REACT-NATIVE.pdf

quiico_

Desarrollo de interfaces

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

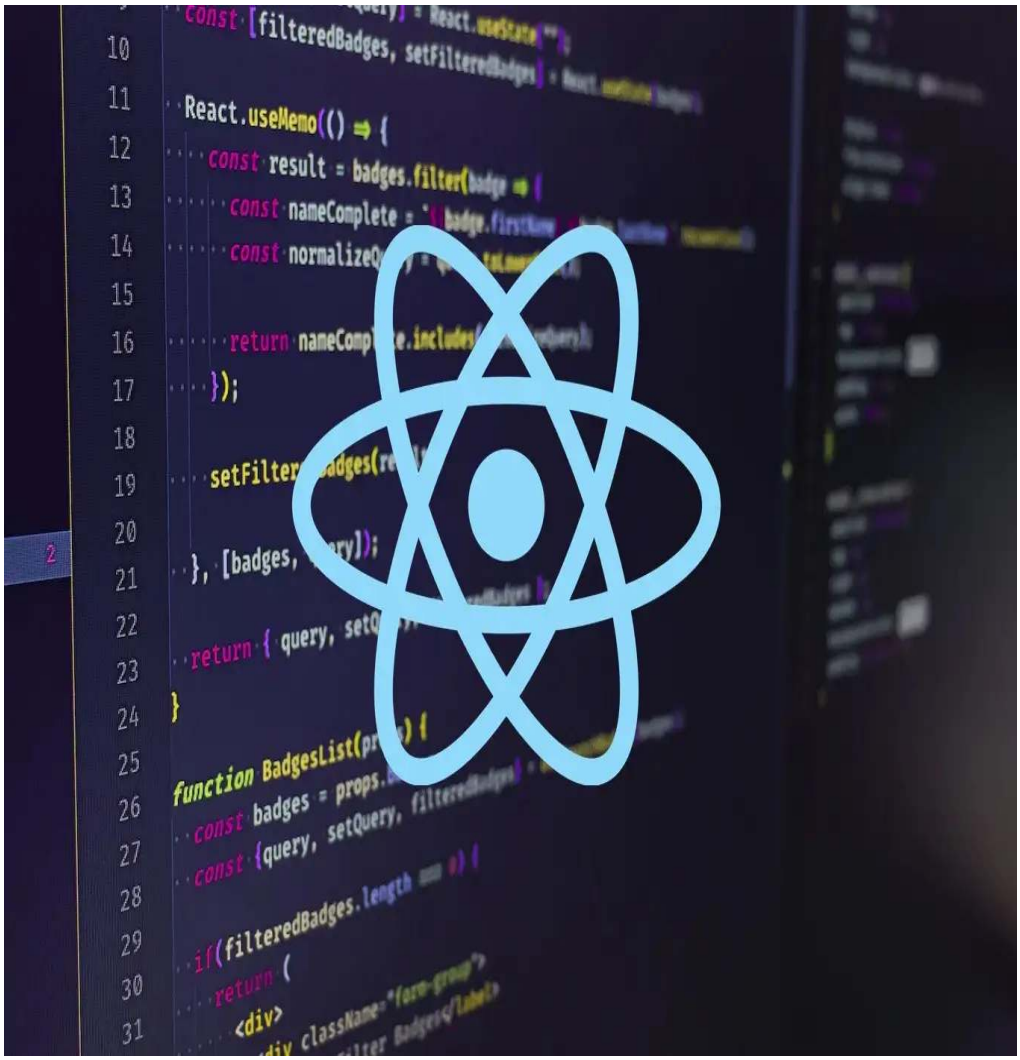
Accede al documento original

REACT NATIVE

2º DAM Curso 2024/2025

Asignatura: Desarrollo de Interfaces

- **Profesor:** Manuel Mauri
- **Alumno:** Francisco José Alonso de Caso Ortiz Nº2



Índice

Introducción

¿Qué es React Native?

React Native es un marco (framework) de desarrollo que permite crear aplicaciones móviles nativas utilizando JavaScript y React. Con React Native, puedes escribir una sola base de código que se ejecuta en Android e iOS.

En lugar de escribir código nativo separado para cada plataforma (Java para Android, Swift o Objective-C para iOS), React Native utiliza componentes nativos, pero la lógica de la aplicación se escribe en JavaScript.

¿Por qué usar React Native?

- **Desarrollo cruzado (Cross-platform)**: Con una sola base de código, puedes generar aplicaciones para Android e iOS.
- **Rendimiento cercano al nativo**: React Native traduce el código JavaScript a componentes nativos, lo que da como resultado un rendimiento bastante bueno, similar al de las aplicaciones nativas.
- **Comunidad activa:** React Native tiene una gran comunidad, lo que significa muchas bibliotecas y soluciones disponibles.
- **Fácil de aprender:** Si ya tienes conocimientos de JavaScript y React, aprender React Native es relativamente sencillo.

Componentes Básicos en React Native

Los componentes son bloques de construcción en React Native. A continuación se muestran algunos de los componentes básicos:

- View: Es como un div en HTML, sirve para agrupar otros componentes.
- Text: Se utiliza para mostrar texto en la pantalla.
- Image: Muestra imágenes.
- TouchableOpacity: Permite hacer que los elementos sean interactivos, como botones.
- TextInput: Para capturar la entrada de texto del usuario.

Conceptos Básicos en React Native

- **JSX (JavaScript XML):** React Native usa JSX, que es una extensión de sintaxis de JavaScript que te permite escribir componentes de forma similar a HTML. Sin embargo, en lugar de etiquetas HTML, JSX usa componentes de React Native como View, Text, etc.

```
import React from 'react';
import { Text, View } from 'react-native';

const App = () => (
  <View>
    <Text>¡Hola, Mundo!</Text>
  </View>
);

export default App;
  return go(f, seed, [])
}
```

- **State:** El state es una forma de almacenar datos dentro de un componente que pueden cambiar con el tiempo (como la entrada de un usuario, la respuesta de una API, etc.).

```
import React, { useState } from 'react';
import { Button, Text, View } from 'react-native';

const App = () => {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>Contador: {count}</Text>
      <Button title="Incrementar" onPress={() => setCount(count + 1)} />
    </View>
  );
};

export default App;
```

- **Estilos en React Native:** Los estilos en React Native se definen de manera similar a CSS, pero en lugar de usar hojas de estilo externas, se usa un objeto JavaScript.

```
const styles = {
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  texto: {
    fontSize: 20,
    color: 'blue',
  },
};

const App = () => (
  <View style={styles.container}>
    <Text style={styles.texto}>Bienvenido a React Native</Text>
  </View>
);
```

Creación de un Nuevo Proyecto Expo

1. **Inicializa el proyecto:**
   npx create-expo-app@latest --template blank
2. **Muévete a la carpeta del proyecto recién creado:**
   cd nombre-del-proyecto
3. **Instala las dependencias del proyecto:**
   npm install
4. **Inicia el proyecto en el entorno de desarrollo:**
   npx expo start

Configuración para Entorno Web

Instala las bibliotecas necesarias para el soporte web:

```
npx expo install react-native-web react-dom
@expo/metro-runtime
```

Esto permitirá que la aplicación funcione en un navegador web.

OFERTAS BLACK FRIDAY

msi

Esta oferta

es como una doble victoria tuya: disfrútala ahora porque no pasa dos veces.

Ver ofertas

HASTA -40%

Tu viejo portátil ya dio lo que tenía que dar. Pásate a MSI: rápido, potente y sin dramas. Lo enciendes y estás listo para todo. Aprovecha las ofertas y despídete del modo “se cuelga cada dos por tres”.

Navegación en React Native

1. **Instala la biblioteca de navegación principal**:
   `npm install @react-navigation/native`
2. **Instala las dependencias adicionales para la navegación**:
   `npx expo install react-native-screens react-native-safe-area-context`
3. **Instala el stack de navegación** para manejar las pantallas de la aplicación:
   `npm install @react-navigation/native-stack`
   Si da error probar:
   `npm install react-native-screens@latest`
   `npm install @react-navigation/native-stack`

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

function HomeScreen({ navigation }) {
  return (
    <View>
      <Text>Pantalla Principal</Text>
      <Button
        title="Ir a Detalles"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

function DetailsScreen() {
  return (
    <View>
      <Text>Detalles de la pantalla</Text>
    </View>
  );
}

const Stack = createStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

Animaciones:

Se utiliza `Animated` y `withSpring` de `react-native-reanimated` para animaciones personalizadas.
Para instalarlo hay que seguir estos pasos

1. **Instala la biblioteca de animación principal**:
   `npm install react-native-reanimated`
2. **Configuración en `babel.config.js`**:

```
module.exports = {
  presets: ['module:metro-react-native-babel-preset'],
  plugins: ['react-native-reanimated/plugin'], // Agrega esta línea
};
```

3. **Detén el servidor de desarrollo si está en ejecución y limpia el caché**:
   `npx react-native start --reset-cache`
4. **Luego reconstruye la app:**
   npx react-native run-android → Para Android
   npx react-native run-ios → Para iOS
5. **Import:**
   import Animated, { withSpring, SharedTransition } from 'react-native-reanimated';

Transiciones Compartidas con `SharedTransition`

- **`SharedTransition.custom`**: Se crea una transición animada personalizada para los elementos compartidos entre pantallas usando `react-native-reanimated`.

```
const transicion = SharedTransition.custom((values) => {
  'worklet';
  return {
    height: withSpring(values.targetHeight, { damping: 10, stiffness: 90 }),
    width: withSpring(values.targetWidth, { damping: 10, stiffness: 90 }),
    originX: withSpring(values.targetOriginX),
    originY: withSpring(values.targetOriginY),
  };
});

  return go(f, seed, [])
}
```

Creación de un Nuevo Proyecto con TypeScript

1. Crear un nuevo proyecto

   Ejecuta el siguiente comando en tu terminal para crear un proyecto con TypeScript:
   npx react-native init NombreDelProyecto --template
   react-native-template-typescript
   Este comando inicializa un proyecto con la plantilla de TypeScript preconfigurada.

2. Navegar al proyecto

   cd NombreDelProyecto

3. Iniciar el proyecto

   Para dispositivos Android:
   npx react-native run-android
   Para dispositivos iOS (requiere macOS con Xcode configurado):
   npx react-native run-ios

4. Editar y añadir tipos

   Puedes empezar a trabajar en archivos `.tsx` y aprovechar los beneficios de TypeScript.

Añadir TypeScript manualmente a un proyecto existente

Si ya tienes un proyecto en React Native, puedes agregar TypeScript con los siguientes pasos:

1. Instalar dependencias necesarias:

   ```
   npm install --save-dev typescript @types/react @types/react-native
   ```

2. Crear un archivo `tsconfig.json` en la raíz del proyecto:

   ```
   {
     "compilerOptions": {
       "target": "esnext",
       "module": "esnext",
       "strict": true,
       "jsx": "react",
       "moduleResolution": "node",
       "allowSyntheticDefaultImports": true,
       "esModuleInterop": true,
       "skipLibCheck": true,
       "resolveJsonModule": true
     }
   }
   ```

Hookers

En React Native (y en React en general), un "hook" es una función especial que permite utilizar funcionalidades como el estado y el ciclo de vida en componentes funcionales, sin necesidad de convertirlos en componentes de clase. Los hooks simplifican el código y permiten reusar lógica entre diferentes componentes de manera más eficiente.

Algunos de los hooks más comunes en React Native son:

- **useState**: permite añadir y manejar el estado en un componente funcional.
- **useEffect**: permite manejar efectos secundarios como peticiones a una API, suscripciones o cambios en el DOM.
- **useContext**: permite acceder al contexto de la aplicación para compartir datos entre componentes sin tener que pasar props manualmente.

Estos hooks (y otros) hacen que el código en React Native sea más modular, limpio y fácil de mantener.

useEffect:

Es un hook que permite ejecutar efectos secundarios en componentes funcionales. Es útil para tareas como:

- Realizar llamadas a una API cuando el componente se monta.
- Configurar suscripciones o temporizadores.
- Actualizar algo en respuesta a cambios en el componente.

```
useEffect(() => {
  console.log("El componente se montó o el estado cambió");
}, []);
```

Si el segundo parámetro es un array vacío [ ], useEffect solo se ejecutará una vez al montar el componente.

useState:

Es un *hook* que permite añadir un estado local a los componentes funcionales. Usa useState para declarar variables que puedan cambiar con el tiempo y actualizar la interfaz en respuesta a esos cambios.

```
const [contador, setContador] = useState(0);

const incrementar = () => setContador(contador + 1);
```

Aquí, contador es la variable de estado, y setContador es la función para actualizar su valor.

Ejemplo del uso de ambos Hooks

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

function Contador() {
  const [contador, setContador] = useState(0);

  useEffect(() => {
    console.log("Contador actualizado a:", contador);
  }, [contador]); // Solo se ejecuta cuando `contador` cambia

  return (
    <div>
      <p>Valor del contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}

export default Contador;
```