

# EARFIT: Aplicación Para Entrenamiento Auditivo Musical Basada en Next.js y TypeScript

Trabajo fin de grado

**Grado en Ingeniería Informática**

Autor: *Alberto Gómez Cano*

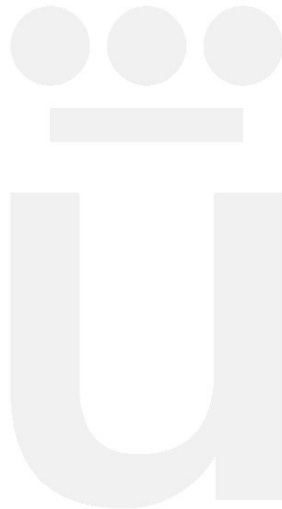
Tutor: *Manuel Rubio Sánchez*



Universidad  
Rey Juan Carlos

Escuela Técnica Superior  
Ingeniería Informática

# Contenido



## 1 Introducción

## 2 Metodologías

- Design Thinking
- Lean Startup
- Scrum
- DevOps

## 3 Desarrollo

- Tecnologías
- Arquitectura
- Estructura de Archivos
- Detalles de Implementación
- Progressive Web App
- Software QA

## 4 Conclusiones



# Resumen

- Herramienta para ayudar a músicos a desarrollar su oído (Musical Ear Training).
- EARFIT es una PWA basada en Next.js y TypeScript.
- Desarrollada bajo metodologías ágiles y desplegada en Vercel.
- Se basa en un conjunto de ejercicios de entrenamiento auditivo.



# Entrenamiento Auditivo

- Es el proceso de identificar y asociar los elementos musicales con la forma en que se siente el sonido físicamente.
- Los músicos, productores y DJs pueden beneficiarse del entrenamiento auditivo.
- Permite sacar canciones más rápido, con mayor precisión, improvisar mejor y llevar al instrumento las melodías que imagines con mayor facilidad.
- Los ejercicios más comunes incluyen habilidades como identificar notas, intervalos, escalas...



# Objetivos

- **Objetivo principal:**

- Crear una aplicación que permita a músicos desarrollar su oído musical mediante el entrenamiento auditivo.

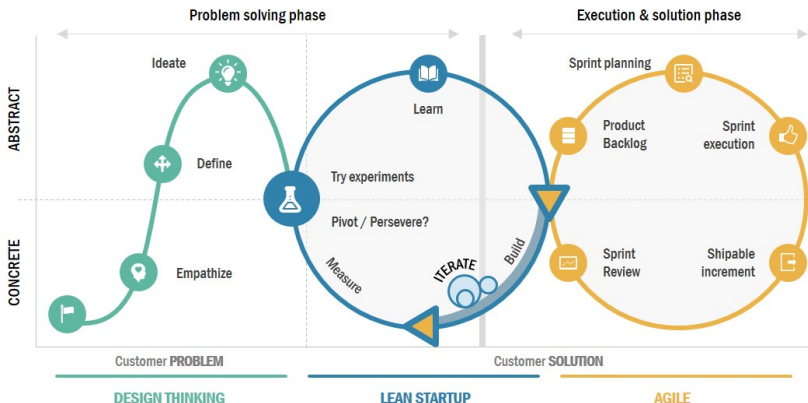
- **Subobjetivos:**

- Desarrollar una interfaz interactiva.
- Implementar diferentes tipos de ejercicio personalizables.
- Incluir varios instrumentos para practicar con sus sonidos.



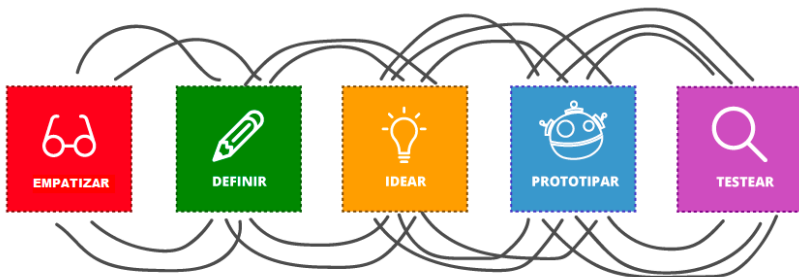
# Metodologías

## Proceso Combinado de Design Thinking, Lean Startup, Scrum y DevOps



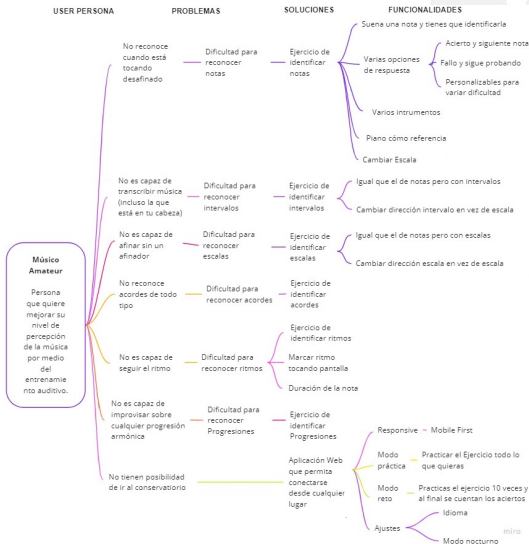
# Design Thinking

## Generar Ideas Innovadoras



# Mindmap (1/2)

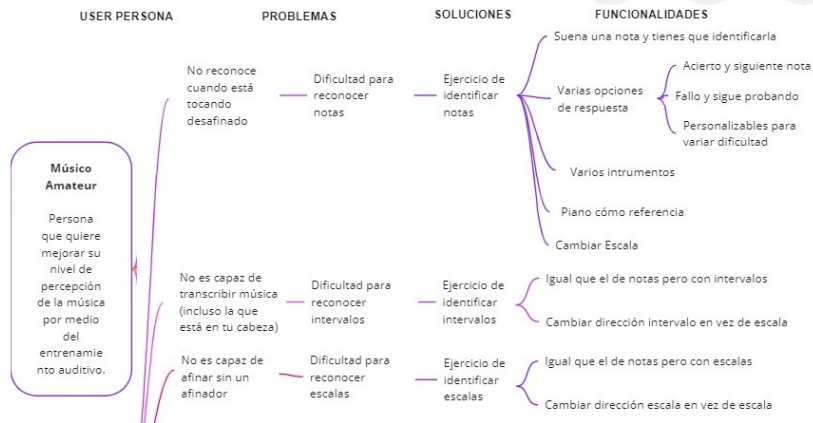
## Representar Ideas o Conceptos y Encontrar Soluciones





# Mindmap (2/2)

## User Persona, Problemas y Soluciones (Hipótesis)



# MoSCoW (1/2)

## Establecer las Prioridades del Proyecto



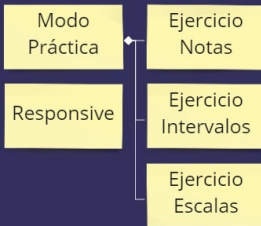
# MoSCoW (2/2)

## Must Have y Should Have

Esto es lo que se va a desarrollar

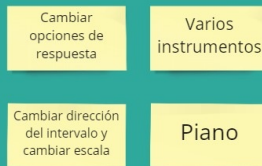
### MUST HAVE

(Debe Tener): Características absolutamente críticas para el nuevo proyecto, sin ellas es un fracaso.



### SHOULD HAVE

(Debería Incluir): Aspectos del proyecto críticos también, pero no imprescindibles.



# Prototipo

## Mobile First y Atomic Design



Notes

Intervals

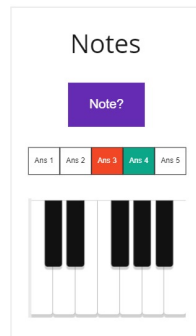
Scales

About



Notes

Note?



Scale

Major

Answers

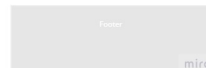
Ans 1 ☒ Ans 2 ☒ Ans 3 ☒

Ans 4 ☒ Ans 5 ☒ Ans 6 ☐

Ans 7 ☐

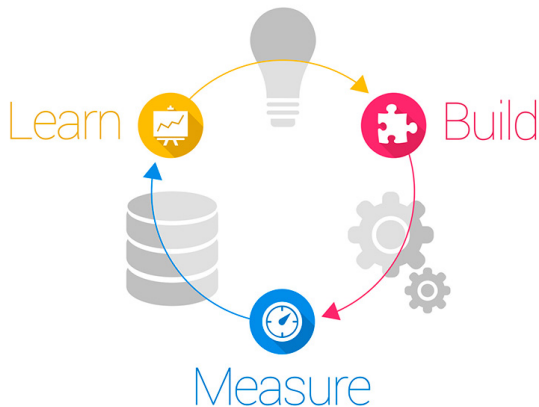
Instrument

☒ Piano ☒ Guitar



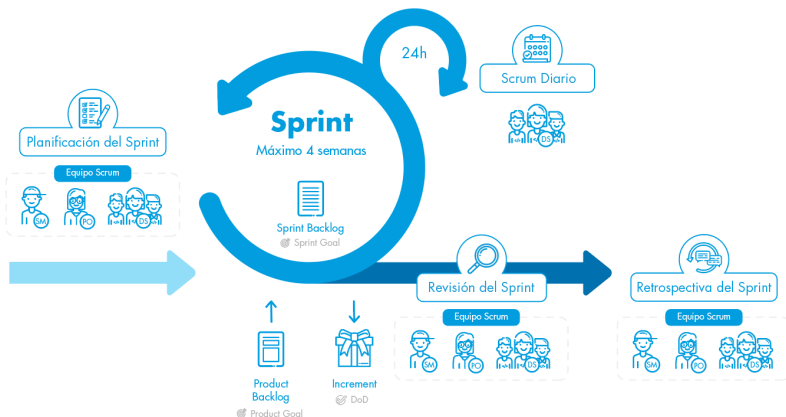
# Lean Startup

## Puesta en Marcha y Optimización de la Solución



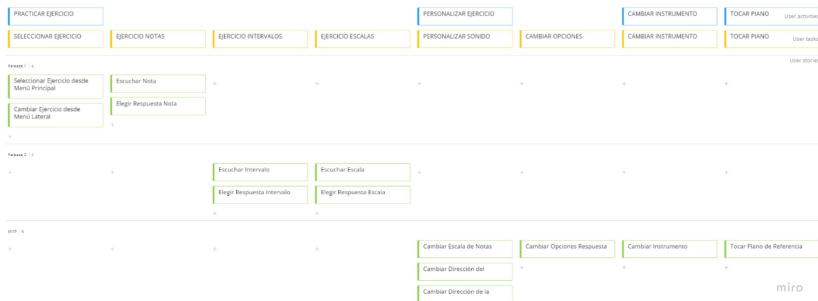
# Scrum

## Proceso de Gestión del Desarrollo



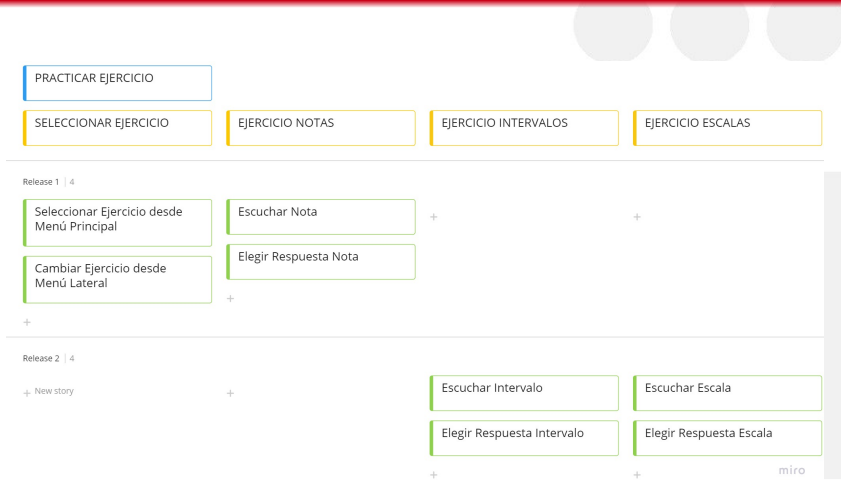
# User Story Map (1/2)

## Definir el Viaje o Casos de Uso del Usuario en el Producto



# User Story Map (2/2)

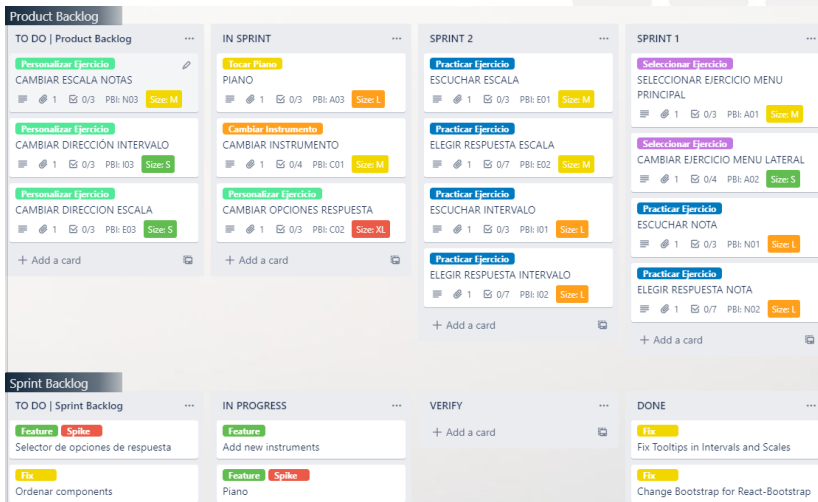
## User Activities, User Tasks, User Stories y Releases





# Scrum Board

## Visualizar el Trabajo y Gestionar el Desarrollo (Product Backlog y Sprint Backlog)



# User Stories

## Características o Requisitos del Sistema desde la Perspectiva del Usuario


**CAMBIAR INSTRUMENTO**

in list [IN SPRINT](#)

Labels
 

Cambiar Instrumento

+


**Description**

Edit

Como: Músico Amateur  
 Quiero: Cambiar entre varios instrumentos  
 Para: Practicar los ejercicios con sus sonidos


**Custom Fields**

T	PBI	Size
	C01	M


**Attachments**



**Prototipo** ↗  
 Added yesterday at 8:00 PM - [Comment](#) - [Delete](#)  
 [Make cover](#)


**Criterios de Aceptación**

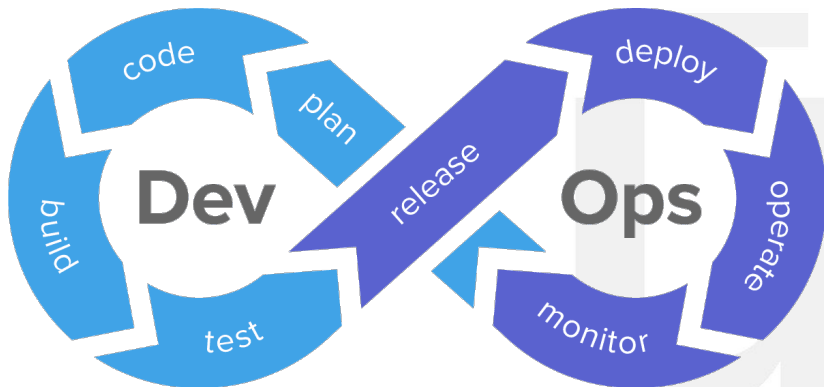
0%

- ☐ Dado: Un selector de instrumentos en la sección de opciones situada a la derecha del ejercicio
- ☐ Cuando: Pulses sobre un botón de instrumento
- ☐ Entonces: El sonido que emita el botón de play será el del instrumento seleccionado
- ☐ Condiciones: Siempre tiene que haber un instrumento seleccionado



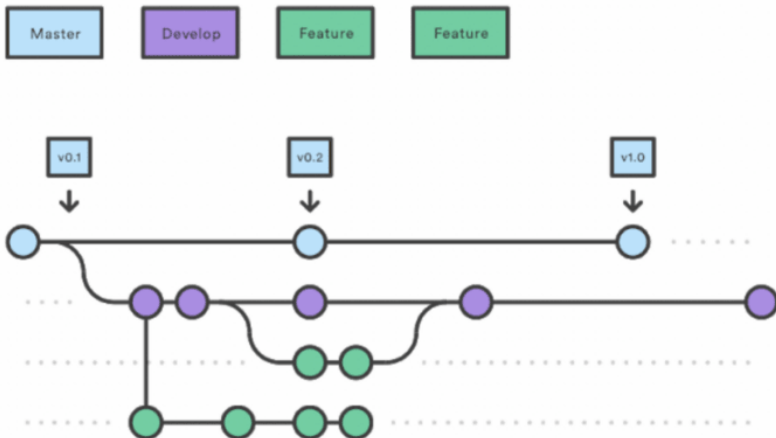
# DevOps

Agilizar los Procesos del Entorno de Desarrollo al de Producción



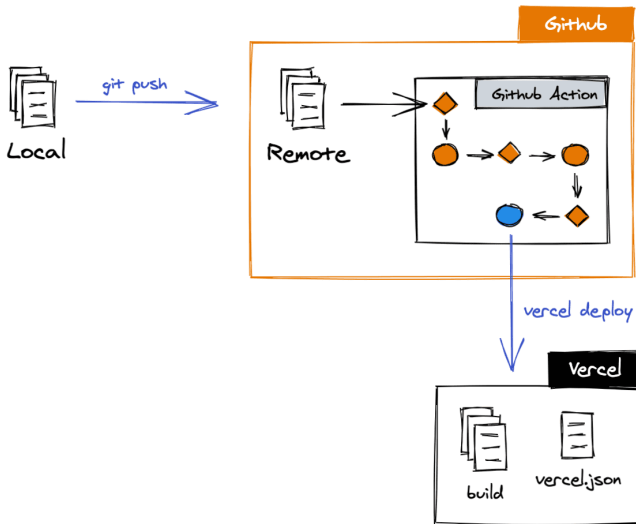
# Integración Continua (CI)

Git, GitHub y GitFlow



# Despliegue Continuo (CD)

## GitHub Actions, Vercel y Flujo DPS



# Stack Tecnológico

NEXT.js



TS

node.js

▲ Vercel

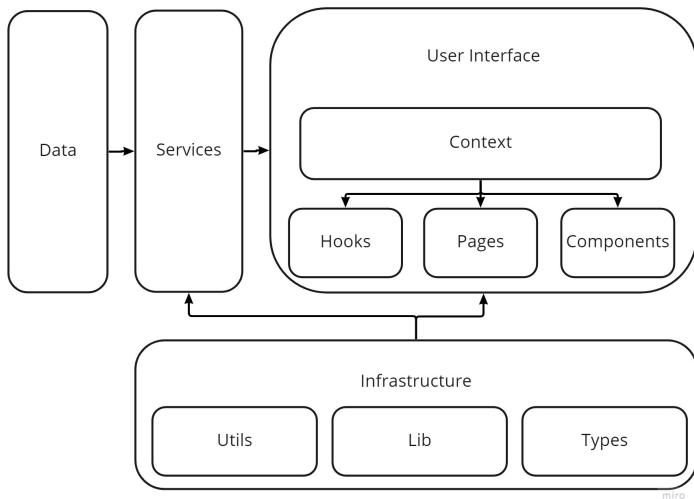


- **Next.js:** Enrutamiento basado en páginas, Prerendering, Code Splitting, Prefetching, Fast Refresh, SWC y WebPack...
- **React:** Componentes, DOM Virtual, Hooks, Context...
- **Typescript:** Tipos estáticos para JavaScript.
- **Node.js:** Entorno de ejecución, librerías (NPM vs. Yarn).
- **VScode:** Git integrado, extensiones (ESLint y Prettier).



# Arquitectura


## Estructura, Funcionamiento e Interacción




# Estructura de Archivos


## Pages y Public son Directorios Especiales en Next.js


### ▼ EARFIT

>  public


### ▼ src


>  components


>  context


>  hooks

>  lib


>  pages


>  services


>  styles


>  types


>  utils


 .eslintrc.js


 .gitignore


 .prettierrc.js


 next-env.d.ts

 next.config.js

 package.json

 README.md

 tsconfig.json

 yarn.lock





# Tipos de TypeScript (1/2)

Cada Instrumento tiene un NotePlayer para Reproducir Notas

```
export type Instrument = {  
  displayName: string;  
  emoji: string;  
  instrumentName: InstrumentName;  
  notePlayer: NotePlayer;  
  isLocal: boolean;  
};
```

```
type NotePlayer = {  
  play: (noteName: string, when?: number, opts?: {duration?: number; gain?: number;  
    attack?: number; decay?: number; sustain?: number; release?: number; adsr?:  
    [number, number, number, number]; loop?: boolean;}) => Player;  
  stop: (noteName?: string) => void;  
  schedule: (when?: number, events?: any[]) => Player;  
  instrumentName: InstrumentName;  
};
```



# Tipos de TypeScript (2/2)

Dependiendo de la Variante, las Respuestas tienen Notas, Intervalos o Escalas

TypeScript

Copy Caption ...

```
export type VariantExercise = 'notes' | 'intervals' | 'scales';
```

```
export type Answer = {  
  id: string;  
  notes: string[];  
  displayName: string;  
};
```

```
export type IsSelectable = {isSelected: boolean};  
export type VariantColor = 'success' | 'secondary' | 'danger';  
export type WithColor = {color: VariantColor};
```

```
export type SelectableAnswer = Answer & IsSelectable;  
export type SelectableAnswerWithColor = SelectableAnswer & WithColor;
```



# Librerías

## Soundfont-wrapper crea los Noteplayer para Cada Instrumento

- **Tonal.js:** Manipular elementos musicales.
- **Soundfont-player:** Cargar y reproducir archivos MIDI.js.
- **Soundfont-wrapper:** Refinar la complejidad de “soundfont-player” y simplificar su uso.
- **React-piano:** Teclado de piano interactivo (sin sonidos).
- **React-use-measure:** Para que el piano sea responsive.
- **React-bootstrap:** Librería de estilos CSS.
- **Next-pwa:** Registrar y generar un Service Worker.



# Servicios

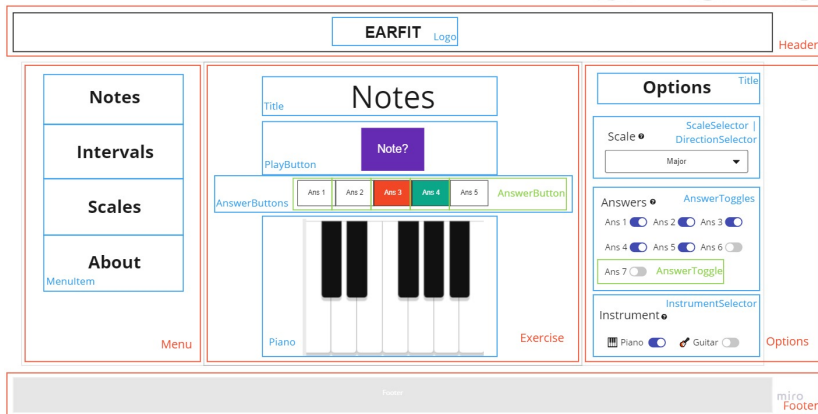
## Proveen los Datos y Alguna Lógica a los Hooks y Context

- **instrumentService.ts**: Provee los instrumentos de la aplicación.
- **noteService.ts**: Provee las respuestas correspondientes al ejercicio de notas.
- **intervalService.ts**: Provee las respuestas correspondientes al ejercicio de intervalos.
- **scaleService.ts**: Provee las respuestas correspondientes al ejercicio de escalas.



# Jerarquía de Componentes

Los Componentes Son Reutilizados entre Páginas



# Comportamiento de una Página (1/2)

## Los Custom Hooks Encapsulan la Lógica de Estado

```
import { Exercise, Menu, Options, PageLayout } from 'components';
import { useAnswer, useAnswerButtons, useAnswerToggles, useDirectionSelector,
useExercise, usePlayButton } from 'hooks';

export default function Intervals(): JSX.Element {
  const { direction, changeDirection } = useDirectionSelector();
  const { answers } = useExercise('intervals');
  const { answerToggles, updateIsSelected, selectAllOrThree } = useAnswerToggles(answers);
  const { answer, setNewAnswer, isCorrectAnswer } = useAnswer('intervals', answerToggles);
  const { playAnswer } = usePlayButton('intervals', answer, direction);
  const { answerButtons, handleAnswerButtonClick, streak } = useAnswerButtons(answerToggles,
    isCorrectAnswer,
    setNewAnswer,
    playAnswer
  );
};
```



# Comportamiento de una Página (2/2)

## Los Componentes reciben Estados y Callbacks a Través de Props

```
return (  
  <PageLayout  
    leftCol={<Menu />}  
    rightCol={  
      <Options  
        direction={direction}  
        handleDirectionChange={changeDirection}  
        answerToggles={answerToggles}  
        handleAnswerToggleAllChange={selectAllOrThree}  
        handleAnswerTogglesChange={updateIsSelected}  
      />  
    }  
  >  
    <Exercise  
      title="Intervals"  
      playButtonLabel="Interval?"  
      handlePlayButtonClick={playAnswer}  
      answerButtons={answerButtons}  
      handleAnswerButtonClick={handleAnswerButtonClick}  
      streak={streak}  
    />  
  </PageLayout>  
);
```



# Comportamiento de un Componente

## Los Componentes Actúan como Funciones Puras

```
import { AnswerButton } from 'components/Exercise/AnswerButtons/AnswerButton';
import { ButtonGroup } from 'react-bootstrap';
import { SelectableAnswerWithColor } from 'types';

interface Props {
  answerButtons: SelectableAnswerWithColor[];
  handleAnswerButtonClick: (answerButton: SelectableAnswerWithColor) => void;
}

export const AnswerButtons = ({ answerButtons, handleAnswerButtonClick }: Props):
  JSX.Element => {
  return (
    <ButtonGroup className="btn-group btn-group-toggle d-flex justify-content-center"
      data-toggle="buttons">
      <div>
        {answerButtons.map((answerButton) => (
          <AnswerButton
            key={answerButton.id}
            answerButton={answerButton}
            handleAnswerButtonClick={handleAnswerButtonClick}
          />
        ))}
      </div>
    </ButtonGroup>
  )
}
```





# Progressive Web App (1/3)

Confiable e Instalable como una App Nativa en PC, Móvil y Tablet

Para que una aplicación sea PWA debe tener:

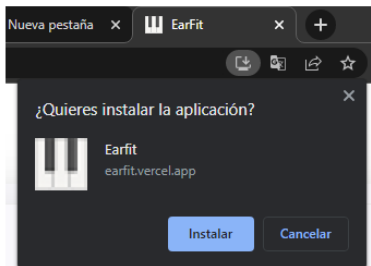
- Una conexión segura HTTPS.
- Cargue sin conexión, para ello requiere un Service Worker.
- Información como nombre, autor, icono y descripción en un documento JSON llamado Manifest.
- Un icono de al menos 144x144 px en formato PNG.



# Progressive Web App (2/3)

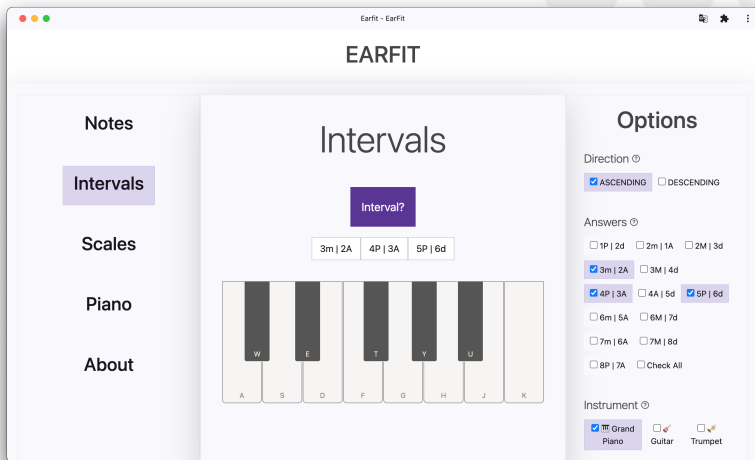
## ¿Cómo se Instala?

En Safari, la opción se llama “añadir a pantalla de inicio” y en Chrome aparece un icono en la barra de búsqueda.



# Progressive Web App (3/3)

## Earfit como PWA en MacOs



# Google Lighthouse

## Auditoría de Calidad de la Página Web (Web Vitals)



Performance



Accessibility



Best Practices



SEO



PWA

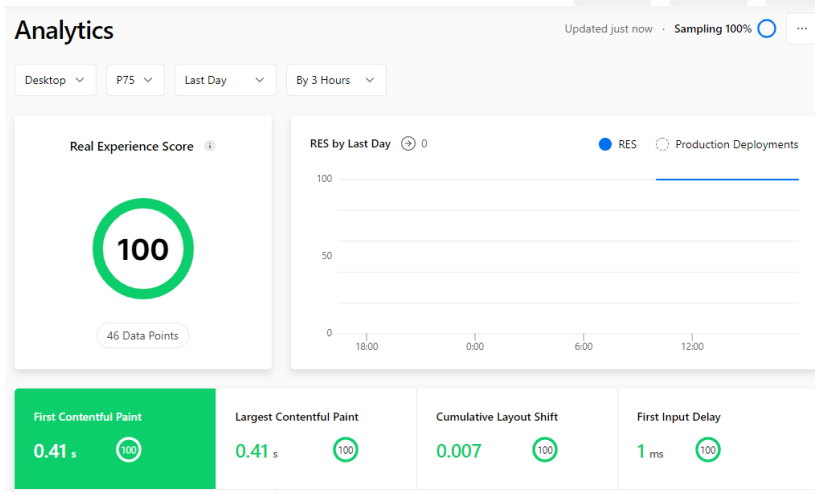
En el apartado Performance:

- Lighthouse da falsas mediciones para aplicaciones Next.js.
- Lighthouse estima las Web Vitals ejecutando una simulación.
- En este caso, usar Vercel Analytics aporta ventajas, como datos reales de los dispositivos de los usuarios.



# Vercel Analytics

## Experiencia de Usuario de la Página Web



# Conclusiones

## Objetivos Alcanzados:

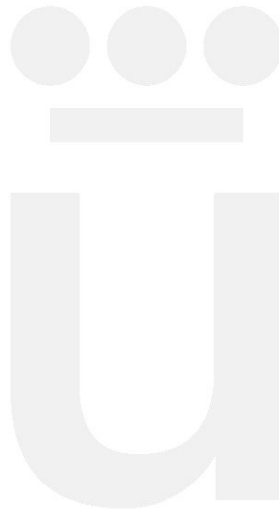
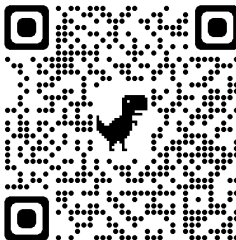
- OBJ1
- OBJ2

## Trabajos Futuros:

- Funcionalidades: Ejercicios de acordes, ritmos, progresiones, modo nocturno y varios idiomas.
- Añadir cobertura de código y test automáticos al flujo de Integración Continua (CI)
- Considerar Software libre vs Negocio: Plantear alguna forma de monetización, marketing, etc.



# Video de demostración



# EARFIT: Aplicación Para Entrenamiento Auditivo Musical Basada en Next.js y TypeScript

Trabajo Fin de Grado

**Grado en Ingeniería Informática – Curso 2021-2022**

Autor: *Alberto Gómez Cano*

Tutor: *Manuel Rubio Sánchez*



Universidad  
Rey Juan Carlos

Escuela Técnica Superior  
Ingeniería Informática