



VOICEDEX

POKÉDEX CON VOZ



ALEJANDRO FLORES GONZÁLEZ

ÍNDICE

INTRODUCCIÓN Y JUSTIFICACIÓN DEL PROYECTO	3
Tecnologías usadas	4
Librerías usadas.....	4
Hardware empleado	4
Esquema de casos de uso	5
Diagrama de clases	6
Mapa de navegación.....	7
Descripción de ficheros	7
GraphQL: el back-end.....	8
Próximas actualizaciones.....	9
Subida a Play Store	9
Imágenes de presentación	11
Logo.....	12
Estructura del proyecto.....	13
Estilos	27
Futura implementación de anuncios.....	28
Bibliografía	28
Agradecimientos	29

INTRODUCCIÓN Y JUSTIFICACIÓN DEL PROYECTO

Este proyecto trata de una aplicación móvil para los aficionados de Pokémon.

Basada en la Pokédex, es una aplicación que muestra los datos de los Pokémon, enfocándome en el diseño de la aplicación y añadiendo un elemento que no he visto en otra aplicación de este tipo, descripción por voz mediante TTS (Text to Speech).

Consiste en una vista en la que se ve una lista de Pokémon. Al pulsar un componente navega a la vista de los detalles del Pokémon seleccionado.

Desarrollé esta aplicación móvil para mejorar mi aprendizaje en nuevas tecnologías, y a la vez publicarlo en Play Store y en mi lista de proyectos a mostrar para mi portafolio.

El proyecto tiene oportunidades de tener éxito a través de Play Store, ya que he realizado una investigación de este tipo de aplicaciones, y hay muy pocas realmente logradas. A parte, no hay ninguna que use una voz para describir a los Pokémon.

Pokémon es una compañía muy popular y los próximos lanzamientos podrían dar un empujón a la aplicación.

Es seguro que habrá mayor afluencia de usuarios buscando aplicaciones de este tipo en la Play Store.

Aunque la aplicación no tuviera demasiadas descargas en Play Store, sería un activo valioso en mi portafolio.

Tecnologías usadas

- **React Native**
- **Expo**
- **GraphQL**

React Native es un framework JavaScript para crear aplicaciones reales nativas para iOS y Android, basado en la librería de JavaScript React.

Expo es un conjunto de herramientas y servicios creados en torno a React Native y plataformas nativas que lo ayudan a desarrollar rápidamente en iOS, Android.

GraphQL es un lenguaje de consulta y manipulación de datos para APIs, y un entorno de ejecución para realizar consultas con datos existentes.

Todo el proyecto se ha creado en el editor de código Visual Studio Code.

Librerías usadas

- Apollo Client: para manejar GraphQL
- Vector Icons: Iconos gratuitos
- React Navigation: Navegación
- Expo Speech: TTS
- React Native Gesture Handler: Para las animaciones de la navegación

Hardware empleado

La aplicación se ha desarrollado en mi equipo de escritorio.

Las pruebas se han realizado en mi dispositivo móvil, un Poco X3 NFC.

Esquema de casos de uso

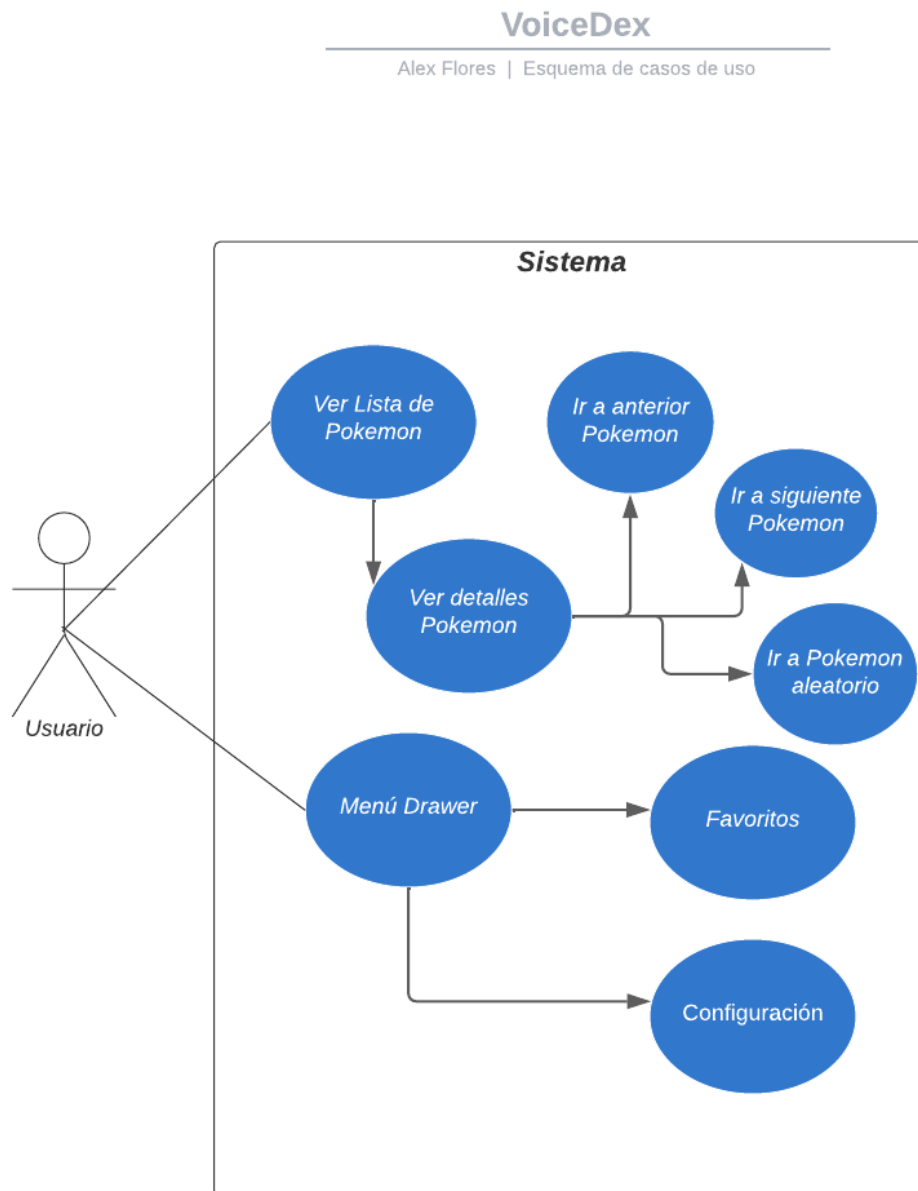
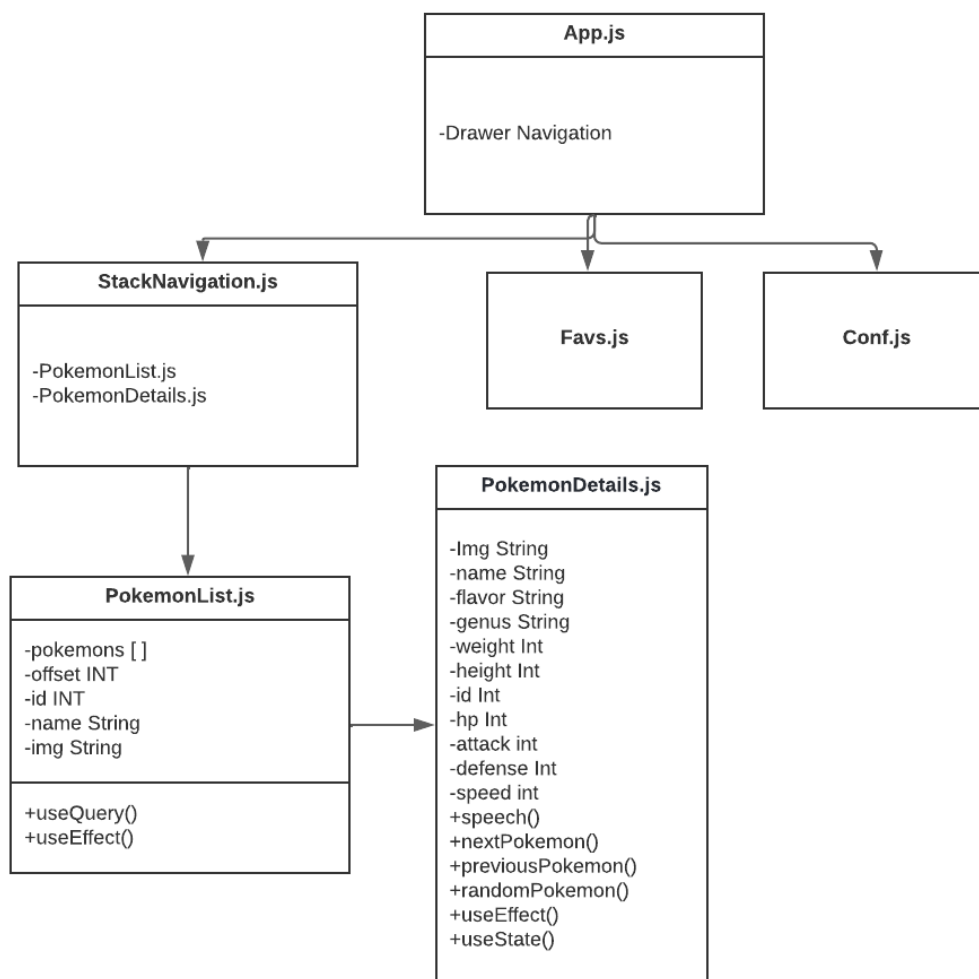


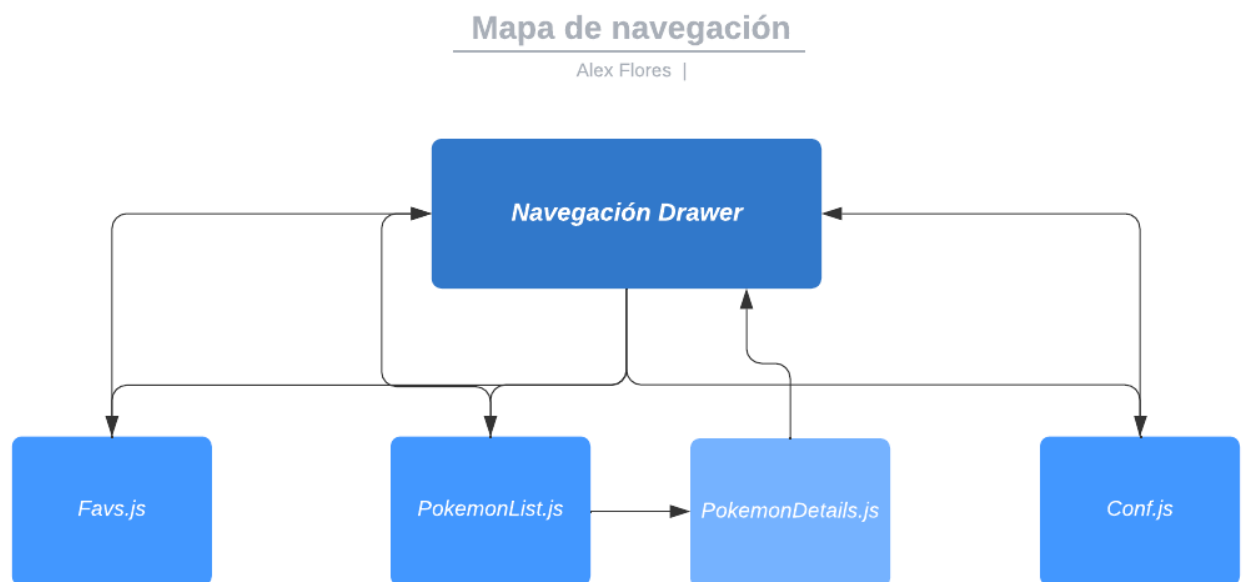
Diagrama de clases

VoiceDex - Pokédex con Voz

Alex Flores | Diagrama de clases



Mapa de navegación



Descripción de ficheros

En el fichero principal, **App.js**, se encuentra la navegación principal, que consiste en un Drawer, un menú de navegación lateral que aparece tras pulsar un botón.

App.js te lleva a **StackNavigation.js**, que es otro menú de navegación diferente, que consiste en apilar las vistas por encima de las anteriores.

En primer lugar, te muestra **PokemonList.js**, que es una lista que recorre los Pokémon, sacando como datos el id, el nombre, el tipo y la imagen. Se carga un límite de 15 Pokémon, para esto he creado el componente **PokemonCard.js**, para separar el código de la lista a la de cada componente de la lista, haciendo el código más fácil de leer y desarrollar.

Los Pokémon se van actualizando conforme al usuario llegué al final de la lista, gracias a la propiedad **onReachEnd** del elemento **Flatlist**. Esto nos permite rapidez a la hora de traer los datos a la vista y también fluidez para cargar los siguientes.

Cuando pulsas en un componente de la lista (un Pokémon) te lleva a la sección de detalles, **PokemonDetails.js** que contiene todos los datos de detalles de estos.

Tanto **Favs.js** como **Conf.js** son ficheros sin propiedades por ahora, se desarrollarán en un futuro si la aplicación tiene buen recibimiento.

GraphQL: el back-end

Los datos de los Pokémon son traídos mediante un servidor alojado en [‘https://beta.pokeapi.co/graphql/v1beta/’](https://beta.pokeapi.co/graphql/v1beta/)

A parte de esto, ciertas imágenes se sacan a través de un repositorio público de GitHub.

GraphQL es un lenguaje de consulta y un tiempo de ejecución del servidor para las interfaces de programación de aplicaciones (API); su función es brindar a los clientes exactamente los datos que solicitan y nada más.

Gracias a GraphQL, las API son rápidas, flexibles y sencillas para los desarrolladores. Como alternativa a REST, GraphQL permite que los desarrolladores creen consultas para extraer datos de varias fuentes en una sola llamada a la API.

Los desarrolladores de API utilizan esta herramienta para crear un esquema que describa todos los datos posibles que los clientes pueden consultar a través del servicio.

Los esquemas de GraphQL están compuestos por tipos de objetos, que definen los que puede solicitar y sus campos.

A medida que ingresan las consultas, GraphQL las aprueba o rechaza en función del esquema, y luego ejecuta las validadas.

El desarrollador de API adjunta cada campo de un esquema a una función llamada resolver. Durante la ejecución, se llama al resolver para que genere el valor.

Próximamente actualizaciones

En las próximas actualizaciones serán desarrollados:

Un sistema de favoritos, mediante el manejo de un estado global que guarde los id de los pokemon en un array.

La vista de configuración, donde se podrá deshabilitar la opción de descripción por voz.

Un buscador, muy interesante para facilitar el acceso a un Pokémon si quieres buscarlo.

Estos cambios no son fijos. Será escuchado el feedback que me den otros usuarios por Play Store.

Subida a Play Store (Pruebas)

Tu primera aplicación puedes subirla en 8 pasos:

1. Crea una cuenta de desarrollador.
2. Crear nueva app.
3. Completa la ficha de Play Store.
4. Subir la APK.
5. Clasificación de contenido.
6. Establece precio y distribución.
7. Enviar la app a revisión.
8. Esperar

Para subir aplicaciones a Play Store lo primero que tienes que hacer es crearte una cuenta de desarrollador.

Cuesta 25\$ pero es de pago único.

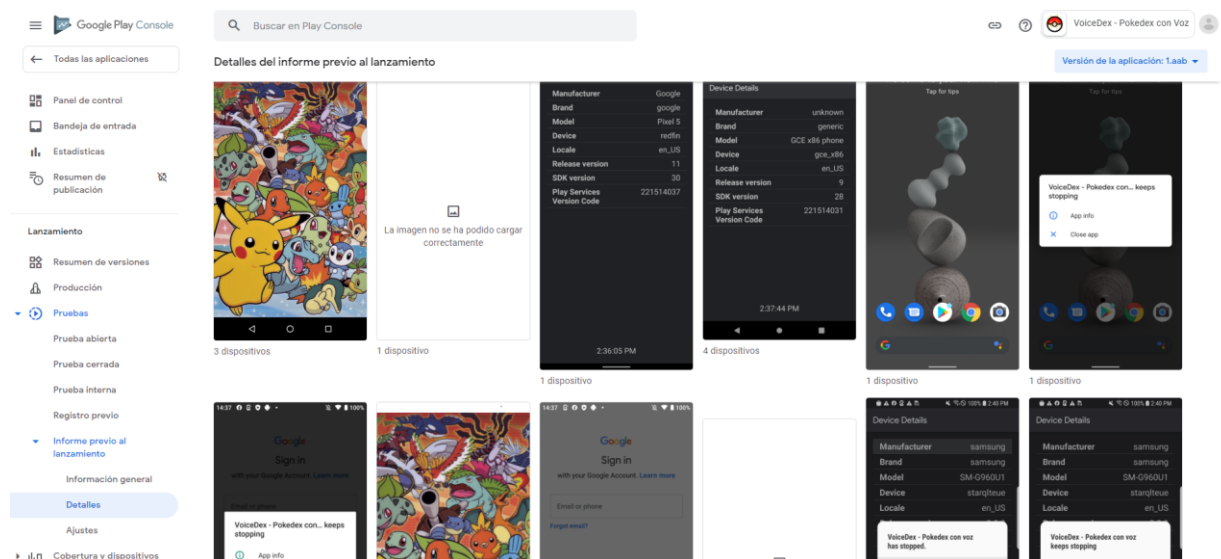
En Apple(ios) la licencia cuesta 99\$ al año.

Cuando tienes la licencia tienes acceso a Play Console, donde puedes editar toda la información respecto a tus aplicaciones.

Hay que crear una aplicación y subirla cuando tengas una build hecha.

Después completar una ficha en la que tienes que incluir descripción, icono, capturas de pantalla e información relacionada.

Clasificar tu contenido, establecer precio y enviar la aplicación dónde te la revisan con varios dispositivos.



La aplicación fue enviada a revisar y encontraron un error, fue solventado y vuelto a enviar. Suelen tardar 1-3 días.

Hay otros detalles a tener en cuenta en una aplicación:

- Splash – Imagen que se muestra mientras carga la aplicación.
- Las vistas en todos los dispositivos móviles, en la revisión me mandaron capturas de un Nokia.
- La navegación y el tiempo de carga son otros factores importantes.

Imágenes de presentación

Estas son las imágenes que serán visibles a la hora de ver la aplicación en Play Store.

Me fijé en que el resto de aplicaciones de este tipo no mostraban las imágenes de los Pokemon limpiamente por tema de derechos de autor y edité las imágenes tanto en Photoshop web como en otros editores web de imágenes.



A parte de las capturas también es requerido un gráfico de funcionalidades:



Diseñado en Paint3D

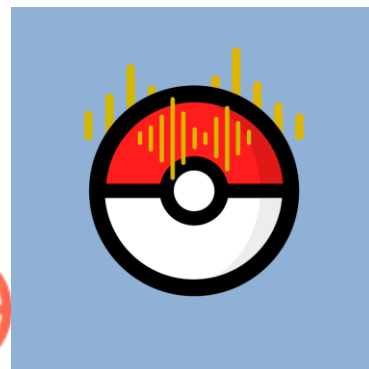
IMÁGENES DE PRESENTACIÓN NO DISPONIBLES EN PDF

Logo

Diseño del logo final:

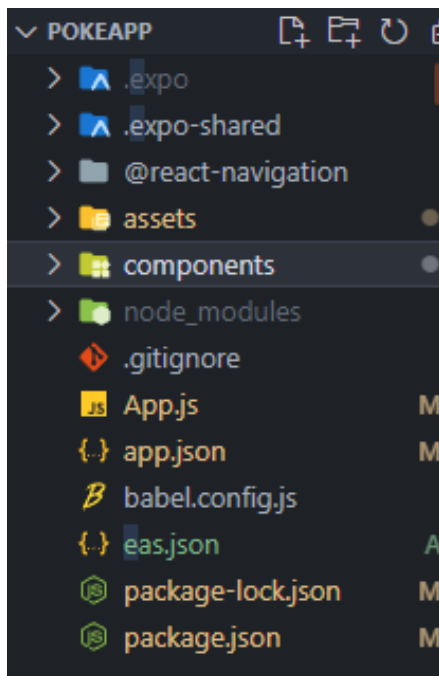


Diseño de anteriores íconos:



Estructura del proyecto

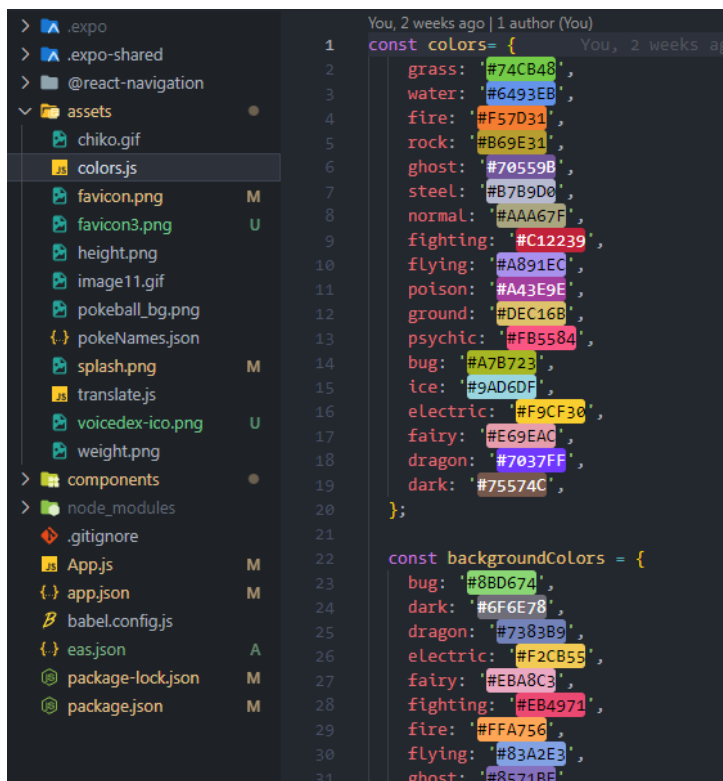
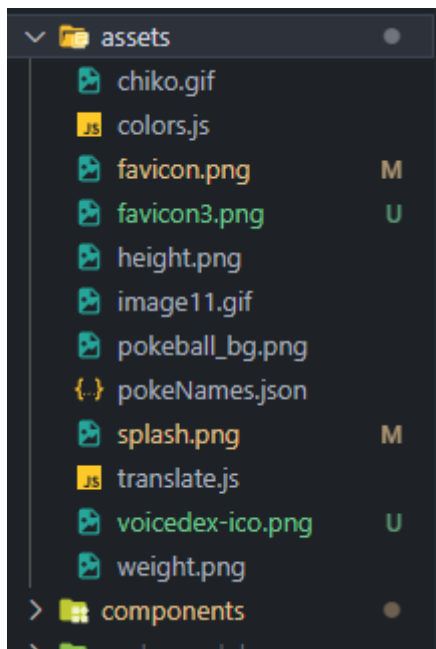
El proyecto está basado en Javascript, React Native utiliza React, cualquier duda respecto a cómo funcionan los componentes pueden consultarla conmigo o en la documentación oficial.



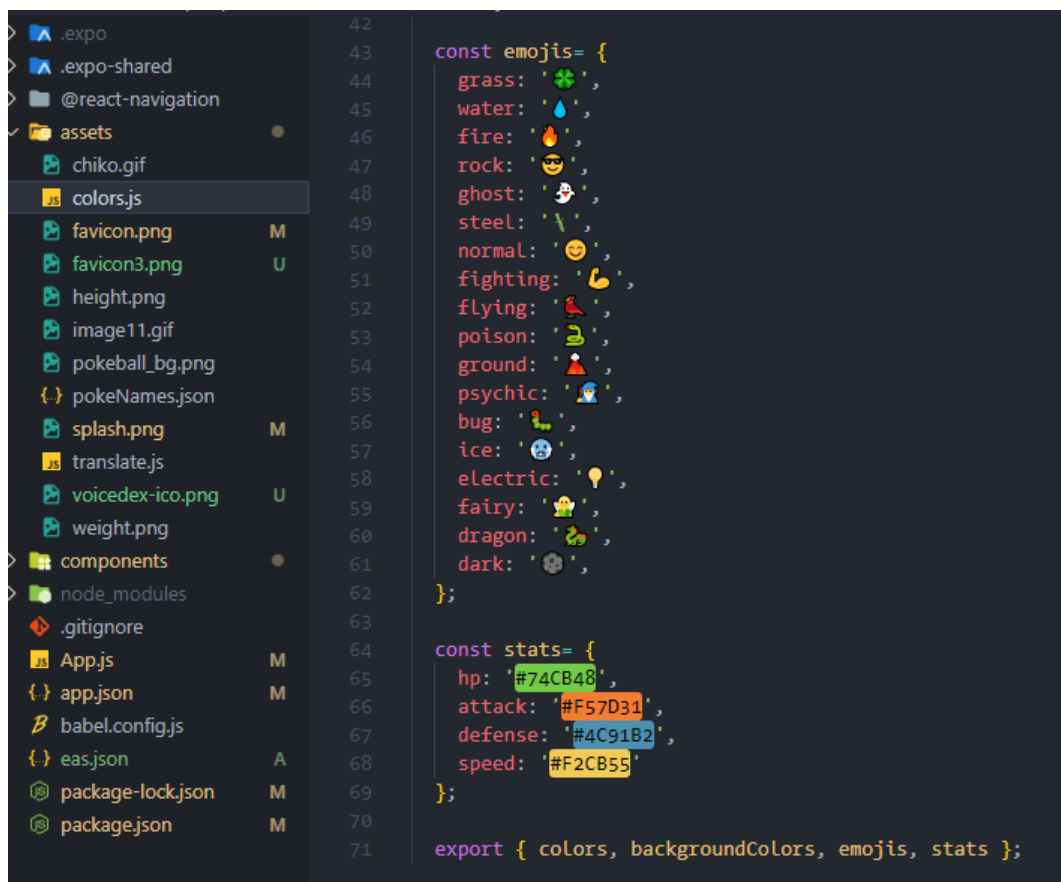
En la carpeta pokeapp tenemos las primeras tres carpetas de configuración y también .gitignore, que ignora carpetas o archivos grandes que no deben subirse a los repositorios, como node_modules.

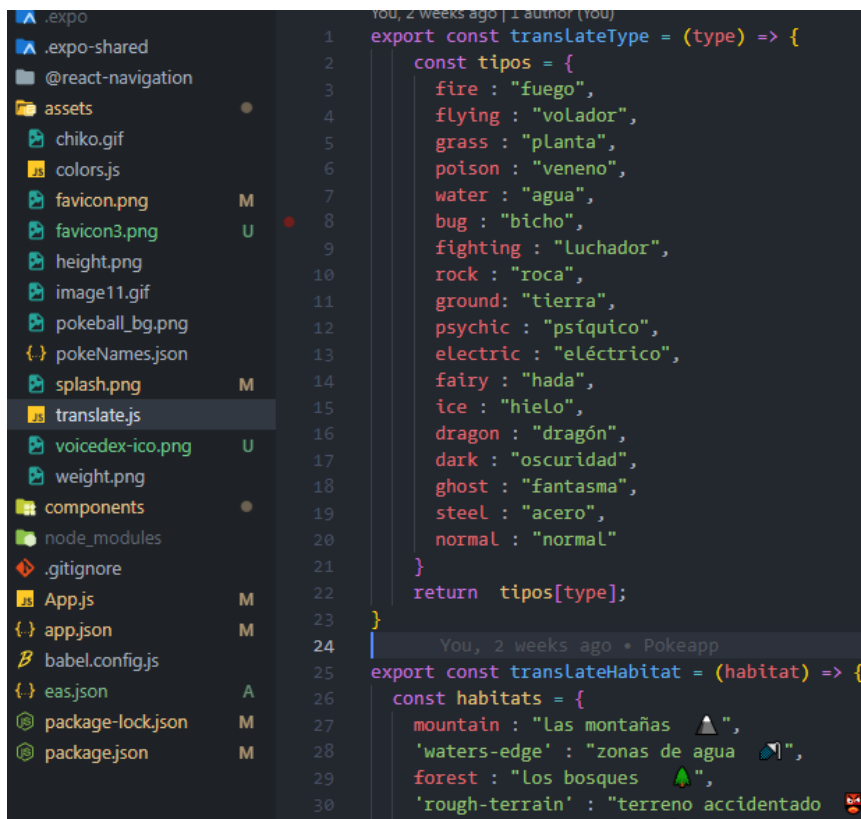
Node_modules es una carpeta que contiene todas las dependencias y librerías.

Assets es una carpeta dónde tengo las imágenes y componentes externos.



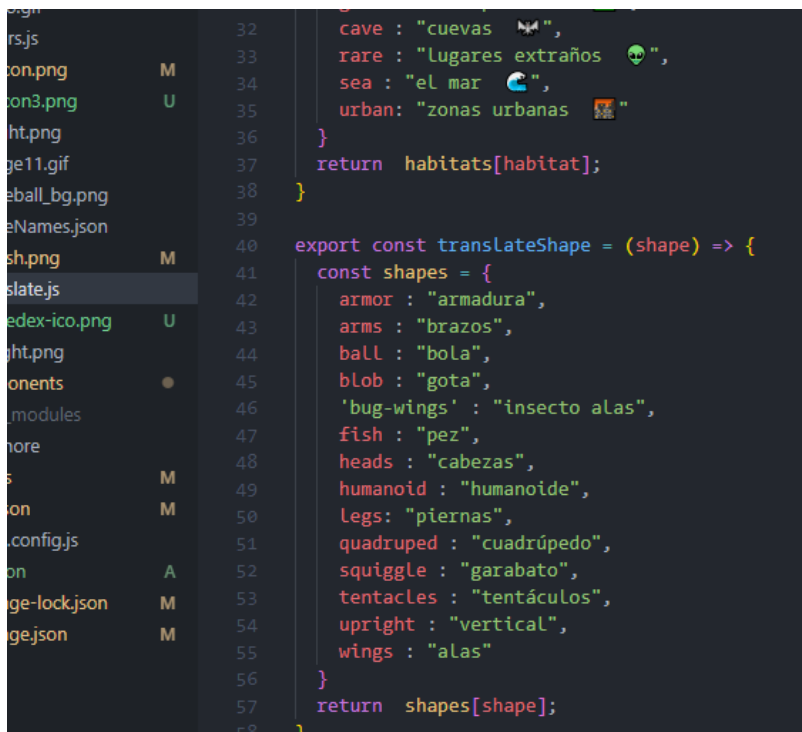
Colors.js es un archivo en el que asigné a cada tipo un color primario y otro de fondo. También emojis por tipo y colores por stats:





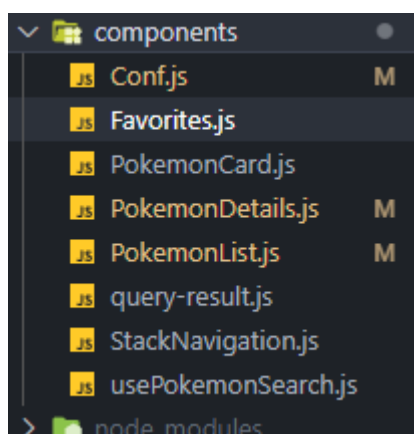
```
1 export const translateType = (type) => {
2   const tipos = {
3     fire : "fuego",
4     flying : "volador",
5     grass : "planta",
6     poison : "veneno",
7     water : "agua",
8     bug : "bicho",
9     fighting : "luchador",
10    rock : "roca",
11    ground: "tierra",
12    psychic : "psíquico",
13    electric : "eléctrico",
14    fairy : "hada",
15    ice : "hielo",
16    dragon : "dragón",
17    dark : "oscuridad",
18    ghost : "fantasma",
19    steel : "acero",
20    normal : "normal"
21  }
22  return tipos[type];
23 }
24
25 export const translateHabitat = (habitat) => {
26   const habitats = {
27     mountain : "Las montañas 🏔️",
28     'waters-edge' : "zonas de agua 🌊",
29     forest : "Los bosques 🌲",
30     'rough-terrain' : "terreno accidentado 🏞️"
```

Algunos datos traídos del servidor venían en inglés. Implementé traducciones y las metí en el componente translate.js



```
32   cave : "cuevas 🕒",
33   rare : "Lugares extraños 🧟",
34   sea : "el mar 🌊",
35   urban : "zonas urbanas 🏙️"
36 }
37 return habitats[habitat];
38 }
39
40 export const translateShape = (shape) => {
41   const shapes = {
42     armor : "armadura",
43     arms : "brazos",
44     ball : "bola",
45     blob : "gota",
46     'bug-wings' : "insecto alas",
47     fish : "pez",
48     heads : "cabezas",
49     humanoid : "humanoide",
50     legs : "piernas",
51     quadruped : "cuadrúpedo",
52     squiggle : "garabato",
53     tentacles : "tentáculos",
54     upright : "vertical",
55     wings : "alas"
56   }
57   return shapes[shape];
58 }
```


La carpeta que más tiempo lleva para completar es components. Estos son los archivos que contiene:



Conf.js y Favs.js son muy similares, solo se devuelve texto.

```
components > JS Conf.js > ...
You, last week | 1 author (You)
1 import React, { useState, useEffect } from 'react'; 7.4k (gzipped: 3k)
2 import { StyleSheet, View, Text, Image, Dimensions, TouchableOpacity } from 'react-native';
3
4 import pokeico from '../assets/favicon.png'
5
6
7 export default function Conf({navigation}) {
8   return (
9     <View style={styles.container}>
10
11       <Text style={{fontWeight:'bold', fontSize:22, paddingBottom:50}}>¡Configuración estará c
12       <Text style={{fontSize:50}}>@</Text>
13       <Text style={{fontSize:16, marginTop:50}}>Ajustes relacionados a la voz</Text>
14       <Text style={{fontSize:20}}>@</Text>
15       <View style={styles.openDrawer}>
16         <TouchableOpacity onPress={() => navigation.openDrawer()}>
17           <View style={{paddingLeft:3,paddingBottom:1,paddingTop:1}}>
18             <Image source={pokeico}/>
19           </View>
20         </TouchableOpacity>
21       </View>
22     </View>
23   );
24 }
```

PokemonCard.js es el componente de un Pokemon en la lista de Pokemon, que trae una lista de este componente, pasandole props id progresivo.

Como podemos ver PokemonCard devuelve un elemento TouchableOpacity, que es pulsable.

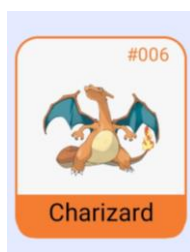
Al pulsarse llama a una función que navega a la vista de detalles, pasandole el id para reconocer el pokemon.

```
return (
  <TouchableOpacity
    activeOpacity={0.7}
    onPress={() => navigation.navigate("Pokemon Details", {
      screen: 'Pokemon Details',
      type:type,
      id: id
    })}
  >

    <View style={[styles.card, { borderColor: colors[type] }]}>
      <Text style={[styles.digits, { color: colors[type] }]}>
        #
        {id.toString().length == 1 ? '00' : ''}
        {id.toString().length == 2 ? '0' : ''}
        {id}
      </Text>
      <Image source={{ uri: imageUrl }} style={styles.image} resizeMode='contain' />
      <View style={[styles.nameContainer, { backgroundColor: colors[type] }]}>
        <Text style={styles.name}>
          {name.length > 11 ? name.substring(0,11) :
            name.charAt(0).toUpperCase() + name.slice(1)}
        </Text>
      </View>
    </View>
  </TouchableOpacity>
)
```

La vista de PokemonCard.js usa datos traídos: nombre, tipo, id e imagen.

Aquí un ejemplo de PokemonCard:



PokemonDetails.js es el componente más largo, ya que tiene mayor cantidad de detalles y también mayor cantidad de funcionalidades:

```
const PokemonDetails = ({ route, navigation }) => {

  const { id } = route.params
  const [types, settypes] = useState([])
  const [name, setname] = useState()
  const [genus, setgenus] = useState()
  const [flavor, setflavor] = useState()
  const [habitat, sethabitat] = useState()
  const [hp, sethp] = useState()
  const [attack, setattack] = useState()
  const [defense, setdefense] = useState()
  const [speed, setspeed] = useState()
  const [islegendary, setislegendary] = useState()
  const [tinyGifUri, settinyGifUri] = useState(null)
  const [tinyBackGifUri, settinyBackGifUri] = useState(null)
  const [tinyImgUri, settinyImgUri] = useState(null)
  const [tinyBackImgUri, settinyBackImgUri] = useState(null)
  const [weight, setweight] = useState()
  const [height, setheight] = useState()
  const [fav, setFav] = useState(false)

  const { loading, error, data } = useQuery(GET_DETALLES, {
    variables: { "_eq": id },
  });
```

Declaración de variables con useState, un hook de React. El primer valor del array es la variable y el segundo el método para setearlo. Entre parentésis el valor por defecto, sino hay nada es undefined.

También llamada a useQuery, dónde llama al servidor mediante una consulta:

Aquí parte de la consulta:

```

export const GET_DETALLES = gql`
query getDetalles($_eq: Int) {
  pokemon_v2_pokemonspecies(where: {id: {_eq: $_eq}}) {
    pokemon_v2_pokemonspeciesflavortexts(where: {language_id: {_eq: 7}}, limit: 1) {
      flavor_text
    }
  }
  pokemon_v2_pokemonshape {
    name
  }
  pokemon_v2_pokemonspeciesnames(where: {language_id: {_eq: 7}}) {
    genus
  }
  name
  pokemon_v2_pokemons_aggregate {
    nodes {
      pokemon_v2_pokemonstats {
        base_stat
        pokemon_v2_stat {
          name
        }
      }
      height
      weight
    }
  }
}
`

```

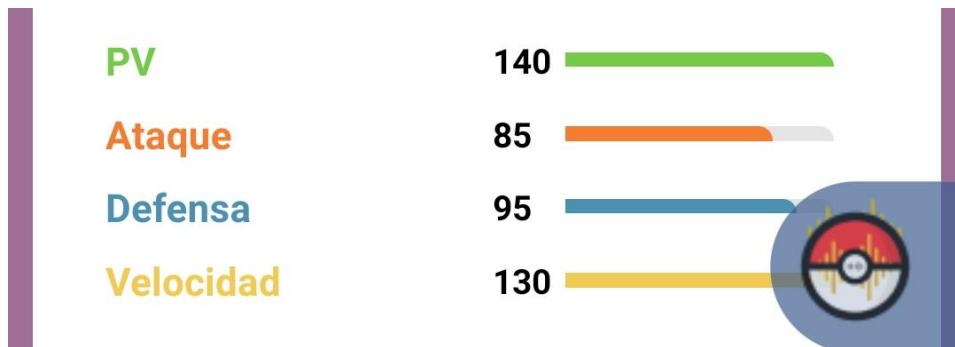
```

const Progress = ({ step, stat, height }) => {
  return (
    <>
    <Text style={{ fontSize: 14, fontWeight: 'bold', }}>
    <View style={{
      height,
      backgroundColor: 'rgba(0,0,0,0.1)',
      borderTopEndRadiusRadius: height,
      borderTopRightRadius: height,
      overflow: 'hidden',
      width: 110,
      alignSelf: 'center',
    }}>
    <View style={{
      height,
      width: step,
      borderTopEndRadiusRadius: height,
      borderTopRightRadius: height,
      backgroundColor: stats[stat],
      overflow: 'hidden',
      position: 'absolute',
      left: 0,
      top: 0,
    }}></View>
    </View>
  )
}

```

Los distintos stats de detalles se muestran mediante la función Progress, se podría llevar a un componente externo para seccionar código.

Aquí se muestra el componente y a la derecha el botón de navegación:



ImageUri es un String dinámico que cambia mediante según el id

```
work/${id}.png`
```

Esto es template string, una característica lanzada en ES6 de JS

```
const imageUri = `https://raw.githubusercontent.com/PokeAPI/sprites/master/pokemon/${id}.png`

const speak = () => {
  Speech.speak(name + "." +
    data.pokemon_v2_pokemonspecies[0].pokemon_v2_pokemonspecies
    rate: 1.1,
    language: 'es-ES',
    pitch: 1,
    voice: "es-es-x-eeed-network"
  );
};

if (!loading) {
  if (data && Object.keys(data)?.length > 0 && name !== undefined) {
    console.log("EL loading mas data >")
    speak();
  }
}
```

Speak() es una función que implementa la librería de expo-speech para reproducir la voz. Se le indica el String a reproducir.

Se ve también un condicional para controlar si los datos todavía no han llegado. Ya que si no hay datos y una función los usa la aplicación no funciona como se espera o suele haber errores.

```
useEffect(() => {  
  if (route.params?.type) {  
    settypes([route.params.type])  
  }  
  if (data && Object.keys(data)?.length > 0)  
    setname(data.pokemon_v2_pokemonspecies[0].name)  
    settypes([data.pokemon_v2_pokemonspecies[0].types])  
    setflavor(data.pokemon_v2_pokemonspecies[0].flavor_text_entries[0].text)  
    setgenus(data.pokemon_v2_pokemonspecies[0].genus)  
    sethabitat(data.pokemon_v2_pokemonspecies[0].habitat)  
    sethp(data.pokemon_v2_pokemonspecies[0].base_experience)  
    setattack(data.pokemon_v2_pokemonspecies[0].base_experience)  
    setdefense(data.pokemon_v2_pokemonspecies[0].base_experience)  
    setspeed(data.pokemon_v2_pokemonspecies[0].base_experience)  
    setislegendary(data.pokemon_v2_pokemonspecies[0].is_legendary)  
    setweight(data.pokemon_v2_pokemonspecies[0].weight)  
    setheight(data.pokemon_v2_pokemonspecies[0].height)  
    settinyGifUri(`https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream_world/1.png`)  
    settinyBackGifUri(`https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream_world/1.png`)  
    settinyImgUri(`https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream_world/1.png`)  
    settinyBackImgUri(`https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream_world/1.png`)  
    navigation.addListener('beforeRemove',  
      () => {  
        Speech.stop()  
      })  
  })  
})
```

UseEffect es otro hook de React que usa una función cuando hay nuevos props o estados. También se le llama al entrar en el componente por primera vez.

Se controla el que hayan llegado los datos y se setean las variables.

También se controla si salimos de esta vista cortar la descripción por voz.

```

return (
  <View style={ [styles.screen, { backgroundColor: backg
    <View style={ [styles.name]}>
      { /* GOBACK/NAME */ }
      <View style={{ flexDirection: "row" }}>
        <TouchableOpacity
          onPress={goPokemonList}
        >
          <AntDesign name="arrowleft" size={30} color="
        </TouchableOpacity>
        { !loading && name != undefined &&
          <Text style={{ fontWeight: 'bold', fontSize:
            { name.charAt(0).toUpperCase() + name.slice(
          </Text>
        }
      </View>
    { /* ID */ }
    <Text style={{ fontWeight: 'bold', fontSize: 24,
      # {id.toString().length == 1 ? '00' : ''}
      {id.toString().length == 2 ? '0' : ''}
      {id}
    </Text>
  </View>
  { /* FAV and RANDOM */ }
  <View style={ styles.fav }>

```

El return que devuelve los elementos a la vista. Se puede ver el contenedor principal, una flecha que vuelve a la lista, el nombre y el id.

Ejemplo de flecha y nombre:



PokemonList.js es un componente similar al de detalles pero con muchos menos detalles. Trae los datos nombre, id, tipo e imagen y los pasa a pokemonCard para que los pinte.

```

<FlatList
  data={pokemons}
  keyExtractor={({pokemon}) => pokemon.id}
  ListHeaderComponent={() =>
    <View>
      <Text style={{ fontWeight: 'bold', fontSize: 18 }}>
        <Text style={{ fontWeight: 'bold', fontSize: 18 }}>
      </Text>
    </View>
  )
  ListFooterComponent={() => Loading ? <View>
    <Image
      source={pokegif}
      style={{
        height: 179,
        width: 350,
        marginTop: 10,
        marginBottom: 30
      }}
    />
  </View>
  }
  numColumns={3}
  contentContainerStyle={{ alignItems: 'center' }}
  onEndReachedThreshold={0.2}
  onEndReached={() => {
    LoadMore()
  }}
  renderItem={renderData}
/>

```

Lo más interesante es la FlatList, un elemento móvil ideal para listas largas, ya que tiene propiedades como onEndReached que permite cargar datos cuando el usuario se acerque al final de la lista mostrada. renderItem es la función que renderiza, en este caso le paso PokemonCard, en la función renderData.

Data es el objeto del que se sacan los datos, y es requerido diferenciarlos mediante keyExtractor.


```

hents > StackNavigation.js > ...
You, 2 weeks ago | 1 author (You)
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import PokemonList from './PokemonList'
import PokemonDetails from './PokemonDetails'

const Stack = createNativeStackNavigator();

export default function StackNavigation() {

  return (
    <Stack.Navigator screenOptions={{headerShown:false}}>
      <Stack.Screen component={PokemonList} name='Pokemon List' />
      <Stack.Screen component={PokemonDetails} name='Pokemon Details' />
    </Stack.Navigator>
  );
}

```

StackNavigation.js es el elemento de navegación secundario. Agrupa PokemonList y PokemonDetails. Usa la librería react-navigation.

```

import 'react-native-gesture-handler'

import StackNavigation from './components/StackNavigation'
import Favorites from './components/Favorites'
import Conf from './components/Conf'

export default function App() {

  const client = new ApolloClient({
    uri: 'https://beta.pokeapi.co/graphql/v1beta/',
    cache: new InMemoryCache(),
  });

  const Drawer = createDrawerNavigator();

  return (
    <ApolloProvider client={client}>
      <NavigationContainer>
        <Drawer.Navigator screenOptions={{
          drawerPosition: 'right',
          drawerType: 'slide',
          headerShown: false,
          drawerIcon: () => (
            <Image source={poke}/>
          )
        }}>
          <Drawer.Screen name="Home" component={StackNavigation} />
          <Drawer.Screen name="Favoritos" component={Favorites} />
          <Drawer.Screen name="Configuracion" component={Conf} />
        </Drawer.Navigator>
      </NavigationContainer>
    </ApolloProvider>
  );
}

```

Por último App.js, el fichero raíz, contiene ApolloProvider, necesario para extraer los datos de las consultas.

Se define en la constante client la dirección del servidor y la cache

También el contenedor de navegación, y el navegador Drawer que es el principal, en el que se encuentra StackNavigation, Favs y Conf.

Estilos

Se usan también estilos en línea, como el siguiente:

```
<View style={{
  height,
  backgroundColor: 'rgba(0,0,0,0.1)',
  borderTopEndRadius: height,
  borderTopRightRadius: height,
  overflow: 'hidden',
  width: 110,
  alignSelf: 'center',
}}>
```

No obstante, es recomendable usar este tipo de estilos para tener un código más legible:

```
const styles = StyleSheet.create({
  container: {
    backgroundColor: '#E4EAFf',
    flex: 1,
    width: '100%',
    alignItems: 'center',
    justifyContent: 'center',
  },
  openDrawer: {
    backgroundColor: '#3e5f8f',
    position: 'absolute',
    right: 0,
    bottom: 0,
    marginBottom: 110,
    width: 80,
    borderColor: '#cdcddc',
    borderTopLeftRadius: 100,
    borderBottomLeftRadius: 100,
    opacity: 0.7
  }
})

<View style={styles.openDrawer}>
  <TouchableOpacity onPress={() => navigation.openDrawer()}>
    <View style={{paddingLeft: 3, paddingBottom: 1, paddingTop: 1}}>
      <Image source={pokeico}/>
    </View>
  </TouchableOpacity>
</View>
```

Futura implementación de anuncios

Tengo pensado poner anuncios en la aplicación si tiene una audiencia de +100 usuarios al mes.

He mirado varias formas como AdMob, Facebook Ads... Finalmente he escogido

Start.io

Los anuncios serían mínimos y considerando siempre entorpecer lo mínimo la experiencia de usuario.

Bibliografía

GraphQL – Lenguaje de consultas

<https://graphql.org/learn/>

React Native

<https://reactnative.dev/docs/getting-started>

Expo

<https://expo.dev/>

Play Store

<https://developer.android.com/distribute/console>

Start.io - Plataforma de anuncios

<https://www.start.io/>

React

<https://es.reactjs.org/>

Agradecimientos

Gracias a Sebastián Flor por ser mi principal guía en el mundo del desarrollo y por enseñarme las tecnologías más innovadoras como React, React Native o GraphQL y por los consejos recibidos.

Gracias a Jorge Galiano por ayudarme a crear el logo y a Esperanza Trapote por valorarlo y darme consejos.

Gracias a mis compañeros de clase de los cuales también he aprendido cosas.

Gracias a los profesores que han dado lo mejor de ellos para enseñarnos lo que saben.

Gracias a mi familia, al resto de amigos y a todos los que me han apoyado y aconsejado en el proyecto.

