Presentación TFG

Intro

Hola, buenas. Yo soy Alberto Gómez Cano y voy a realizar la presentación de mi TFG.

Que tiene de nombre Earfit: Aplicación para Entrenamiento Auditivo Musical.

Y está basada en Nextjs y TypeScript.

Lo primero de todo vamos a repasar el contenido de la presentación.

→

Contenido

Primero, vamos a ver en la introducción: Un resumen de que es: el entrenamiento auditivo y los objetivos de este trabajo.

Luego, veremos las metodologías aplicadas en el desarrollo:

Design Thinking, Lean Startup, Scrum y DevOps.

Después, vamos a explicar cómo funciona la aplicación:

Tecnologías empleadas, implementación, etc…

Y por último, veremos las conclusiones del trabajo, con los objetivos alcanzados, trabajos futuros y un video de demostración.

→

Entrenamiento Auditivo

¿Qué es el Entrenamiento Auditivo?

El Entrenamiento Auditivo es el proceso de identificar y asociar los elementos de la música con la forma que percibimos el sonido.

Los músicos, productores y DJs pueden beneficiarse de este entrenamiento.

Ya que les permite sacar canciones más rápido, improvisar mejor y llevar al instrumento las melodías que se imaginen.

Los ejercicios más comunes son identificar notas, intervalos y escalas, etc…

Para los músicos puede resultar difícil realizar este Entrenamiento Auditivo por su cuenta.

Por lo que vamos a intentar crear una solución para este problema.

→

Objetivos

El objetivo principal de este TFG es crear una aplicación que permita a los músicos desarrollar su oído musical mediante ejercicios de entrenamiento auditivo.

Cómo subobjetivos tenemos:

Desarrollar una interfaz interactiva.

Implementar diferentes tipos de ejercicio a poder ser personalizables.

E incluir varios instrumentos para practicar con diferentes sonidos.

→

Metodologías

Para crear esta solución, nos basamos en el el siguiente proceso combinado de Design Thinking, Lean Startup y metodologías ágiles.

Design Thinking, se compone de cinco etapas no lineales:

- En Empatizar, comprendemos las necesidades del usuario con entrevistas y un estudio de teoría musical.
- Definimos sus problemas.
- Ideamos soluciones para estos problemas.
- Convertimos estas soluciones en un prototipo.
- Y Testeamos el prototipo con el usuario.

Durante estas etapas se utilizan varias técnicas cómo:

- User Persona para definir al usuario.
- Mindmap para definir los problemas y las soluciones.
- MoSCoW para establecer las prioridades del proyecto.
- Y Mobile First y Atomic Design para diseñar las pantallas.

Luego, para la optimización de la solución, aplicamos Lean Startup, que se basa en el circuito: crear, medir, aprender.

- Creamos nuestro prototipo.
- Medimos su resultado.
- Aprendemos de él para realizar cambios si es necesario.

Esto conlleva que los requisitos puedan cambiar en el proceso.

Por eso, utilizamos metodologías ágiles cómo Scrum para gestionar el desarrollo de la aplicación.

→

Scrum

Con Scrum, un producto se basa en una serie de iteraciones llamadas Sprints.

Simplificándolo:

1. Se ordena todo el trabajo en un Product Backlog.
2. Se planifica un Sprint y se crea un Incremento de valor del producto.
3. Se revisan los resultados para ajustar el próximo Sprint.

Para llevar está gestión se utilizan varios artefactos:

Utilizamos un User Story Map, para definir el viaje o casos de uso del usuario en el producto.

Un Scrum Board, para visualizar el trabajo y gestionar el desarrollo.

Las User Stories o Historias de Usuario, serían los requisitos del sistema vistos desde la perspectiva del usuario.

→

DevOps

Además, para agilizar los procesos en el que una nueva funcionalidad pasa del entorno de desarrollo al de producción utilizamos prácticas de DevOps cómo:

Integración Continua (CI) y Despliegue Continuo (CD).

Usando Git para control de versiones, GitHub como repositorio en la nube y GifFlow cómo modelo de creación de ramas.

Que se basa en una rama de producción, una de desarrollo y varias ramas de función.

Y luego GitHub Actions para automatizar los despliegues y Vercel para alojar la aplicación siguiendo el flujo de trabajo DPS: Develop, Preview and Ship.

Que sería el siguiente: Se escribe código, se sube a GitHub y se crea un despliegue automático.

Esto se hace por cada rama, lo que permite tener diferentes despliegues a la vez para recoger feedback y tomar decisiones para producción.

Con ello conseguimos aportar software al cliente de forma continua y sin inactividad del sistema.

→

Stack Tecnológico

Para el desarrollo, se usan las siguientes tecnologías.

- Next.js, que es un framework de React. Que básicamente nos ofrece: Enrutamiento basado en páginas, Prerendering de cada página, etc.. Este usa SWC como compilador y WebPack como empaquetador...
- React que es una biblioteca de JavaScript, basada en programación orientada a componentes, esta usa DOM Virtual y una sintaxis llamada JSX.
- TypeScript que es un superconjunto de JavaScript que básicamente añade tipos estáticos y objetos basados en clases.
- Node.js como entorno de ejecución de JavaScript. Y en este caso usamos Yarn como gestor de paquetes para las dependencias. Ya que NPM no es muy consistente.
- Y luego, VSCode como editor de código, al que se le pueden añadir extensiones. En este caso las más importantes son ESLint y Prettier que nos permiten definir unas Guías de Estilo de código.

→

Arquitectura

La arquitectura de la aplicación esta formada por varias capas desacopladas.

- La Infraestructura que se compone de funciones, librerías y tipos de Typescript que apoyan la lógica de los servicios y de la interfaz.
- Los Servicios, Proveen los datos y alguna lógica independiente a la capa de interfaz.
- La Interfaz que se compone del Contexto y los Hooks que se hacen cargo de crear y manejar los estados necesarios para los Componentes, que son los que renderizan la aplicación.

Vamos a explicar primero los Componentes, luego los Hooks, luego los Servicios, la Infraestructura y por último cómo interactúan entre ellos.

→

Componentes

Estos son todos los componentes de la aplicación que son reutilizados entre páginas. Las páginas también son componentes sólo que están asociados a una ruta.

Tenemos tres páginas que serían Notas, Intervalos y Escalas.

Y tenemos los instrumentos y respuestas de cada ejercicio.

Tenemos por ejemplo, el componente AnswerButtons que serían los botones de respuesta, el componente AnswerToggles que sirve para añadir o quitar respuestas, el botón de play, etc...

Lo importante es que todos son funciones puras que reciben entradas llamadas Props y renderizan la interfaz.

→

Hooks

Luego tenemos los Hooks que son una alternativa de React a usar clases. Estos sirven para encapsular la lógica de estado de los componentes para que puedan ser reutilizados.

También son funciones, ya que pueden reciben entradas, y devuelven los estados, y funciones para actualizarlos.

Estos son algunos de los Custom Hooks de la aplicación y cada uno cubre un interés en específico.

Los más importantes serían:

- useExercise.tsx: que se encarga de cargar las respuestas correspondientes al ejercicio. Notas, Intervalos o Escalas.
- Y EarfitContext.tsx: que sería el contexto de la aplicación. El Context sirve para compartir valores entre componentes sin tener que pasárselos por Props. Este se encarga cargar los instrumentos y seleccionarlos. Se llama con el Hook useInstrumentContext().

→

Servicios

Luego, tenemos los servicios que proveen los datos a los Hooks y al Context.

Cada uno provee una cosa.

El instrumentService provee los instrumentos y los demás proveen las respuestas correspondientes a cada variante de ejercicio.

→

Infraestructura

Luego, tenemos la Infraestructura donde podemos ver los tipos y librerías más importantes.

En los tipos tenemos a:

- Los Instrumentos: que tienen un id, un noteplayer que es un tipo con funciones para reproducir notas y un nombre.
- Y las Respuestas: que tienen un id para compararlas, un array de notas para tocar y un nombre para mostrar.

Luego, en la librerías tenemos:

- Tonal.js: para manipular elementos musicales. Esta se usa en los servicios, para generar las notas, intervalos y escalas.
- Soundfont-player: que sirve para cargar y reproducir archivos MIDI.js. Estos archivos es donde están los sonidos de los instrumentos codificados.
- Soundfont-wrapper: Que es una librería que he creado para simplificar el uso de la anterior. Esta se encarga de crear los Noteplayer para cada instrumento

aparte de más cosas.

→

Comportamiento de una Página

Ahora vamos a ver cómo interactúan entre sí, explicando el Comportamiento de una Página:

Tenemos los Servicios, los Hooks y los Componentes.

Para los Instrumentos:

- Desde los componentes se llama al Hook useInstrumentContext().
- Este hace uso de la función getInstruments() del servicio para obtener los instrumentos.
- Crea los estados y devuelve el array de instrumentos, el instrumento seleccionado o funciones que necesiten los componentes.

Para las Respuestas:

Tenemos las variantes de ejercicio, notas, intervalos y escalas. Todas funcionan igual.

- Desde la página se llama al Hook useExercise con la variante que queremos usar.
- Este hace uso de la función getNotes(), getIntervals() o getScales() de los servicios para obtener las respuestas correspondientes.
- Estas se les pasan a los demás Hooks. Que crearán los estados y funciones necesarias para los componentes.

Por último, los componentes renderizarán la interfaz con estos datos y harán uso de estas funciones para actualizarse.

Todo apoyado por una Infraestructura de funciones, librerías y tipos.

→

Progressive Web App

La aplicación es una aplicación web progresiva.

Esto quiere decir que es confiable e instalable como una App Nativa en PC, Móvil y Tablet.

Para ello tiene una una conexión segura HTTPS.

Funciona offline, gracias a un Service Worker que es un script para manejar las solicitudes red y la caché.

Y también tiene información sobre la aplicación en un archivo JSON llamado Manifest.

Y un icono PNG.

→

Software QA

Por último, estas son algunas pruebas de calidad del software.

Para ello usamos las herramientas Google Lighthouse y Vercel Analytics.

Google Lighthouse permite medir el rendimiento, la accesibilidad, buenas prácticas, seo y si la aplicación es progresiva. Estas son las puntaciones de estos apartados.

El apartado performance no lo tenemos en cuenta aunque tenga buena puntuación porque Lighthouse da algún problema para aplicaciones de Nextjs.

Para eso usamos Vercel Analytics, que además los datos que recoge son de los dispositivos de los usuarios reales no de una simulación en el ordenador.

Este mide las Web Vitals que son unas métricas establecidas por Google y que son:

- La velocidad de carga del primer contenido de la página.
- La velocidad de carga de todo el contenido de la página.
- Cuánto se mueven los elementos después de mostrarse al usuario.
- La capacidad de respuesta de la página, o cuánto tiempo esperan los usuarios para ver la reacción de su primera interacción con la página.

Estas métricas se analizan cada día lo que permite tener un flujo continuo de mediciones en el tiempo y detectar cambios nuevas implementaciones.

→

Conclusiones

Cómo hemos visto, hemos desarrollado una PWA que permite desarrollar el oído musical mediante

el entrenamiento auditivo.

Que implementa diferentes tipos de ejercicios personalizables.

Y que incluye varios instrumentos para practicar con sus sonidos.

Todo ello intentando aplicar buenas prácticas como guías de estilo de código, principios de clean code, una gestión ágil del desarrollo y prácticas de DevOps.

Cómo trabajos futuros se proponen algunas de las funcionalidades que no se establecieron como prioridades en la etapa de Design Thinking que son Ejercicios de acordes, ritmos, progresiones, un modo nocturno e incluir varios idiomas.

→

Demostración

Y ahora vamos a ver un video de demostración de cómo funciona la aplicación.