



Desarrollo Interpretado



Enfoque de desarrollo de aplicaciones
móviles



Puntos claves

- El código fuente recibe un proceso de “compilación” a una plataforma específica.
- Una parte del código es traducido a código nativo en el momento de la “compilación”.
- La otra parte del código se procesa / interpreta en tiempo de ejecución.

Herramientas



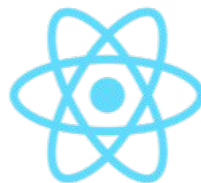
NativeScript



titanium™



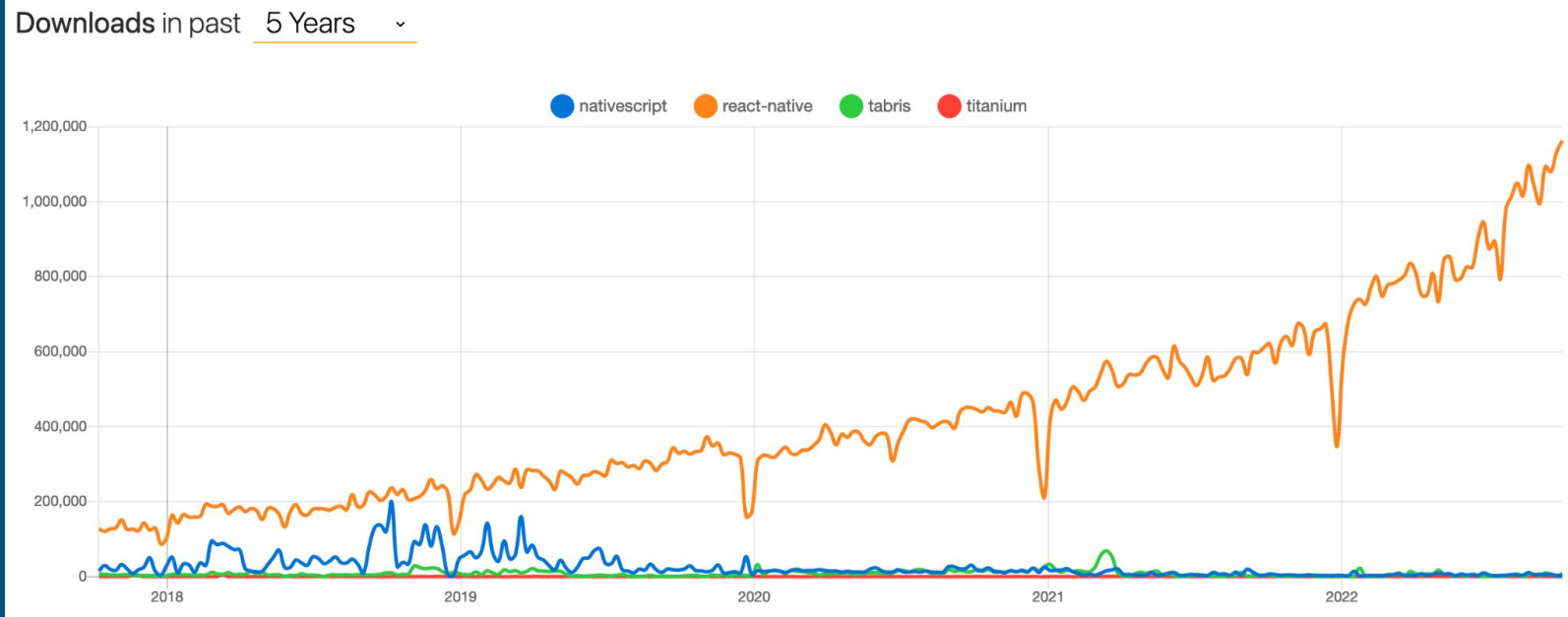
Tabris

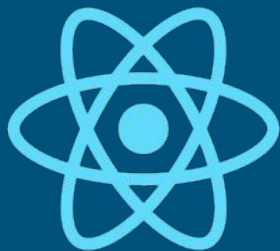


React Native

Trends

<https://www.npmtrends.com/nativescript-vs-react-native-vs-tabris-vs-titanium>





React Native

<https://reactnative.dev/>

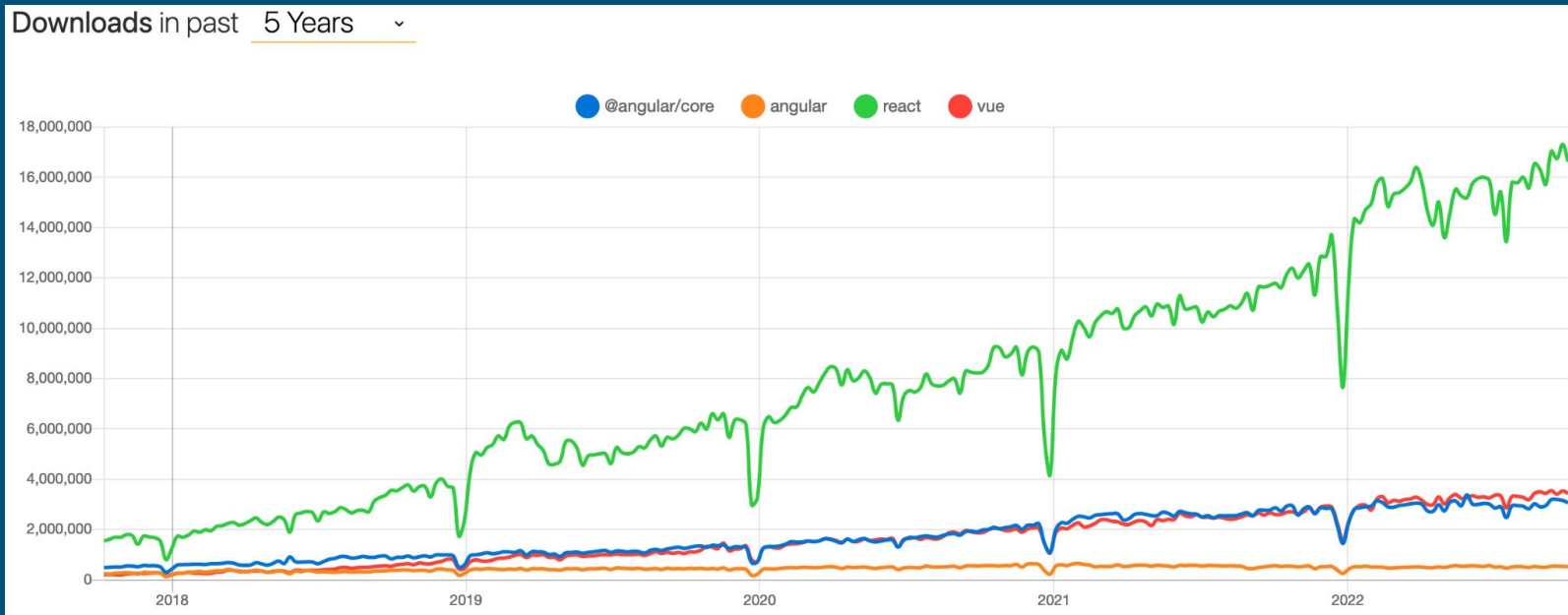
Framework open source para el desarrollo de aplicaciones multiplataforma.

Las aplicaciones se desarrollan utilizando JavaScript y React (este último creado por Facebook y lanzado en el 2015).

Permite usar componentes de UI nativos de iOS y Android (look and feel nativo), y tiene acceso parcial a las capacidades del dispositivo

Trends

<https://npmtrends.com/@angular/core-vs-angular-vs-react-vs-vue>



Ventajas

- Base de código única
- Transición fluida del desarrollador web al desarrollo móvil
- Componentes reutilizables
- Bajos costos de desarrollo
- Menor tiempo de desarrollo
- Actualizaciones en vivo. Los desarrolladores pueden enviar las actualizaciones directamente a los teléfonos de los usuarios en lugar de pasar por el ciclo de actualización de la tienda de aplicaciones.

Desventajas

- Cada plataforma tiene guías de diseño específicas.
- La performance puede verse afectada
- Requiere algún mínimo conocimiento de las API de los sistemas operativos nativos
- No se tiene acceso total al dispositivo.
- Siempre va a estar algo retrasado con respecto a las novedades de los sistemas operativos

Dev / Build Tools

<https://reactnative.dev/docs/environment-setup>

Dos alternativas sólidas

- React Native CLI
- Expo CLI

Dos alternativas sólidas

- **React Native CLI**
 - Herramienta oficial. Otorga libertades al programador
 - Para poder compilar apps para Android e iOS se debe tener instalado Android Studio y xCode (y sus dependencias, productos relacionados)
 - La configuración de cada target es compleja
 - La puesta a punto de la app en su versión de 'producción' es tediosa
- Expo CLI

Dos alternativas sólidas

- React Native CLI
- **Expo CLI**
 - Herramienta no oficial. Restringe las opciones al programador
 - No se necesita tener instalado Android Studio y xCode (y sus dependencias, productos relacionados)
 - No hay que realizar configuraciones específicas para las plataformas
 - No hay que realizar configuraciones específicas para la versión de 'producción'
 - Es posible visualizar de forma ágil el desarrollo desde la misma web de Expo (versión android, ios, y web)
 - No es 100% compatible con todas las librerías
 - Es posible "ejectar" la app de expo, para que la puedas seguir trabajando con React Native CLI. Luego de este paso no hay marcha atrás.

Dos alternativas sólidas

- React Native CLI
- Expo CLI

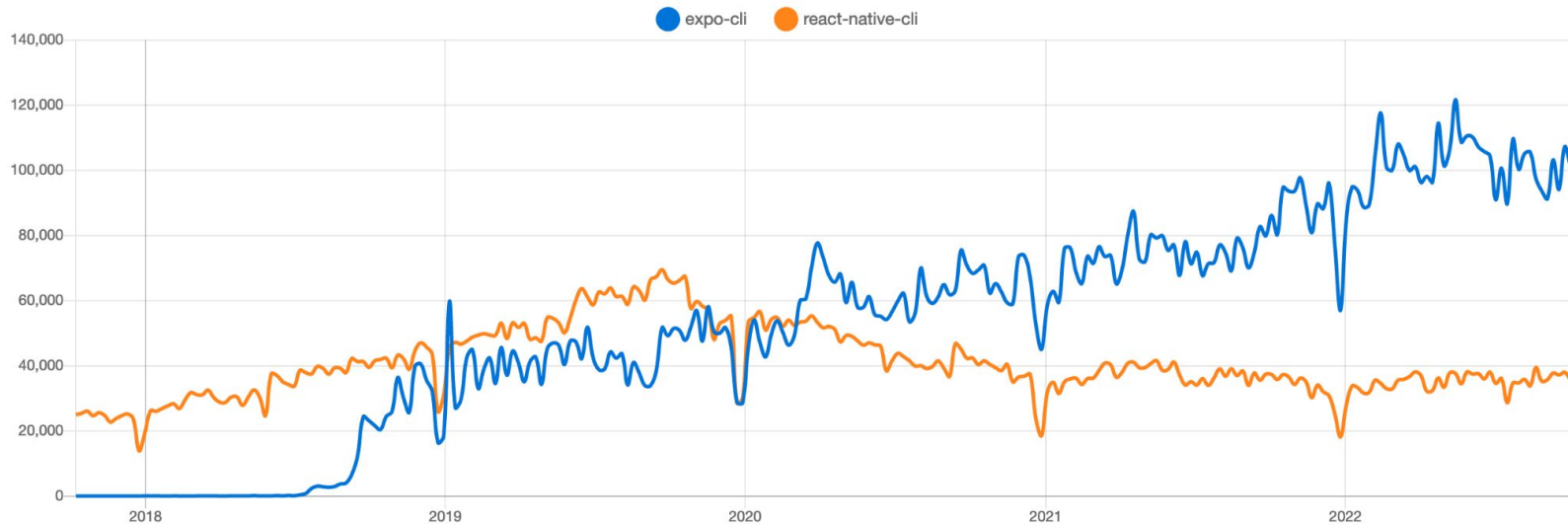
Lectura recomendada:

<https://programmingwithmosh.com/react-native/expo-or-not-building-react-native-apps/>

Trends

<https://www.npmtrends.com/expo-cli-vs-react-native-cli>

Downloads in past 5 Years ▾



Primeros pasos

- Instalar Node
 - <https://nodejs.org/es/download/package-manager/>
- Creación de proyecto
 - `npx create-expo-app NOMBRE_PROYECTO`

```
● ftesone@MacBook-Pro-de-Fernando expo % npx create-expo-app tap-counter
✓ Downloaded and extracted project files.
> npm install
npm WARN deprecated @npmcli/move-file@1.1.2: This functionality has been moved to
npm WARN deprecated @babel/plugin-proposal-numeric-separator@7.18.6: This propo
longer maintained. Please use @babel/plugin-transform-numeric-separator inst
npm WARN deprecated @babel/plugin-proposal-optional-catch-binding@7.18.6: Thi
is no longer maintained. Please use @babel/plugin-transform-optional-catch-bi
npm WARN deprecated @babel/plugin-proposal-nullish-coalescing-operator@7.18.6
ugin is no longer maintained. Please use @babel/plugin-transform-nullish-coal
npm WARN deprecated @babel/plugin-proposal-class-properties@7.18.6: This propo
longer maintained. Please use @babel/plugin-transform-class-properties instea
npm WARN deprecated @babel/plugin-proposal-export-namespace-from@7.18.9: This
s no longer maintained. Please use @babel/plugin-transform-export-namespace-f
npm WARN deprecated @babel/plugin-proposal-optional-chaining@7.21.0: This pro
longer maintained. Please use @babel/plugin-transform-optional-chaining inst
npm WARN deprecated @babel/plugin-proposal-async-generator-functions@7.20.7:
in is no longer maintained. Please use @babel/plugin-transform-async-generato
```

Primeros pasos

- Ya se puede probar nuestra aplicación
 - `cd NOMBRE_PROYECTO`
 - `npm start`



Terminal window showing the Expo CLI interface. The window title is 'ldelia@ldelia-Aspire-R5-471T: ~/Pro'. The main content displays a large QR code for scanning. Below the QR code, instructions are provided for running the app with live reloading, including options for Android emulator, web, email, and signing in. A note mentions that Expo web is in beta and provides a link to report bugs. The 'React' section indicates that the app can be viewed in the browser at `http://localhost:19006/` or `http://192.168.0.247:19006/`. A note states that the development build is not optimized. The 'Expo' section prompts the user to press `?` to show a list of all available commands and mentions that logs for the project will appear below.

ldelia@ldelia-Aspire-R5-471T: ~/Pro



To run the app with live reloading, choose one of:

- Scan the QR code above with the Expo app (Android) or the Camera app (iOS).
- Press **a** for Android emulator, or **w** to run on **w**eb.
- Press **e** to send a link to your phone with email.
- Press **s** to sign in and enable more options.

Expo web is in beta, please report any bugs or missing features on the Expo repo.
You can follow the V1 release for more info: <https://github.com/expo/expo/issues/6782>

React You can now view **RNTest** in the browser.

Local: `http://localhost:19006/`
On Your Network: `http://192.168.0.247:19006/`

Note that the development build is not optimized.

- › To create a production build, run **expo build:web**
- › Press **w** to open the project in browser.
- › Press **Ctrl+C** to exit.

Expo Press **?** to show a list of all available commands.
Logs for your project will appear below. Press **Ctrl+C** to exit.

- Instalar cliente en nuestro dispositivo móvil
 - <https://expo.io/tools#client>
- Escanear código QR y probar nuestra app en el dispositivo

Run your project with Expo Go

Run your project on your own device in seconds with Expo Go.

Download Expo Go



App Store



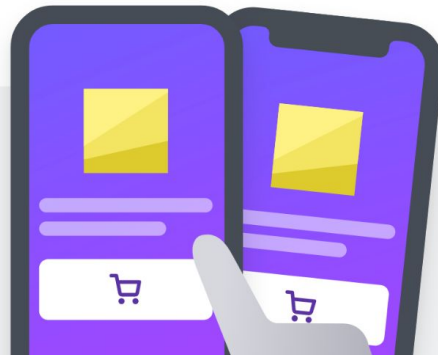
Google Play

Download the [Android .apk archive](#) or the [Apple .ipa archive](#).

```
$ npm start
```

```
Developer tools running on  
http://localhost:19002  
Starting bundler
```

```
> Press a | open Android emulator  
> Press i | open iOS simulator  
> Press w | open web
```



App.js

- Estructura general
- Functional components vs Class components
- Uso del state

Característica	Functional Component	Class Component
Sintaxis	Función	Clase (<code>extends Component</code>)
Estado	<code>useState</code>	<code>this.state</code> / <code>this.setState</code>
Ciclo de vida	<code>useEffect</code>	<code>componentDidMount</code> , <code>componentDidUpdate</code>
Más moderno	✓	✗
Más simple para probar	✓	✗

Ejemplo: tap counter

Mostrar cantidad de *taps* dados a un botón

- Botón para incrementar la cuenta
- Botón para restablecer la cuenta
- *useState*

Cada vez que el estado cambia, React vuelve a renderizar el componente mostrando el nuevo valor.

```
import React, { useState } from "react";

function Contador() {
  // Estado inicial en 0
  const [contador, setContador] = useState(0);

  // Función para incrementar
  const incrementar = () => {
    setContador(contador + 1);
  };

  // Función para restablecer
  const restablecer = () => {
    setContador(0);
  };

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h2>Clicks realizados: {contador}</h2>
      <button onClick={incrementar} style={{ marginRight:
"10px" }}>
        Incrementar
      </button>
      <button onClick={restablecer}>Restablecer</button>
    </div>
  );
}

export default Contador;
```

Ejemplo: tap counter

- Mostrar cantidad de *taps* dados a un botón
- Botón para incrementar la cuenta
- Botón para restablecer la cuenta
- *useState*

```
import React, { Component } from "react";

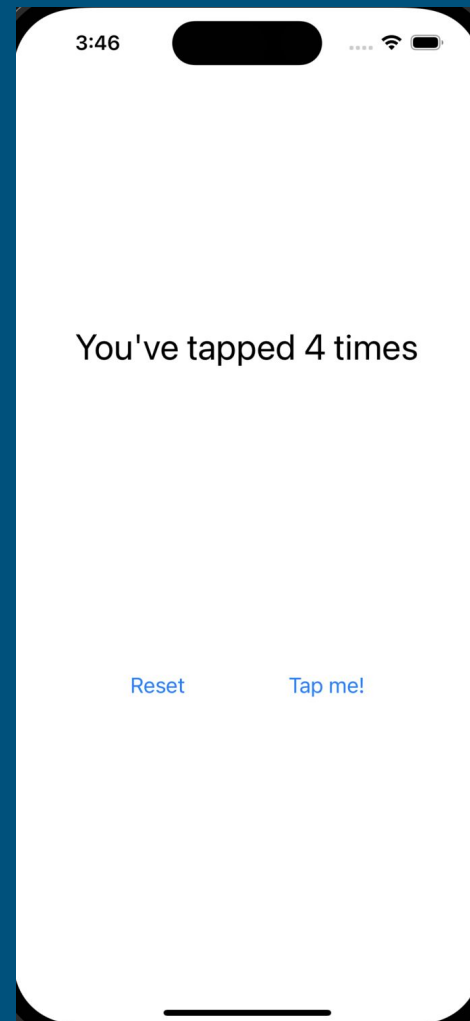
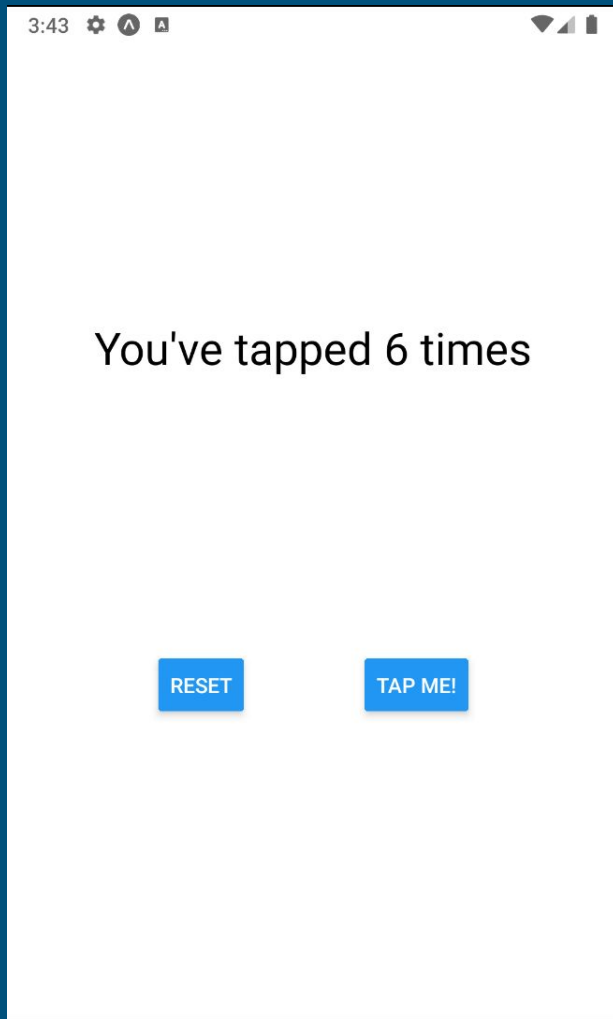
class Contador extends Component {
  constructor(props) {
    super(props);
    this.state = { contador: 0 };
  }

  incrementar = () => {
    this.setState({ contador: this.state.contador + 1 });
  };

  restablecer = () => {
    this.setState({ contador: 0 });
  };

  render() {
    return (
      <div>
        <h2>Clicks realizados: {this.state.contador}</h2>
        <button
          onClick={this.incrementar}>Incrementar</button>
          <button
            onClick={this.restablecer}>Restablecer</button>
        </div>
      );
    }
  }

export default Contador;
```



Functional components vs Class components

Hoy en día, en la mayoría de los proyectos modernos de React se usan casi siempre Functional Components. Razones principales:

- Hooks (useState, useEffect, etc.)
- Solo funcionan en Functional Components.
- Permiten manejar estado y ciclos de vida de forma más clara y concisa que las clases.
- Menos código, más legible
- No necesitas constructor, render() ni this.
- Funciona con funciones normales → más fácil de entender y testear.
- Mejor compatibilidad con librerías modernas
- Muchas librerías actuales asumen que usarás hooks y componentes funcionales.
- Rendimiento
- Functional Components junto con hooks son más fáciles de optimizar con React.memo y otras técnicas.

Demo de una app React Native con Expo

La app permitirá:

- Invocar una API para obtener urls random de fotos
- Mostrar la foto en la app
- Marcar la foto como favorita
- Recuperar las fotos favoritas
- Eliminar foto favorita



API pública para obtener imágenes

- Usaremos un servicio público/gratuito sencillo de usar:
 - <https://picsum.photos/>
- Para obtener una imagen random de 200 de ancho x 300 de alto:
 - <https://picsum.photos/200/300>
- Para obtener una imagen random cuadrada de 200x200:
 - <https://picsum.photos/200>
- Para recuperar una imagen en particular (id 237) de 200 de ancho x 300 de alto:
 - <https://picsum.photos/id/237/200/300>

App.js

- useState → guarda y actualiza información reactiva.

```
const [imageUrl, setImageUrl] = useState(null);
```

- useEffect → ejecuta código al montar o actualizar el componente.
- fetch / then → obtiene datos externos y actualiza el estado.

```
7  ✓  useEffect(() => {  
8      // se ejecuta cuando se monta el componente  
9      loadRandomImage();  
10 }, []);  
11  
12  ✓  /**  
13   * Se ejecuta cuando se presiona el botón "Descubrir nueva" y cuando se abre la app por primera vez  
14   */  
15  ✓  const loadRandomImage = () => {  
16      const width = Math.floor(Dimensions.get('window').width);  
17      const height = Math.floor(Dimensions.get('window').height);  
18      fetch(`https://picsum.photos/${width}/${height}`)  
19  ✓      .then(response => {  
20          setImageUrl(response.url);  
21      })  
22      ;  
23  };
```

App.js

- Estilos
- abstracción similar a CSS

```
const styles = StyleSheet.create({
  main: {
    flex: 1,
    flexDirection: 'column',
  },
  imageHolder: {
    flex: 1
  },
  footer: {
    position: 'absolute',
    left: 0,
    right: 0,
    bottom: 0,
    flexDirection: 'row',
    justifyContent: 'space-between',
    padding: 8,
  },
  image: {
    flex: 1
  }
});
```

App.js

- html

```
return (  
  <View style={styles.main}>  
    <View style={styles.imageHolder}>  
      {imageUrl && <Image style={styles.image} source={{uri: imageUrl}} />}  
    </View>  
    <View style={styles.footer}>  
      <Button title="Descubrir nueva" onPress={loadRandomImage} />  
      <Button title="Mis favoritas" onPress={() => console.log('TODO: ir a Mis Favoritas')} />  
    </View>  
    {loading && (  
      <View style={styles.loading}>  
        <ActivityIndicator size="large" />  
      </View>  
    )}  
    <StatusBar translucent />  
  </View>  
);  
}
```



DESCUBRIR NUEVA

MIS FAVORITAS

Código de la parte 1:

<https://github.com/ftesone/enfoques-photo-discovery-part1>

React Native - Continuación

Continuaremos el desarrollo agregando:

- Posibilidad de marcar como favorita una foto
- Recuperación de fotos favoritas

Navegando entre ventanas

- Al hacer clic en "Mis favoritas" se debe abrir una nueva ventana que liste las fotos favoritas
- Se debe poder volver atrás a la ventana principal
- Necesitaremos instalar algunas dependencias
 - `npm install @react-navigation/native @react-navigation/native-stack`
 - `npx expo install react-native-screens react-native-safe-area-context`

App.js

- Al tener más de una ventana, es incorrecto que el componente donde se descubren fotos se llame 'App'
- Mover su código a un nuevo componente llamado 'Discover' y guardarlo en la carpeta 'screens'
- Crear un nuevo componente (vacío por el momento) llamado 'Favs' y guardarlo en la carpeta 'screens'
- El antiguo componente App define las 'rutas' de los distintos screens

App.js

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { Discover } from './screens/Discover';
import { Favs } from './screens/Favs';

const Stack = createNativeStackNavigator();

export function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Discover">
        <Stack.Screen name="Discover" component={Discover} options={{ headerShown: false }} />
        <Stack.Screen name="Favs" component={Favs} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Navegando entre ventanas

- El componente Discover deberá invocar al screen/componente History cuando se haga click en el botón 'Mis favoritas'
- El componente discover recibirá un objeto de navegación como argumento

```
export function Discover({ navigation }) {  
  const [imageId, setImageId] = useState(null)
```

- Modificar el onPress del botón 'Mis favoritas'

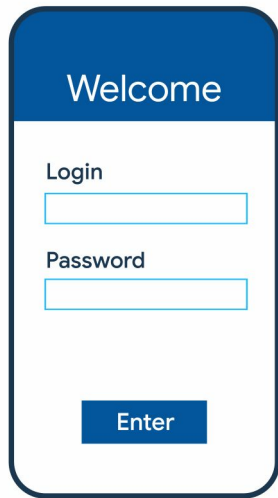
```
<Button title="Discover" onPress={toggleRandomImage} />  
<Button title="Mis favoritas" onPress={() => navigation.navigate('Favs')} />  
</View>
```

State management

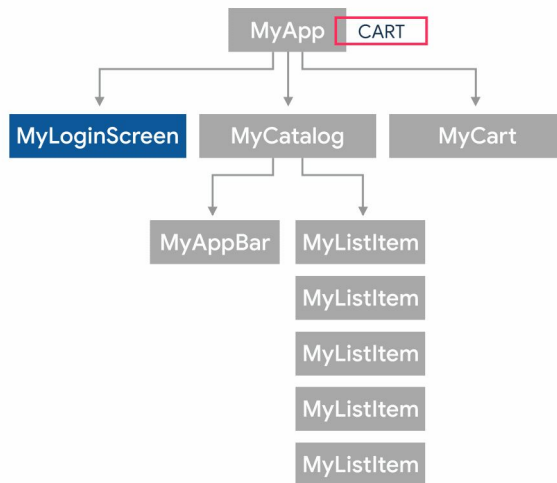
Vimos que cada componente puede tener un estado a través del hook `useState`

¿Pero qué pasa cuando necesitamos compartir el mismo estado entre diferentes componentes?

State management



A mockup of a mobile application login screen. It features a blue header with the text "Welcome". Below the header, there is a "Login" label followed by a text input field. Underneath that is a "Password" label followed by another text input field. At the bottom of the form is a blue button with the text "Enter".



State management

Recordar que React utiliza event-bubbling, lo que significa que cualquier cosa que suceda en un componente, cualquier componente antecesor no se entera.

Un posible enfoque: pasar un estado global a todos los componentes a través de las props.

Redux Toolkit

Wrapper de Redux que provee funciones utilitarias, y formas de organizar mejor y simplificar el código de Redux.

Instalación:

- `npm install react-redux @reduxjs/toolkit`

¡Redux sólo almacena el estado en memoria principal!

- `npm install redux-persist @react-native-async-storage/async-storage`

Async Storage

- Sistema de almacenamiento de clave-valor
- Global a la aplicación
- Asíncrono
- No-encriptado

Redux: Slices

- Redux Toolkit nos permite organizar el store en diferentes partes, denominadas *Slices*
- Creamos una Slice para el state management de favoritas

```
export const favsSlice = createSlice({
  name: 'favs',
  initialState: {
    favs: [],
  },
  reducers: {
    addFav: (state, action) => {
      if (!state.favs.includes(action.payload)) {
        state.favs.push(action.payload);
      }
    },
    removeFav: (state, action) => {
      state.favs = state.favs.filter(fav => fav !== action.payload);
    },
  },
});

export const { addFav, removeFav } = favsSlice.actions;

export default favsSlice.reducer;
```


Redux: Configuración del Store

- Se combinan los Slices en un único Reducer
- Se configura la persistencia utilizando AsyncStorage

```
photo-discovery-part2 > JS store.js > ...
1  import { configureStore, combineReducers } from '@reduxjs/toolkit';
2  import { persistStore, persistReducer } from 'redux-persist';
3  import AsyncStorage from '@react-native-async-storage/async-storage';
4  import favsReducer from '../features/favs/favsSlice';
5
6  const persistConfig = {
7    key: 'root',
8    storage: AsyncStorage,
9  };
10
11  const rootReducer = combineReducers({
12    favs: favsReducer,
13  });
14
15  const persistedReducer = persistReducer(persistConfig, rootReducer);
16
17  export const store = configureStore({
18    reducer: persistedReducer,
19    middleware: getDefaultMiddleware => getDefaultMiddleware({
20      serializableCheck: false,
21    }),
22  });
23
24  export const persistor = persistStore(store);
```

Redux: Provider

- Componente que define un contexto que puede ser accedido por todos sus componentes hijos
- Permite acceder al store a cualquier componente de la aplicación

```
photo-discovery-part2 > JS App.js > ...
You, 29 minutes ago | 1 author (You)
1 import React from 'react';
2 import { Provider } from 'react-redux';
3 import { PersistGate } from 'redux-persist/integration/react';
4 import { store, persistor } from './store';
5 import { NavigationContainer } from '@react-navigation/native';
6 import { createNativeStackNavigator } from '@react-navigation/native-stack';
7 import { Discover } from './screens/Discover';
8 import { Favs } from './screens/Favs';
9
10 const Stack = createNativeStackNavigator();
11
12 export function App() {
13   return (
14     <Provider store={store}>
15       <PersistGate loading={null} persistor={persistor}>
16         <NavigationContainer>
17           <Stack.Navigator initialRouteName="Discover">
18             <Stack.Screen name="Discover" component={Discover} options={{ headerShown: false }} />
19             <Stack.Screen name="Favs" component={Favs} />
20           </Stack.Navigator>
21         </NavigationContainer>
22       </PersistGate>
23     </Provider>
24   );
25 }
26
27 export default App;
```

Marcando fotos como favoritas

- Una foto podrá ser marcada como favorita
- El proceso anterior puede ser revertido
- Ícono de 'corazón' asociado a la foto deberá estar en rojo cuando la foto es favorita, y transparente cuando no lo es.
 - `npm install @expo/vector-icons`

Marcando fotos como favoritas

- Uso de TouchableOpacity
- Uso de Icon
- Dispatch de una action

```

{imageUrl && <Image style={styles.image} source={{uri: imageUrl}} />}
<TouchableOpacity onPress={_ => { favs.includes(imageUrl) ? dispatch(removeFav(imageUrl)) : dispatch(addFav(imageUrl)) }}
  <FontAwesome
    name={favs.includes(imageUrl) ? 'heart' : 'heart-o'}
    size={30}
    color='#F00'
  />
</TouchableOpacity>

```

Marcando fotos como favoritas

- Ya podemos marcar/desmarcar fotos como favoritas !
- Si al descubrir una nueva foto, obtenemos una que ya era favorita, la app detecta y muestra el corazón correspondiente



Recuperando fotos favoritas

- ScrollView
- Map

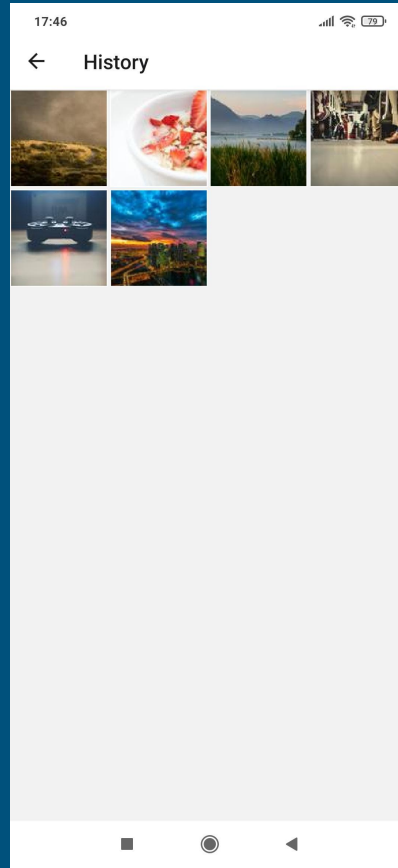
```
export function Favs() {  
  const favs = useSelector(state => state.favs.favs);  
  
  return (  
    <View style={styles.main}>  
      <ScrollView contentContainerStyle={styles.imgGalleryContainer}>  
        {favs.map((imageUrl, index) => (  
          <View key={index} style={styles.itemGallery}>  
            <Image  
              source={{uri: imageUrl}}  
              style={styles.imgGallery}  
              resizeMode="stretch"  
            />  
          </View>  
        ))}  
      </ScrollView>  
    </View>  
  );  
}
```

Recuperando fotos favoritas

- Estilos para la galería de fotos
- Columnas / 4
- Imágenes algo más chica (-2) para tener una separación

```
const styles = StyleSheet.create({
  main: {
    flex: 1,
  },
  imgGalleryContainer: {
    flex: 1,
    flexDirection: "row",
    flexWrap: "wrap",
    alignContent: "flex-start"
  },
  itemGallery: {
    width: COLUMN_WIDTH,
    height: COLUMN_WIDTH,
    alignContent: 'center',
  },
  imgGallery: {
    width: IMAGE_WIDTH,
    height: IMAGE_WIDTH,
    borderWidth: 1,
    borderColor: "#fff",
    margin: 'auto'
  }
});
```

Nuestra app en acción



Código de la parte 2:

<https://github.com/ftesone/enfoques-photo-discovery-part2>