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

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

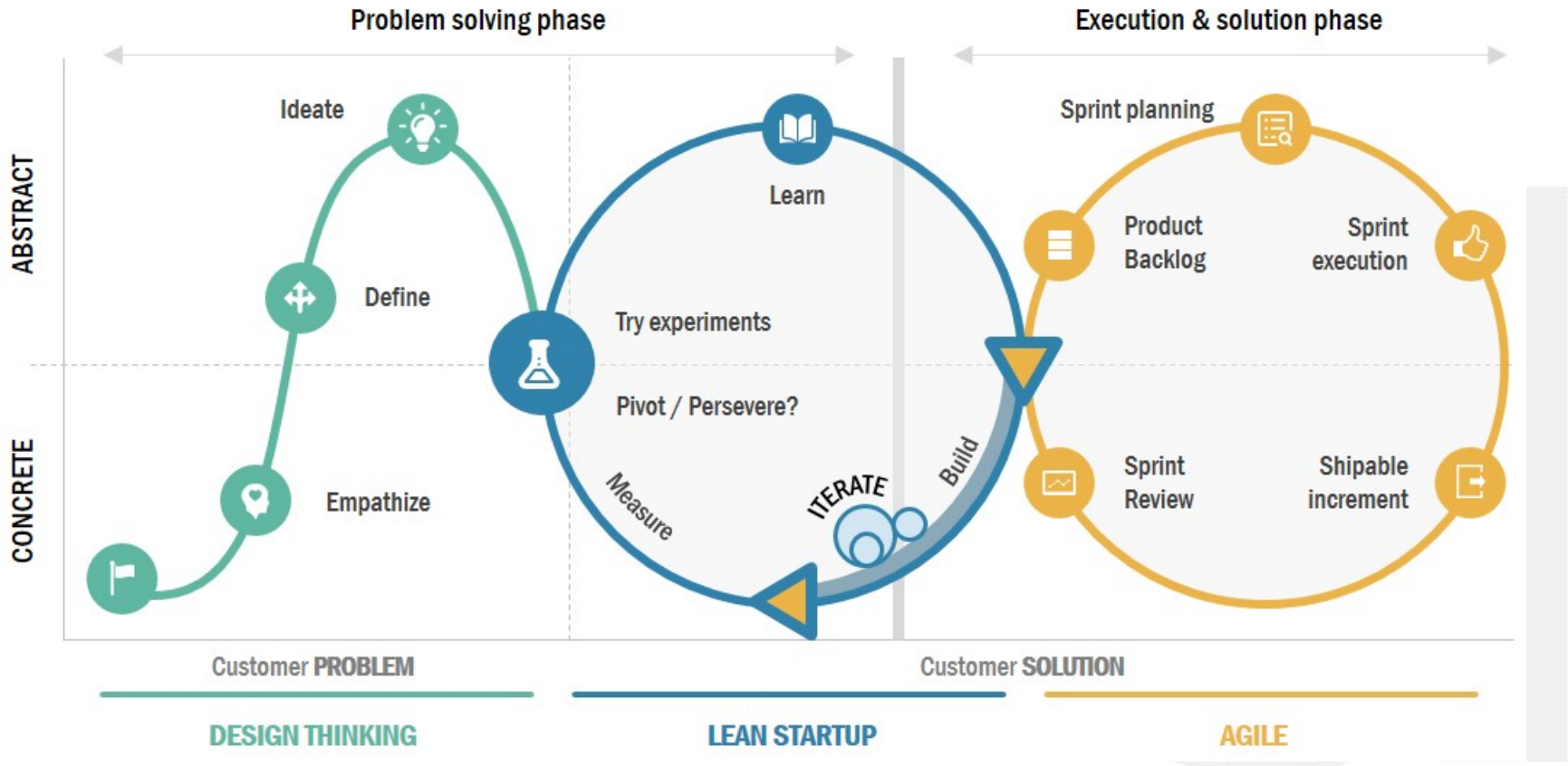
- Es el proceso de identificar y asociar los elementos musicales con la forma en que se percibe el sonido.
- Los músicos, productores y DJs pueden beneficiarse del entrenamiento auditivo.
- Permite sacar canciones más rápido, con mayor precisión, improvisar mejor y llevar al instrumento las melodías 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 prácticar con sus sonidos.

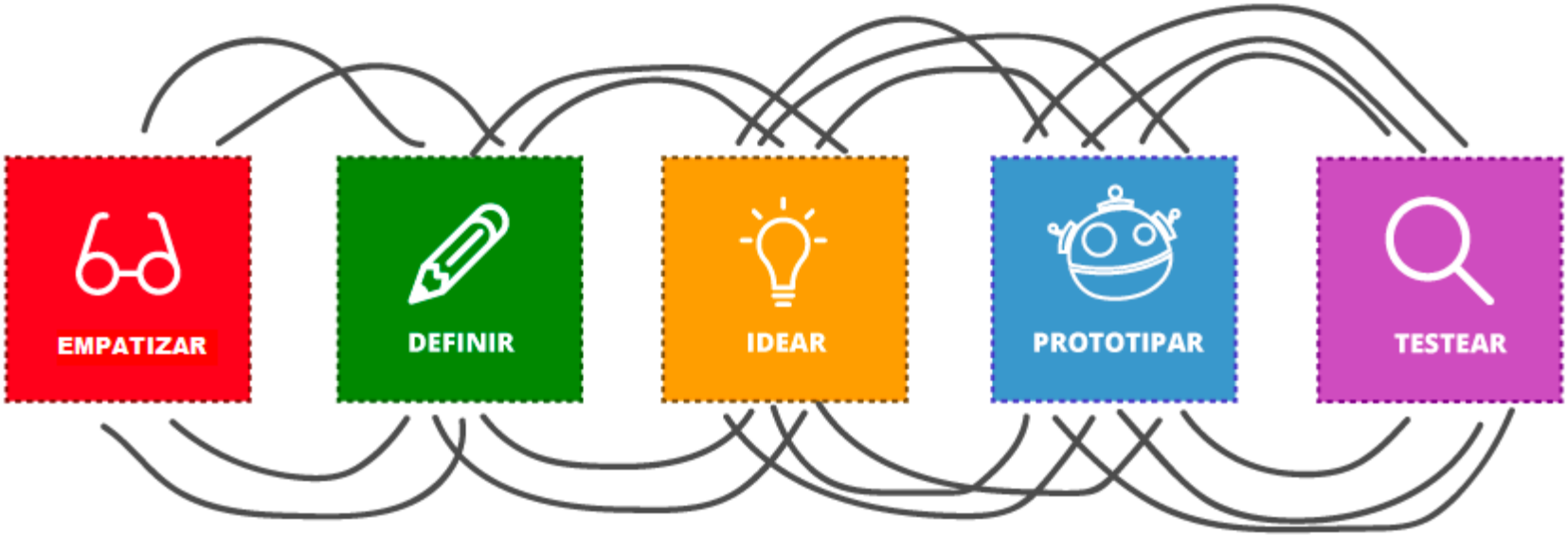
Metodologías

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



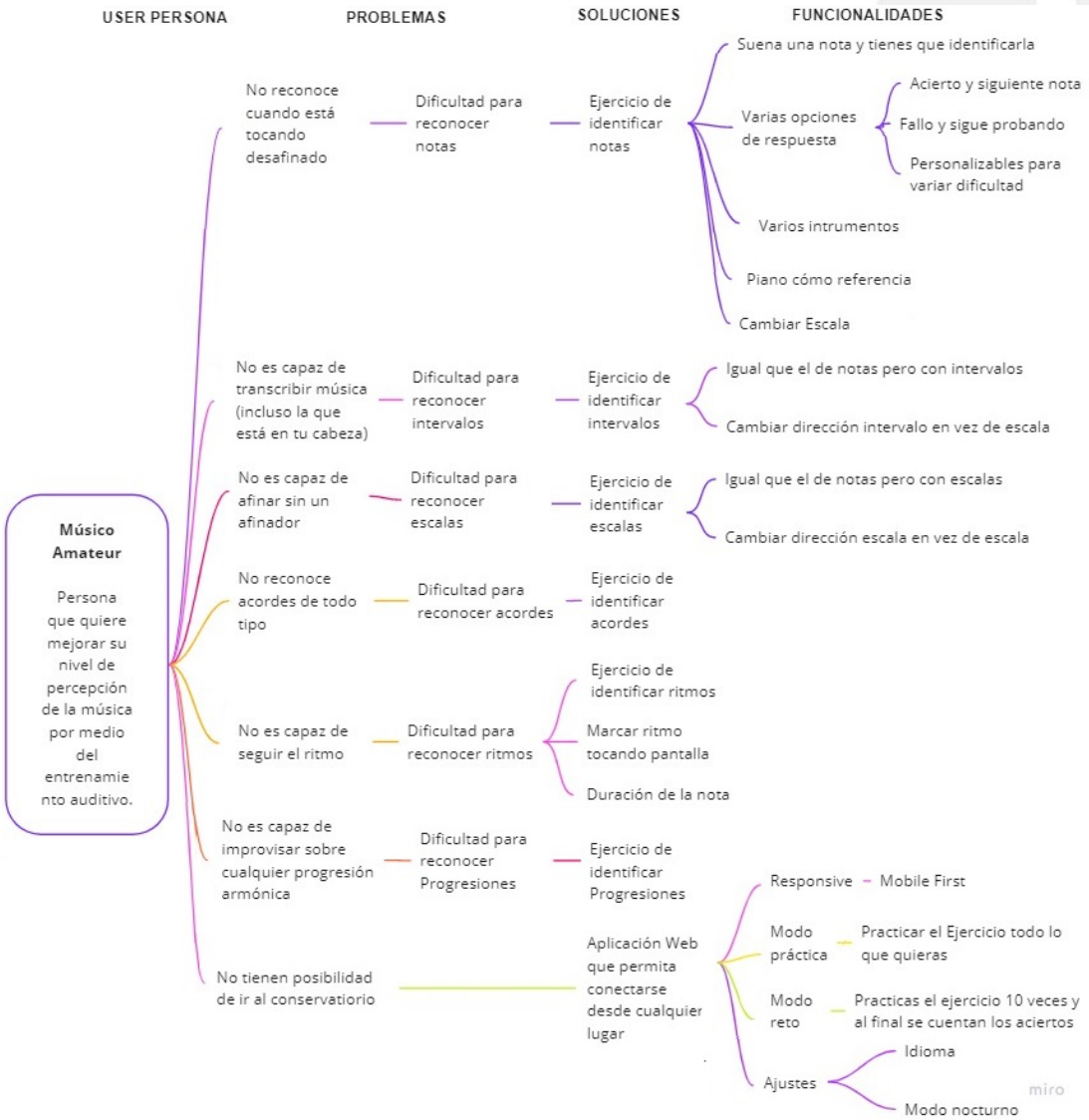
Design Thinking

Generar Ideas Innovadoras



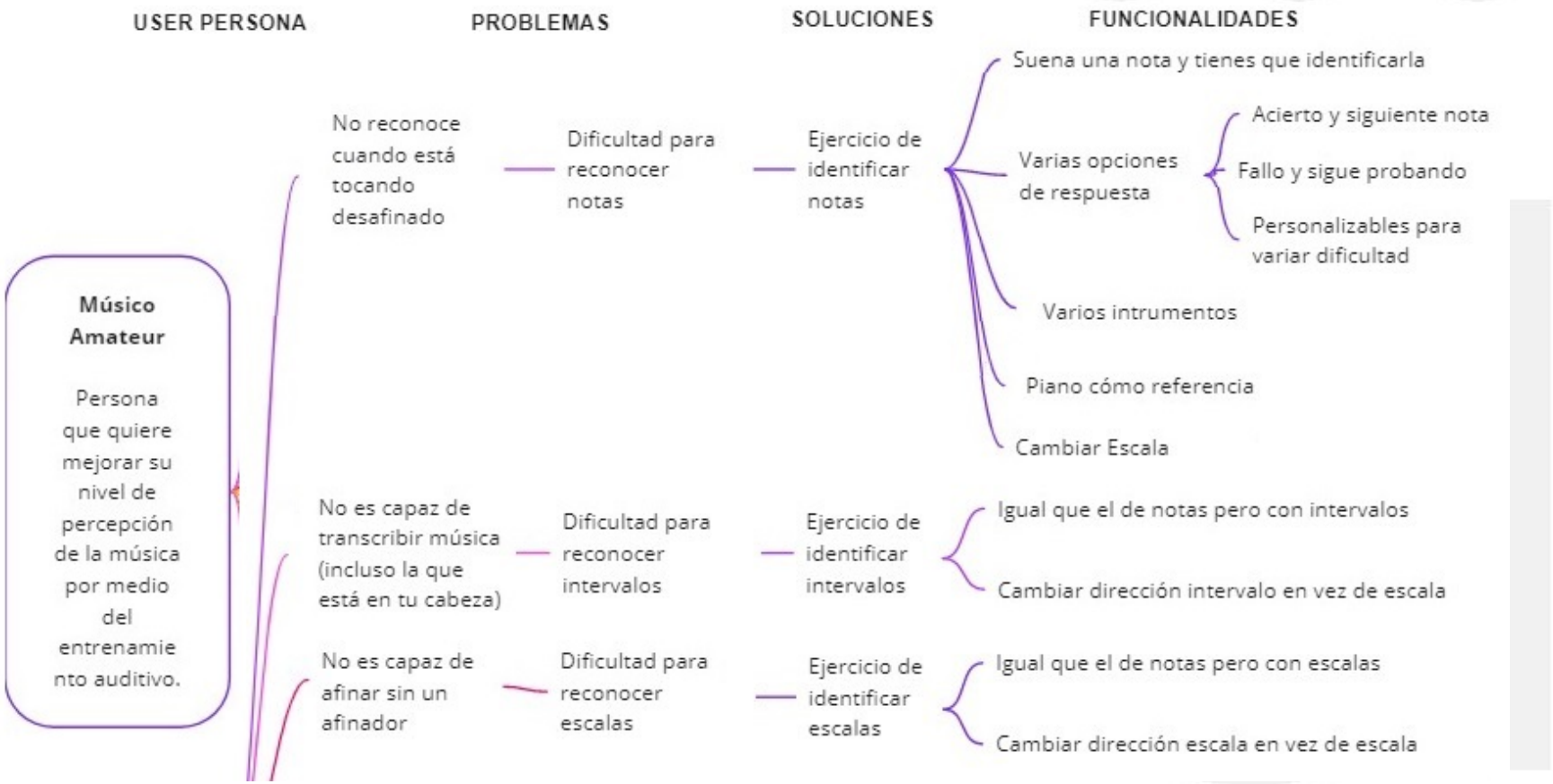
Mindmap (1/2)

Representar Ideas o Conceptos y Encontrar Soluciones



Mindmap (2/2)

User Persona, Problemas y Soluciones (Hipótesis)

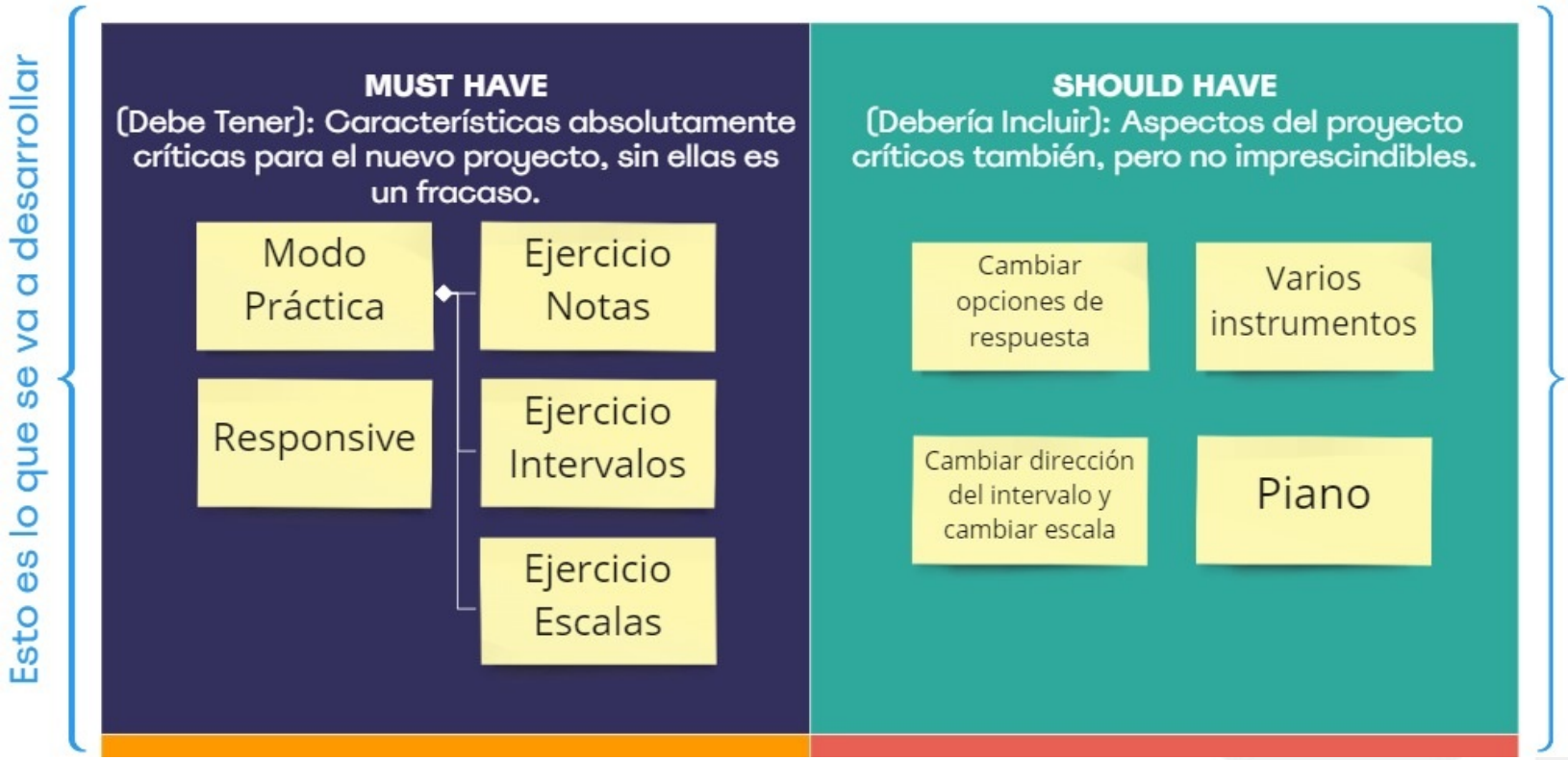


MoSCoW (1/2)

Establecer las Prioridades del Proyecto

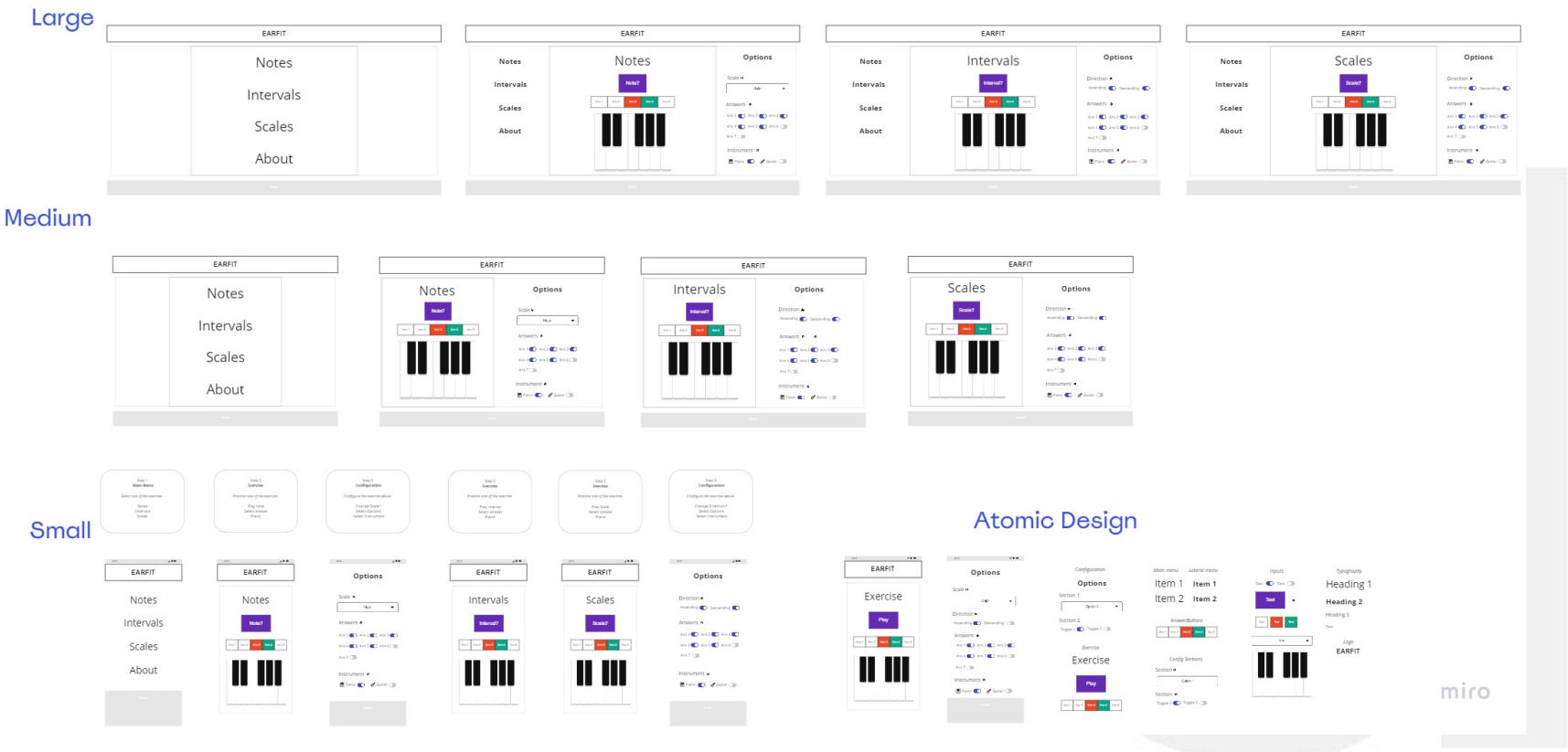
MoSCoW (2/2)

Must Have y Should Have



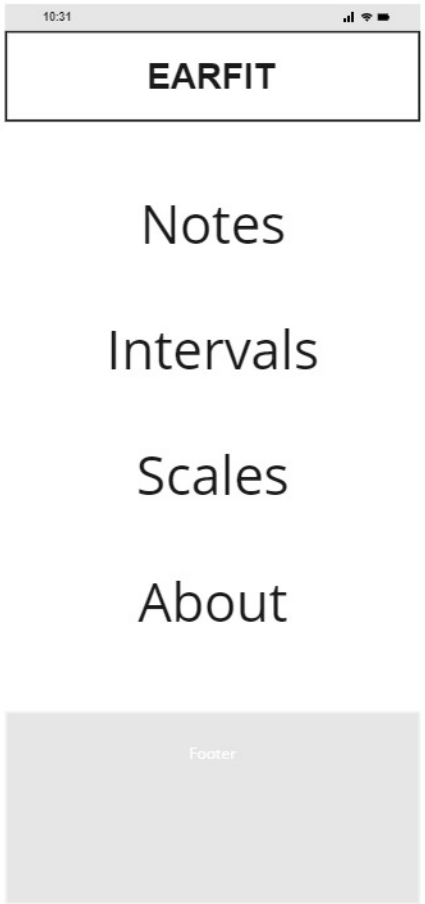
Prototipo (1/2)

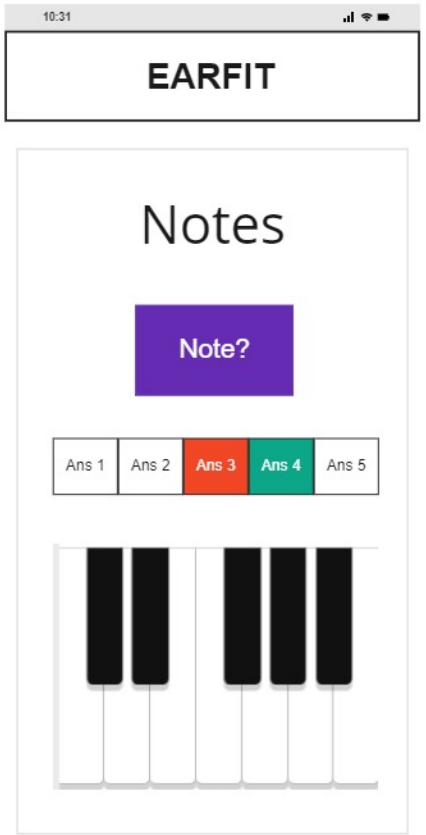
Mobile First y Atomic Design

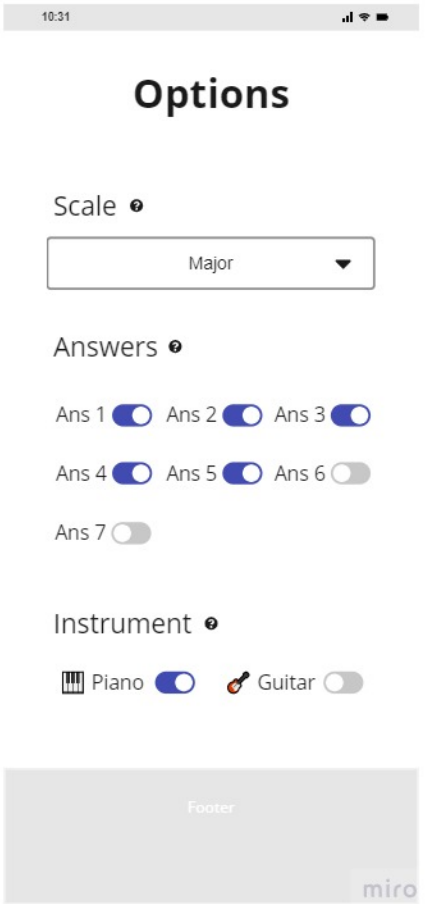


Prototipo (2/2)

Pantallas Pequeñas

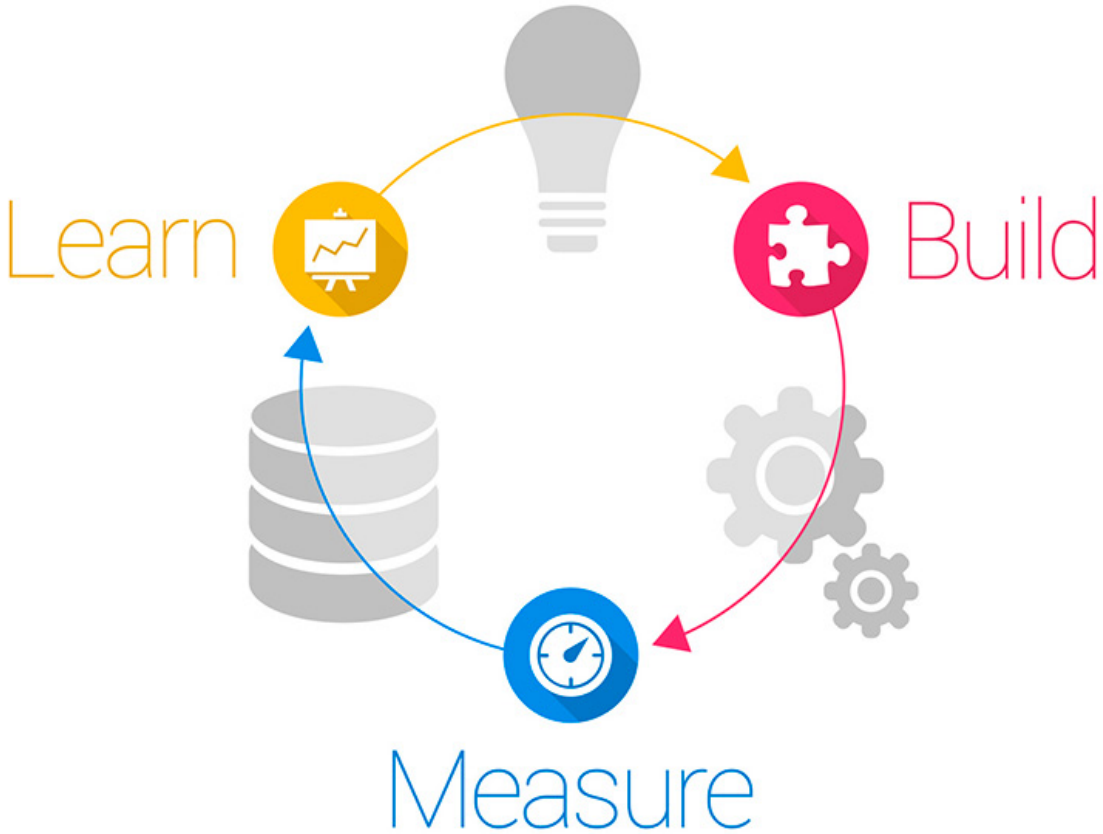






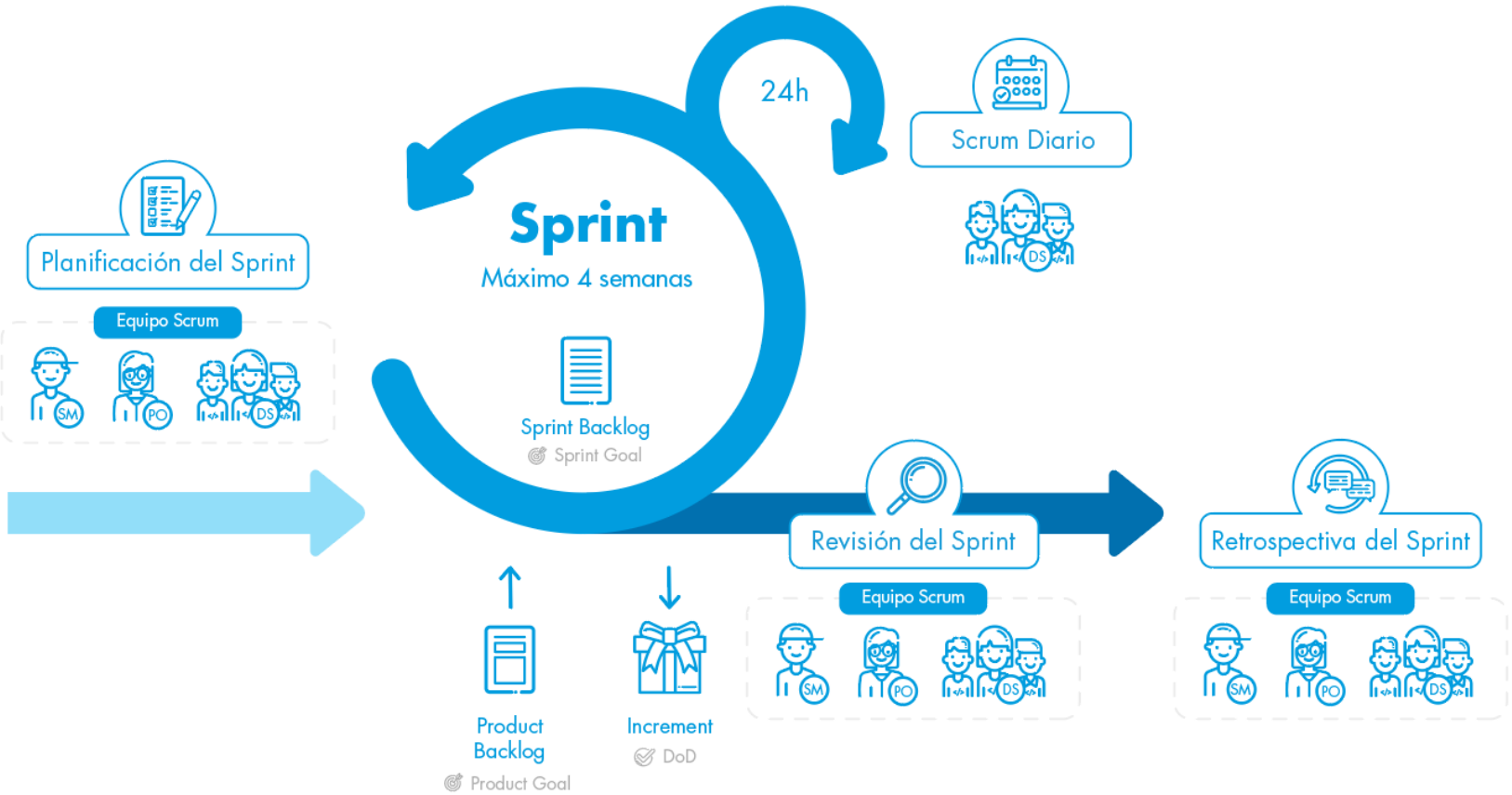
Lean Startup

Puesta en Marcha y Optimización de la Solución



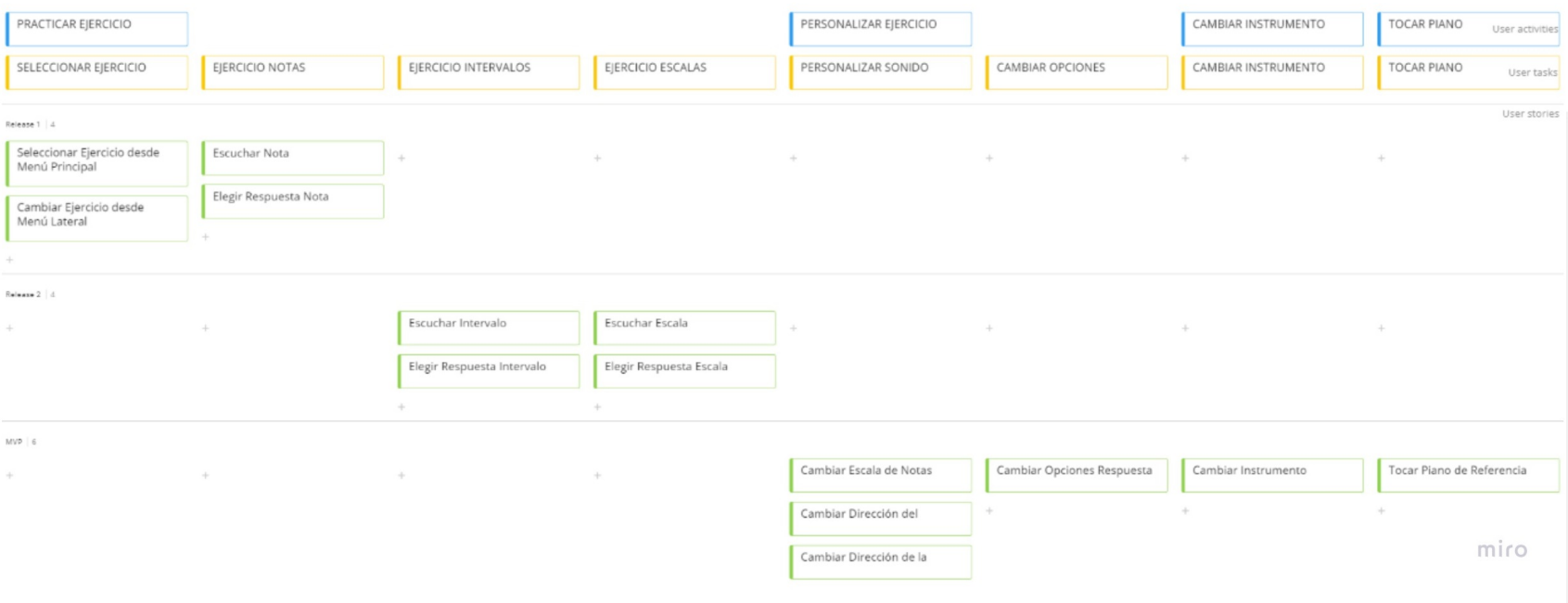
Scrum

Proceso de Gestión del Desarrollo



User Story Map (1/2)

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

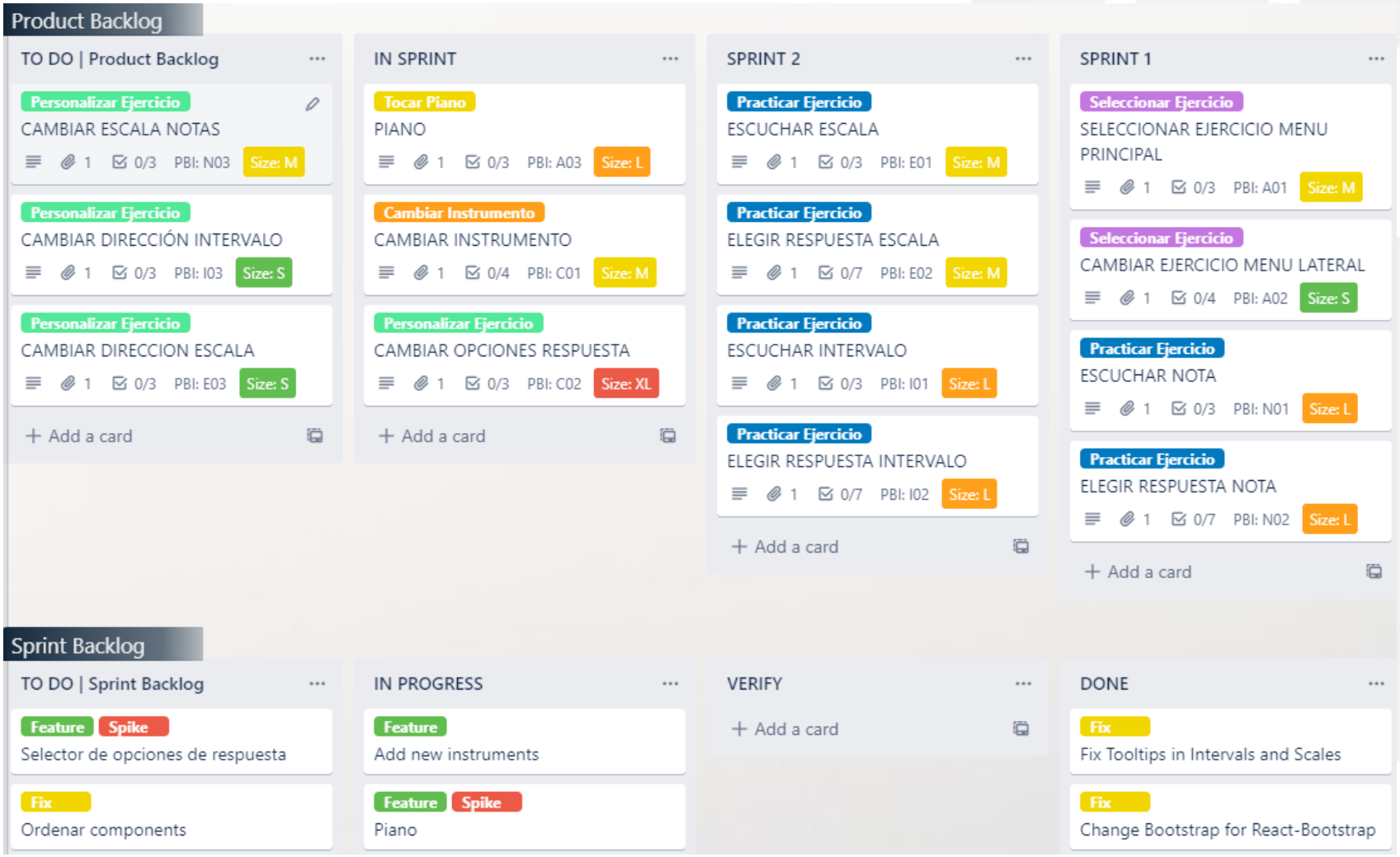


User Story Map (2/2)

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 +

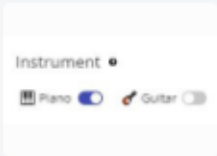
Description Edit

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

Custom Fields

PBI	Size
C01	M

Attachments



Prototipo ↗
Added yesterday at 8:00 PM - Comment - Delete
Make cover

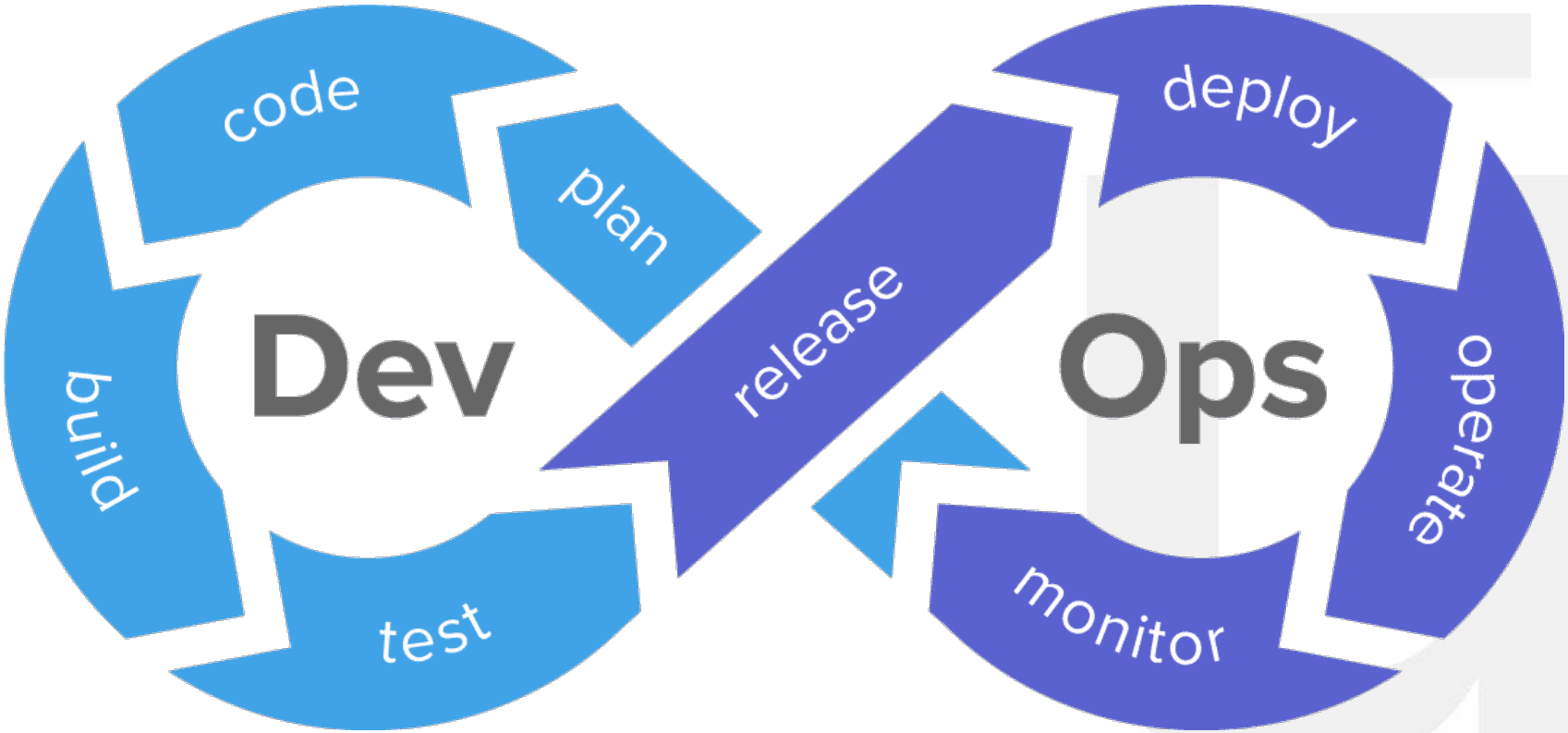
Criterios de Aceptación

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

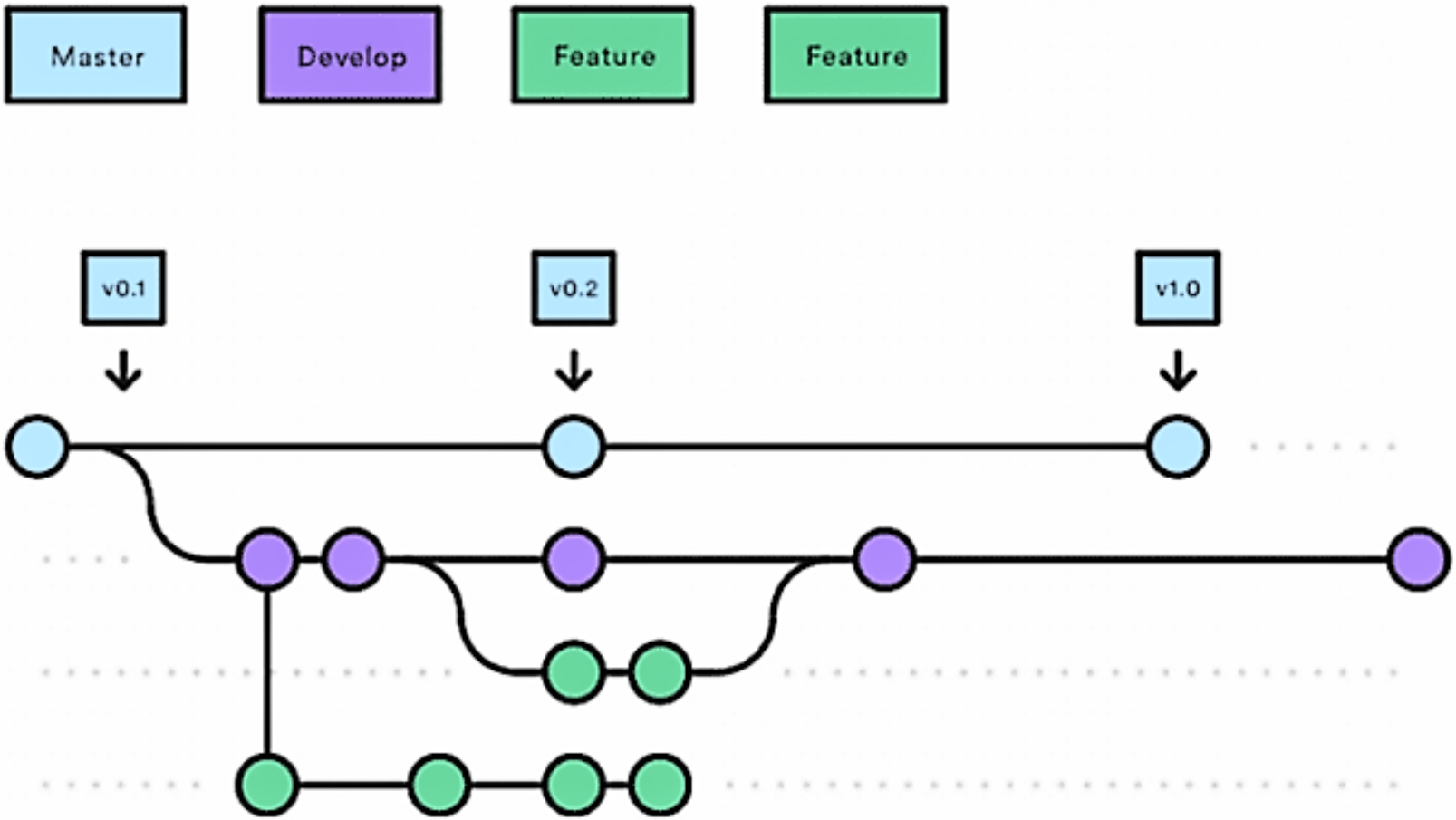
DevOps

Agilizar los Procesos del Entorno de Desarrollo al de Producción



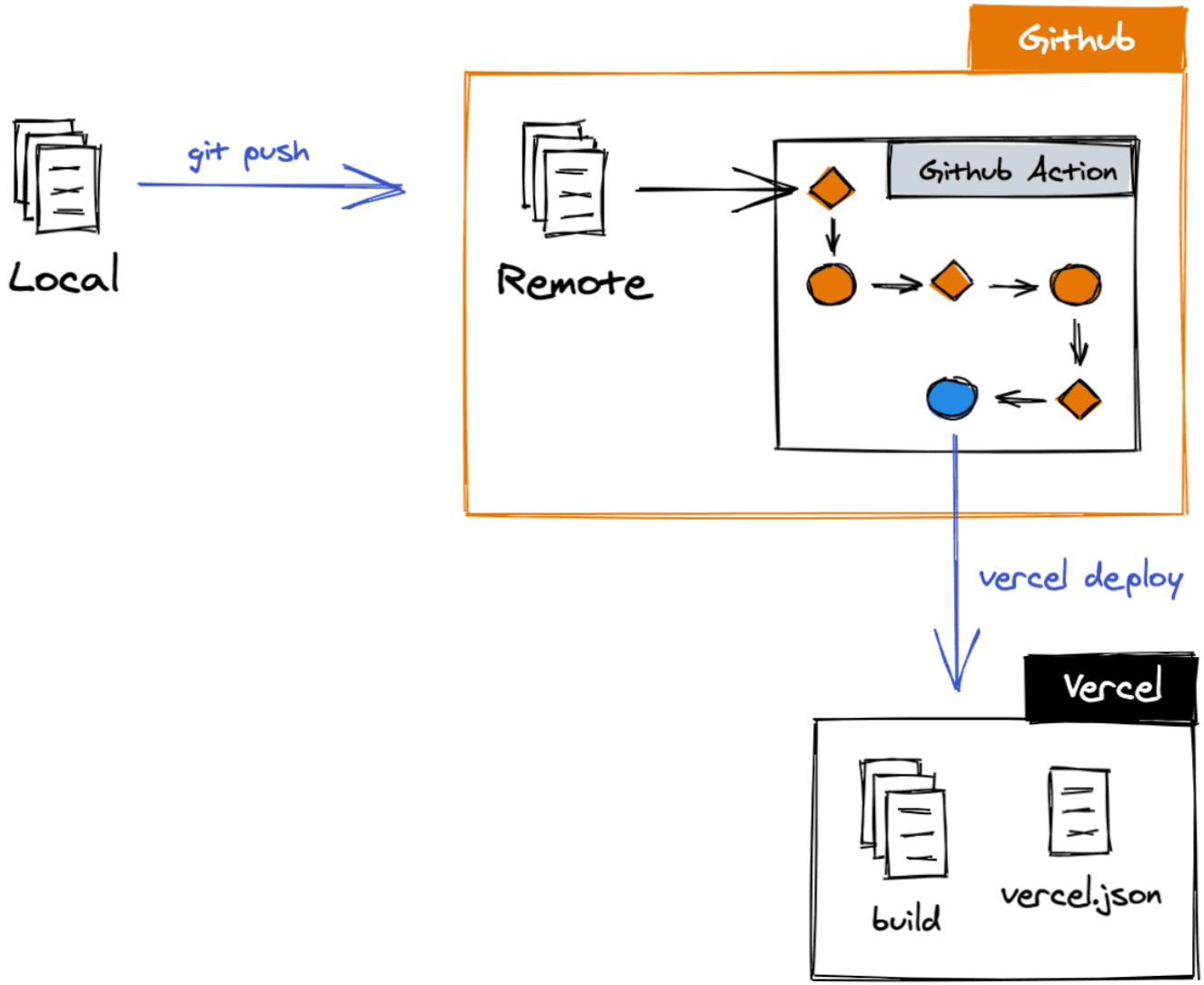
Integración Continua (CI)

Git, GitHub y GitFlow



Despliegue Continuo (CD)

GitHub Actions, Vercel y Flujo DPS

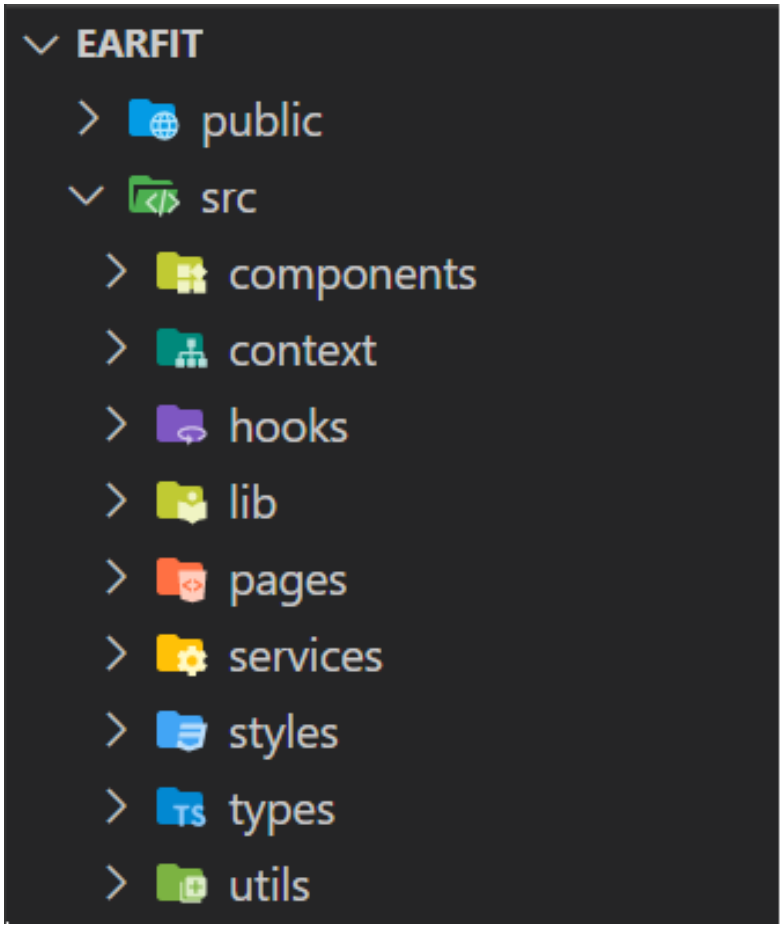


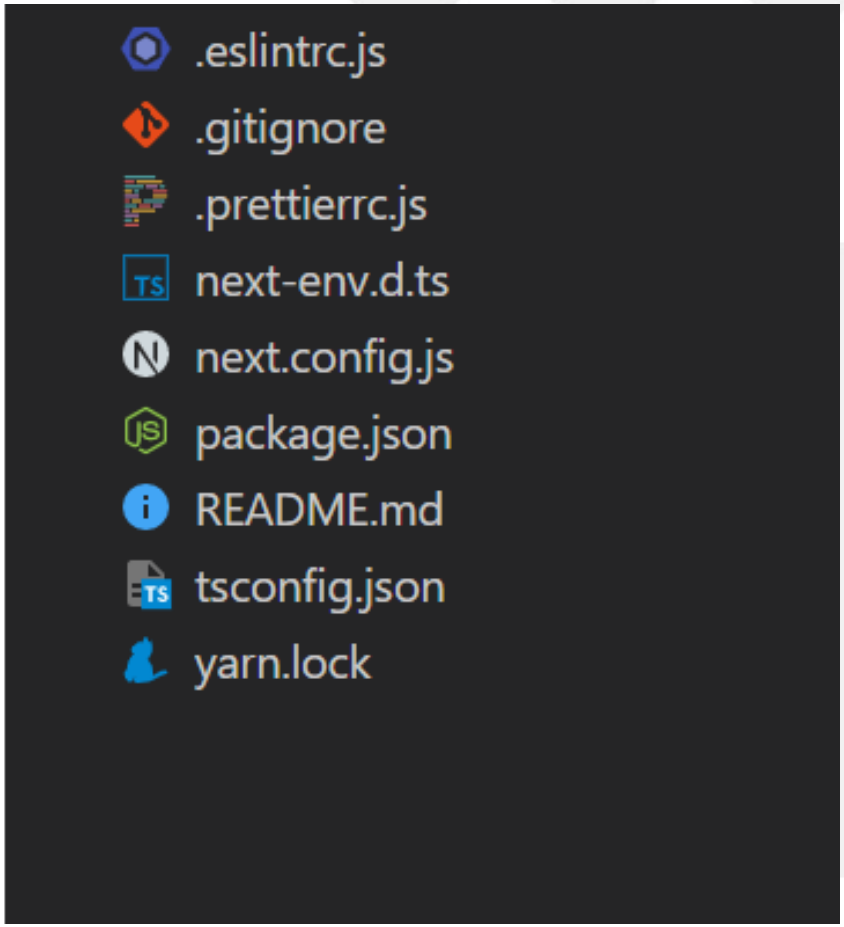
Stack Tecnológico

- **Next.js**: Enrutamiento basado en páginas, Prerendering, Code Splitting, Prefetching, Fast Refrest, 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).

Estructura de Archivos

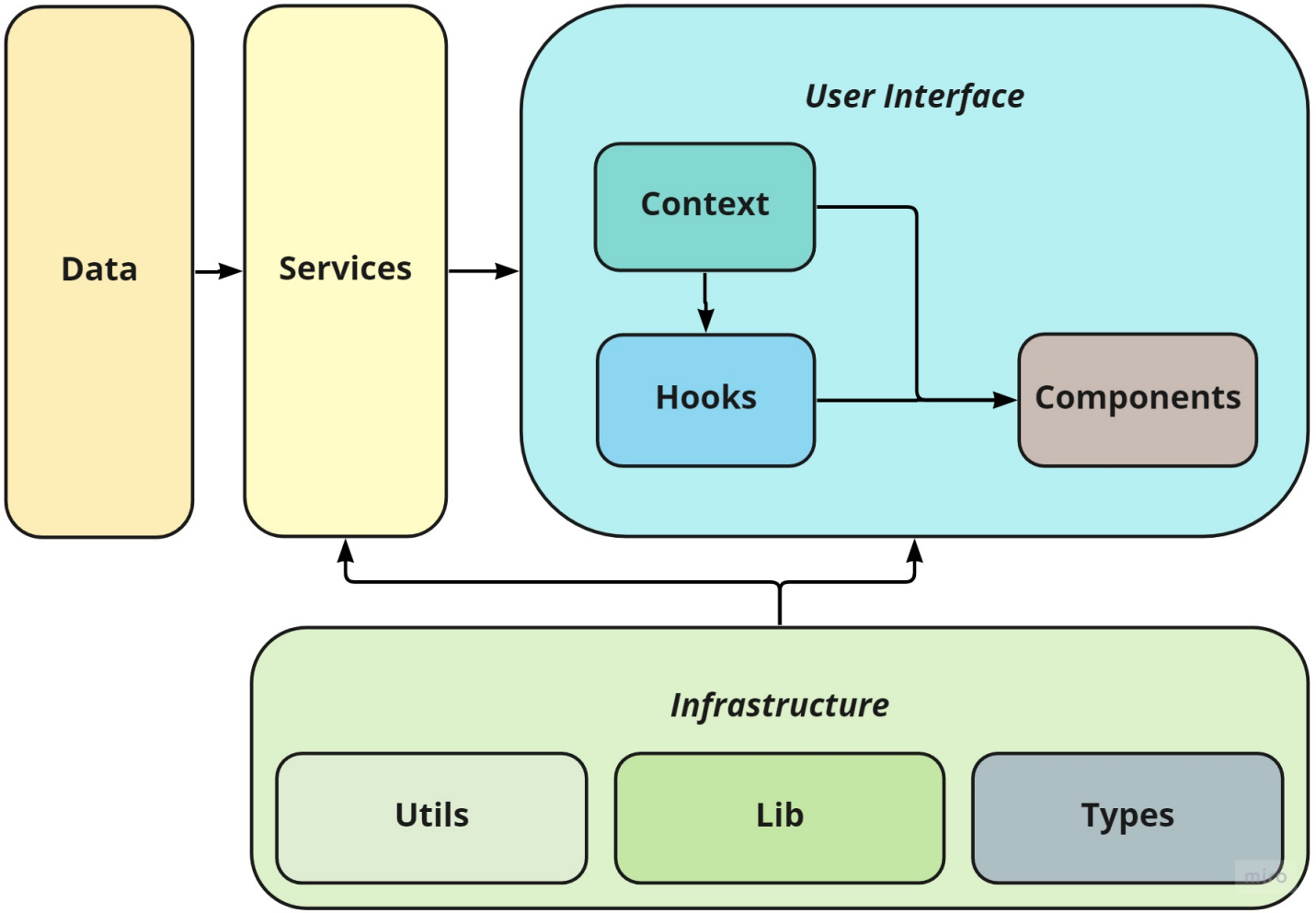
Pages y Public son Directorios Especiales en Next.js





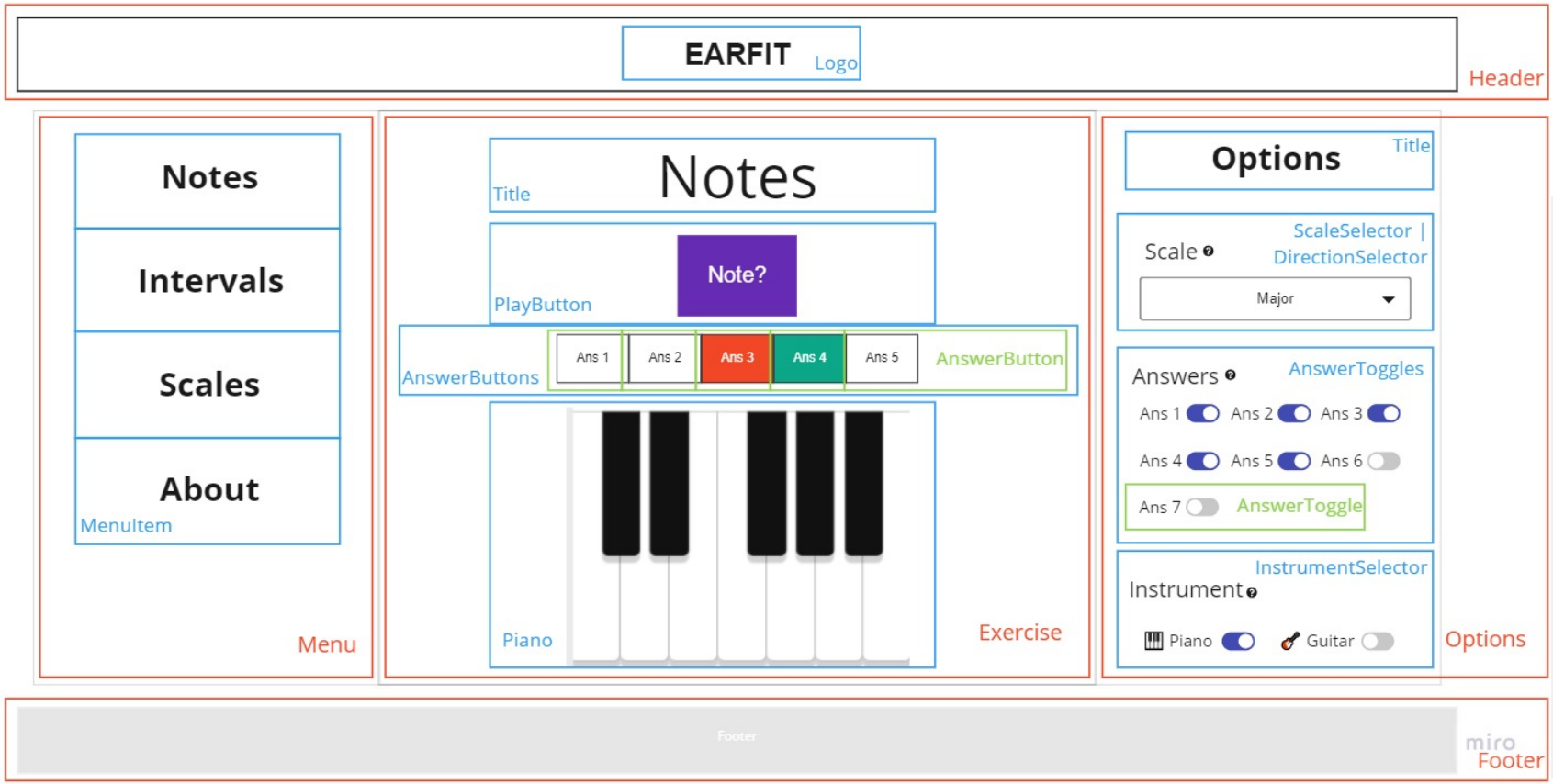
Arquitectura

Estructura, Funcionamiento e Interacción



Jerarquía de Componentes

Los Componentes Son Reutilizados entre Páginas



Hooks

Los Custom Hooks Encapsulan la Lógica de Estado

- **useExercise.tsx**: Establecer las respuestas para el ejercicio.
- **useAnswerToggles.tsx**: Añadir y quitar respuestas a la pregunta.
- **useAnswerButtons.tsx**: La lógica de los botones de respuesta.
- **useAnswer.tsx**: Calcular la respuesta a preguntar.
- **usePlayButton.tsx**: Reproducir la respuesta.

- **EarfitContext.tsx**: Establecer y seleccionar los instrumentos. Se usa con el Hook useInstrumentContext().

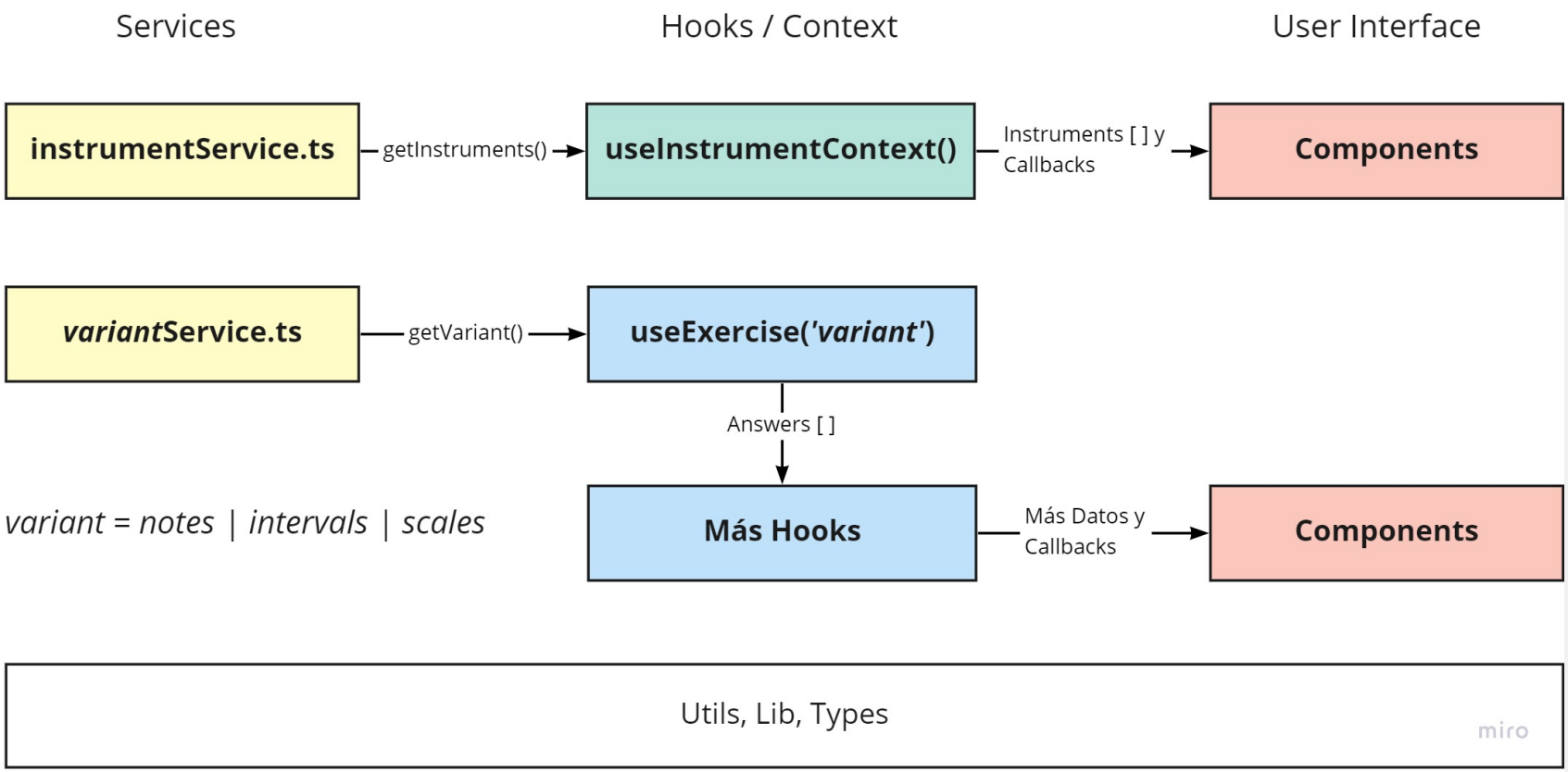
Servicios

Proveen los Datos a los Hooks y al Context

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

Comportamiento de una Página

Interacción entre Componentes, Hooks y Servicios



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.

Tipos de TypeScript

Los Instrumentos y las Respuestas

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

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

Ejemplo (1/3)

Llamada a los Hooks Necesarios

```
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
);
```

Ejemplo (2/3)

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>
  );
```

Ejemplo (3/3)

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

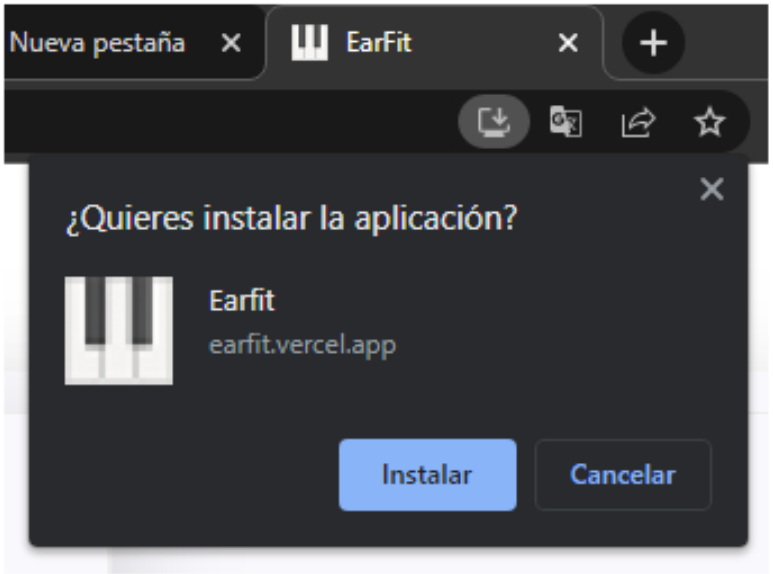
Para que una aplicación sea PWA debe tener:

- 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)

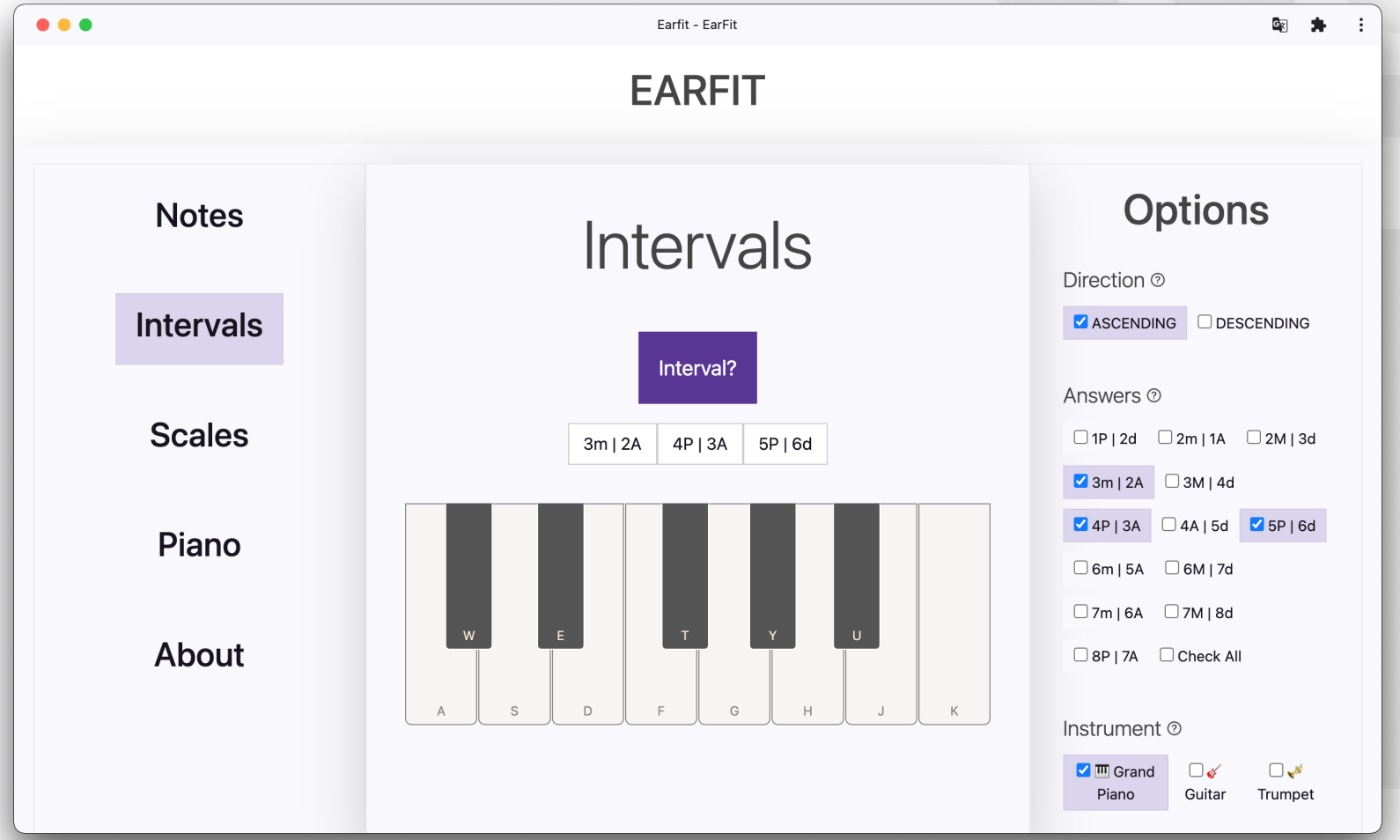
¿Cómo se Instala?

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



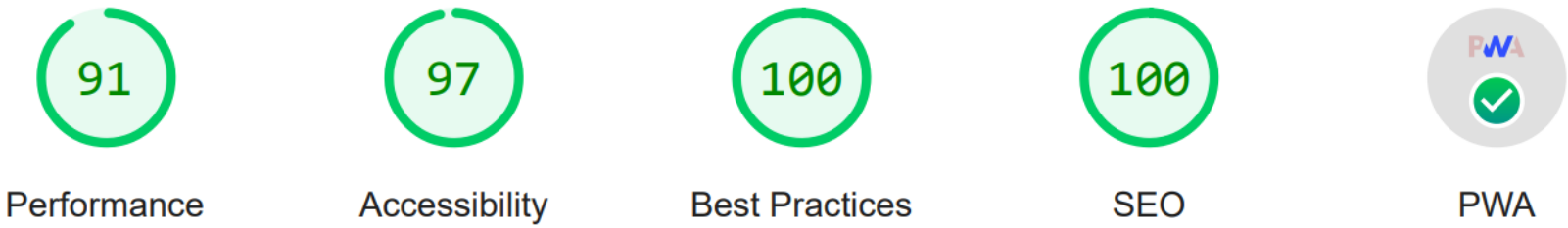
Progressive Web App (3/3)

Earfit como PWA en MacOs



Google Lighthouse

Auditoría de Calidad de la Página Web

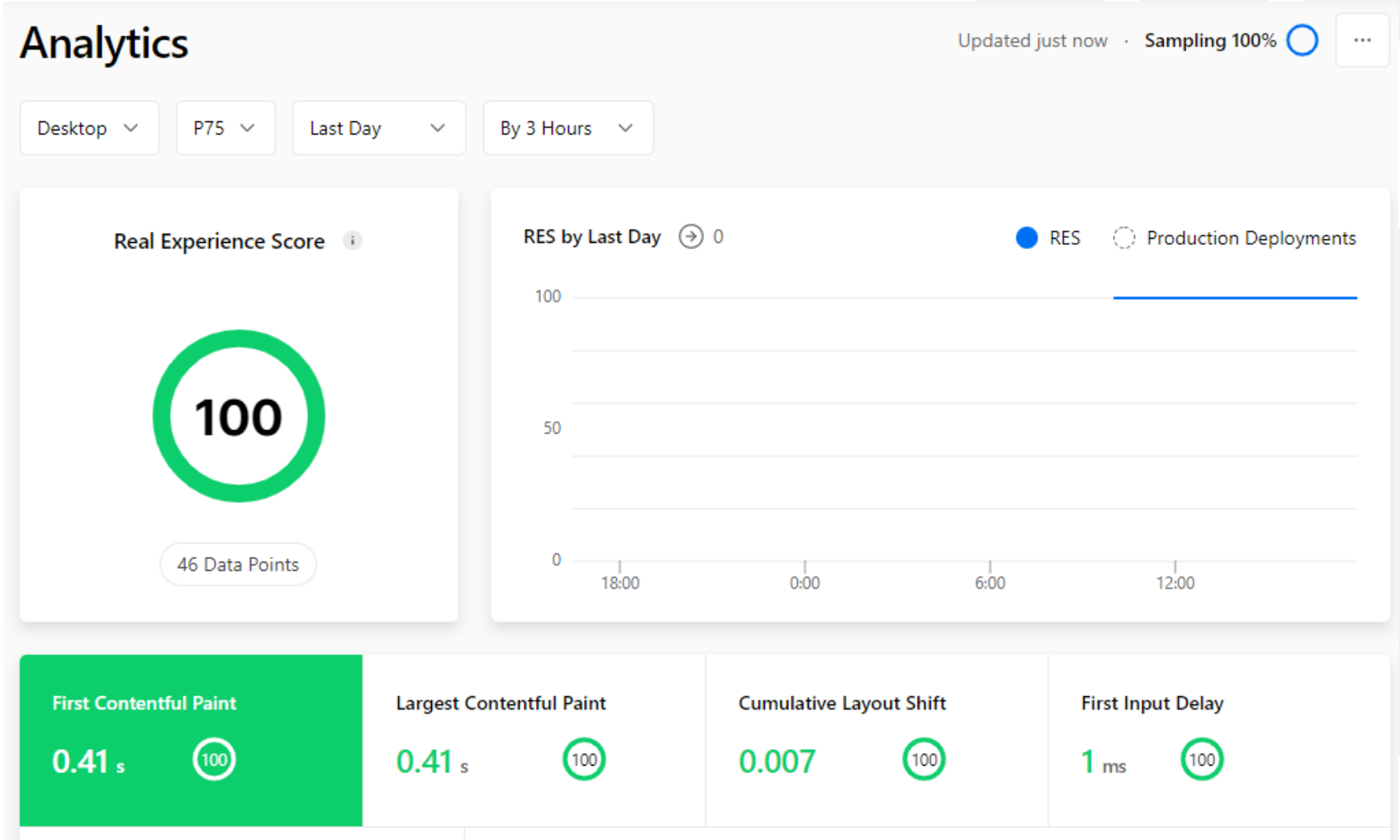


En el apartado Performance:

- 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 (Web Vitals)



Conclusiones

Objetivos Alcanzados:

- PWA que permite desarrollar el oído musical mediante entrenamiento auditivo.
- Diferentes tipos de ejercicios personalizables.
- Varios instrumentos para practicar con sus sonidos.
- Buenas prácticas, gestión ágil y DevOps.

Trabajos Futuros:

- Ejercicios de acordes, ritmos, progresiones, modo nocturno y varios idiomas.

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