum*, lo importante es trabajar de una manera que permita iterar las entregas del desarrollo. En cada una de estas entregas, se revisa la situación, se aclaran y corrigen desviaciones, y se prepara un nuevo conjunto de tareas para la siguiente iteración, con la decisión propia de cada componente del equipo de responsabilizarse en cada una de las tareas para la siguiente entrega.
  > Cuando hablamos de iteraciones, nos referimos a *Sprints*. Los *Sprints* son periodos breves de tiempo fijo en los que un equipo trabaja para completar unas tareas previamente escogidas. En cada *Sprint,* el equipo irá trabajando, entonces, sobre las tareas previamente decididas. Por lo general, estos *Sprints* suelen ser de dos semanas (*standard*) o de una semana (con mayor control). Una vez finalizado el *Sprint*, el equipo realizará una serie de reuniones en las que revisará y hará una demostración interna de lo que se ha desarrollado, confirmará la finalización de todas las tareas marcadas como completadas y se volverá a distribuir un nuevo conjunto de tareas para la siguiente iteración.
  > Para el correcto desarrollo del proyecto, son necesarias reuniones del grupo, completando al menos una reunión semanal de una hora (*Sprint review & Sprint planning*) y una reunión diaria de quince minutos (*Daily StandUp)*.
  > Se crea una división en cuatro *Sprints*, correspondientes a cuatro semanas de trabajo, en las que se dividen las tareas de forma que la carga de trabajo semanal sea congruente con las capacidades del grupo y equitativa entre semanas.

- **Canales de comunicación**

  Se definen los siguientes canales de comunicación:

  > *Microsoft Teams,* para la comunicación oral entre los integrantes del grupo.
  > *Google Meats,* para las reuniones pertenecientes a la metodología *Agile* y las reuniones con expertos y mentores.
  > *Github Projects,* para el control del avance de las tareas y su estado.

- **Tareas**

Las tareas básicas vienen definidas en el proyecto semilla, por lo que se pasa a integrarlas en *Github Projects*. Además de estas tareas, se crean y establecen nuevas tareas por parte de los integrantes del grupo con las que añadir valor y funcionalidades al proyecto.

La asignación de las tareas se hace a través de *Github Projects*, en la que los componentes del grupo se asignan las tareas de forma equitativa, de manera que todos hagan tareas que incluyan diseño, tecnologías del cliente y tecnologías del servidor.

La ejecución de la mayoría de las tareas se lleva a cabo de forma grupal, en formato *Pair-Programming,* de forma que todos los integrantes se hacen partícipes del conjunto completo del código.

- **Escritura de código**

Para la escritura de código se decide utilizar *Visual Studio Code*, ya que todos los integrantes del grupo conocen bien sus funcionalidades, además de poder acceder a configuraciones generales de estilo y del código a través de sus distintos *Plugs-in* gracias a la configuración básica que acompaña a la semilla del proyecto.

Desarrollo del proyecto

Descripción del producto

Existen muchas aplicaciones en el mercado relacionadas con la reproducción de música. Algunas de ellas son *Soundcloud*, *Tidal* o *Spotify*. Musikverein se encuentra dentro del sector de este tipo de aplicaciones, con mayor similitud con *Soundcloud*, por el hecho de que este permite la publicación de canciones por parte del usuario y no existe una diferenciación clara entre usuario y artista.

Objetivo del producto

El objetivo del producto es realizar una *fullstack webapp* empleando tecnología *MERN* para la publicación y reproducción de canciones entre una comunidad de usuarios. Los usuarios podrán acceder a la aplicación para reproducir canciones subidas por ellos mismos o canciones subidas por otros usuarios.

La elección del nombre *Musikverein* se debe a su significado en alemán: “asociación musical”, así como por el Wiener Musikverein de Viena, edificio que alberga varias salas de conciertos y otras instituciones musicales, abierto al público desde el 6 de enero de 1870, famoso por su acústica (situada entre las mejores salas del mundo), además de por su archivo y biblioteca, ya que alberga una de las principales colecciones musicales del mundo.

Las principales funcionalidades del producto son:

- Publicación, visualización, reproducción y gestión de canciones y *playlists.*
- Gestión de “me gusta” de canciones, “follows” de usuarios y *playlists.*
- Buscador de canciones, *playlists* y usuarios.
- Visualización de recomendaciones en la pantalla *Home.*
- Sistema de cola de reproducción.

Las funcionalidades se distribuyen en los siguientes *Sprints:*

- *Sprint* 1: registro y autenticación de usuarios. Gestión del perfil del usuario.
- *Sprint* 2: subida, reproducción y gestión de canciones.
- *Sprint* 3: creación y administración de playlists.
- *Sprint* 4: visualización y seguimiento de usuarios. Estadísticas de las entidades del sistema.
- *Sprint* extra: finalización de tareas no acabadas en anteriores iteraciones, documentación y presentación de producto.

Desarrollo

El proyecto se inicia el día 12 de abril de 2021, con la copia de la semilla y la configuración de todos los canales y metodologías necesarias para el desarrollo. A partir de ese momento, se sigue con las metodologías indicadas, realizando las distintas tareas correspondientes al *Sprint* en el que el grupo se encuentra, acudiendo a las reuniones agendadas hasta el día 13 de mayo, en el que se da por concluido el desarrollo del proyecto.

A continuación, se detalla el desarrollo por *Sprints:*

- ***Sprint* 1**: a lo largo de la primera iteración, se desarrolla el registro y autenticación de usuarios, la edición de los datos de usuario, página de términos de uso y condiciones, validación de *ReCaptcha* y validaciones de datos tanto en la parte de cliente como en el servidor.
- ***Sprint* 2**: en la segunda iteración, se desarrolla el contenido relacionado con las canciones y su reproducción. Para ello, en la parte final de la iteración, se empieza a realizar una refactorización de la arquitectura de *Redux* para dividir los estados en *entities* y *ui,* con lo que se consigue un mejor funcionamiento de la aplicación, al tener una sola “fuente de la verdad” donde cambiar los datos cuando cambian en el servidor.
- ***Sprint* 3**: en la tercera iteración, se procede con el desarrollo del contenido relacionado con *playlists* y su reproducción. Gracias a la refactorización de la iteración anterior, se crea un sistema de cola en el reproductor, de manera que las canciones o *playlists* se pueden ordenar de forma independiente en una *playlist* o en dicha cola. Además, se comienza con el desarrollo del buscador, para su uso a la hora de añadir canciones a una *playlist.*
- ***Sprint* 4**: en la cuarta iteración, se desarrolla la vista de usuarios, completando el bucle de acceso entre todas las entidades de la aplicación, con ello se cierran las distintas entidades presentes. También se desarrolla el buscador final, el cual busca dentro de todas las entidades para un correcto funcionamiento de la aplicación.
- ***Sprint* extra**: en la quinta iteración, se procede a desarrollar e incluir en la aplicación la parte de estadísticas y recomendaciones, terminando la visualización de la página principal de la aplicación. Se realizan correcciones en el diseño, añadiendo la información sobre la consecución o no de las acciones de comunicación con servicios propios y de terceros, se crea toda la documentación adjunta y los test que acompañan al código, además de la presentación del producto.

Problemas encontrados

La arquitectura de *Redux* generó ciertos problemas al tener duplicidades de información, el cambio a una arquitectura con una sola “fuente de la verdad” nos hizo superar este obstáculo y crear una aplicación más robusta.
La creación de los modelos de metadatos generaron problemas debido a la complejidad a la hora de agrupar los datos por fechas y poder enviarlos de nuevo al cliente. Finalmente, optamos por un modelo mensual en el que guardar en distintas colecciones los datos de las distintas entidades.
No poder eliminar canciones en el servicio de *Cloudinary* debido a que la petición de subida se hacía desde el cliente y no se podían eliminar canciones sin una petición firmada (desde el servidor). En este caso, no se optó por la eliminación de las canciones en la base de datos propia, sino por tener un campo en el que detallar si la canción está activa o no.
La aplicación mostraba la página de LogIn en cada recarga, debido a las peticiones al servicio de *FireBase* que eran necesarias para comprobar si existía algún usuario *logueado.* Este problema fue solucionado con la introducción de un *spinner* de carga mientras se realizan dichas peticiones, por lo que no se muestra la página correspondiente hasta que no se terminan todas.
La sinergia entre *ReactJs* y *Redux* generaba errores de sincronización con componentes que habían dejado de estar *renderizados*, por lo que hubo que usar ciertos “hacks” para evitar dichos errores.

Problemas internos del equipo

El desarrollo del proyecto ha sido fluido y cordial, por lo que no han aparecido problemas de ningún tipo entre los integrantes del equipo, ya sean de comunicación o de diferencias de opinión no resueltas.

Resultados obtenidos

Los resultados obtenidos del desarrollo del proyecto fueron los esperados. Se cumplió con los objetivos mínimos, así como con la mayoría de los extras que el grupo propuso en sus tiempos estimados. Es por ello por lo que se puede afirmar que el equipo ha desarrollado una *fullstack webapp* en la que un usuario puede registrarse, *loguearse*, editar sus datos, subir canciones y reproducirlas, crear *playlists* y reproducirlas, seguir la actividad y música de otros usuarios, buscar otras canciones, *playlists* y usuarios, así como ver las canciones, usuarios y *playlists* destacadas dentro de la aplicación.

De la misma forma, se puede afirmar que el equipo ha aprendido cómo trabajar con la metodología *Agile Scrum*, cumpliendo con todos sus procesos para el mejor entendimiento entre los integrantes del grupo.

Por último, se ha cumplido con la creación de un producto que puede ser un buen *portfolio* de presentación de los integrantes del grupo, pudiendo defender de manera individual todo el código que contiene la aplicación desarrollada.

Mejoras en futuras iteraciones

Debido a la limitación de tiempo, hemos descubierto mejoras en el desarrollo que no se han podido llegar a implementar, por lo que el grupo pensó que es conveniente la descripción de las mismas para quien pueda utilizar este proyecto con licencia *open source* o para posibles futuras iteraciones sobre el mismo.

- Optimización de comunicaciones con el servidor: la comunicación de datos entre cliente y servidor no se realiza de forma óptima, pues no se ha desarrollado una paginación de los datos: con una base de datos no muy extensa no es un problema, pero si la aplicación comienza a recibir una gran cantidad de usuarios, se debería desarrollar dicha optimización para el correcto funcionamiento de la aplicación.
- No se ha podido integrar el multiidioma: a lo largo del desarrollo, se pensó en introducir dicha funcionalidad con la librería *React-i18next* por su facilidad de uso y aplicación, se recomienda la introducción de dicha funcionalidad para crear una aplicación accesible de forma internacional.
- Introducción de roles de usuario: crear diferenciación entre rol de artista o de usuario puede aportar un gran valor a la aplicación, ya que el artista podría tener una visualización de metadatos de sus canciones en distintas gráficas que le informasen sobre el estado en el que se encuentra dentro de la aplicación.
- Visualización de recomendaciones por géneros: la introducción en la página principal de una recomendación por los géneros más escuchados, mostrando las canciones pertenecientes a estos géneros, aportaría valor de uso a la aplicación.
- Aplicación de filtros al buscador. El buscador solo realiza búsquedas por título de canción o playlist, así como por nombre de usuario. La introducción de distintos filtros al buscador para una realización de búsquedas más avanzadas sería un gran aporte al proyecto.
- Eliminación de canciones: al encontrarnos con el problema anteriormente descrito de no poder eliminar canciones del servicio *Cloudinary* desde el cliente, se plantea la posibilidad de crear un *script* en el servidor que, periódicamente, realice una eliminación de las canciones que no están activas en la base de datos, así como los archivos correspondientes guardados en el servicio, de forma que no se acumulen datos innecesarios en ambos.

Conclusiones

Se ha realizado un proyecto completo y atractivo, con funcionalidades prácticas y sin errores de ejecución. Del mismo modo, se han cumplido los objetivos establecidos en el planteamiento inicial, tanto los generales como los específicos. El proyecto, en su versión actual, tiene una gran capacidad de escalado y crecimiento, por lo que podría retomarse en futuro, implementando las mejoras necesarias, y terminar de desarrollar un producto comercial competente en el mercado.