



EBook Gratis

APRENDIZAJE react-native

Free unaffiliated eBook created from
Stack Overflow contributors.

#react-
native

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con react-native.....	2
Observaciones.....	2
Examples.....	2
Configuración para Mac.....	2
Configuración para Windows.....	14
Configuración para Linux (Ubuntu).....	15
Inicie el terminal y ejecute los siguientes comandos para instalar nodeJS:.....	15
Si el comando de nodo no está disponible.....	16
Alternativas de instalaciones de NodeJS:.....	16
comprueba si tienes la versión actual.....	16
Ejecuta el npm para instalar el reactivo-nativo.....	16
Android SDK o Android Studio.....	16
Android SDK y ENV.....	16
Ejemplo de inicio de aplicación.....	17
Obs: siempre verifique si la versión de android/app/build.gradle es la misma que la de las.....	18
Abre Android AVD para configurar un androide virtual. Ejecutar la línea de comando:.....	18
Capítulo 2: Accesorios.....	19
Introducción.....	19
Examples.....	19
¿Qué son los accesorios?.....	19
Uso de accesorios.....	19
PropTypes.....	20
Props por defecto.....	21
Capítulo 3: Android - Botón Atrás de Hardware.....	22
Examples.....	22
Detecta pulsaciones del botón Atrás del hardware en Android.....	22
Ejemplo de BackAndroid junto con Navigator.....	22
Ejemplo de detección de botón de retroceso de hardware utilizando BackHandler.....	23
Manejo del botón de retroceso del hardware usando BackHandler y propiedades de navegación.....	23

Capítulo 4: API de animación	25
Examples	25
Animar una imagen	25
Capítulo 5: Componentes	26
Examples	26
Componente basico	26
Componente de estado	26
Componente sin estado	26
Capítulo 6: Crear una APK compatible para Android	28
Introducción	28
Observaciones	28
Examples	28
Crea una clave para firmar el APK	28
Una vez que se genera la clave, úsela para generar la compilación instalable:	28
Genera la construcción usando gradle	28
Sube o comparte el APK generado	28
Capítulo 7: Depuración	30
Sintaxis	30
Examples	30
Iniciar la depuración remota de JS en Android	30
Utilizando console.log ()	30
Capítulo 8: Diseño	31
Examples	31
Flexbox	31
direccion flex	31
Eje de alineación	32
Alineación	34
Tamaño de la flexión	34
Capítulo 9: Ejecutar una aplicación en el dispositivo (versión de Android)	35
Observaciones	35
Examples	35

Ejecutar una aplicación en el dispositivo Android.....	35
Capítulo 10: Enlace de API nativa.....	36
Introducción.....	36
Examples.....	36
Enlaces salientes.....	36
Esquemas URI.....	36
Enlaces entrantes.....	37
Capítulo 11: Enrutamiento.....	38
Introducción.....	38
Examples.....	38
Componente navegador.....	38
Capítulo 12: ESLint en reaccion-nativo.....	39
Introducción.....	39
Examples.....	39
Cómo empezar.....	39
Capítulo 13: Estado.....	40
Sintaxis.....	40
Examples.....	40
setState.....	40
Ejemplo completo.....	40
Inicializar estado.....	42
Capítulo 14: Estilo.....	43
Introducción.....	43
Sintaxis.....	43
Observaciones.....	43
Examples.....	43
Estilismo usando estilos en línea.....	43
Estilo usando una hoja de estilo.....	43
Añadiendo múltiples estilos.....	44
Estilo condicional.....	45
Capítulo 15: Examen de la unidad.....	46

Introducción.....	46
Examples.....	46
Pruebas unitarias con broma.....	46
Prueba de unidad en reaccionar nativo usando broma.....	47
Capítulo 16: Fuentes personalizadas.....	48
Examples.....	48
Pasos para usar fuentes personalizadas en React Native (Android).....	48
Pasos para usar fuentes personalizadas en React Native (iOS).....	48
Fuentes personalizadas tanto para Android como para iOS.....	49
Androide.....	50
iOS.....	51
Capítulo 17: Hola Mundo.....	52
Examples.....	52
Editando index.ios.js o index.android.js.....	52
¡Hola Mundo!.....	52
Capítulo 18: Imágenes.....	53
Examples.....	53
Módulo de imagen.....	53
Ejemplo de imagen.....	53
Fuente de imagen condicional.....	53
Usando variable para ruta de imagen.....	53
Para encajar una imagen.....	54
Capítulo 19: Instrucciones de línea de comando.....	55
Examples.....	55
Comprobar versión instalada.....	55
Actualizar el proyecto existente a la última versión de RN.....	55
Explotación florestal.....	55
Inicializa y comienza con el proyecto React Native.....	55
Iniciar React Packing nativo.....	56
Añadir proyecto de Android para su aplicación.....	56
Capítulo 20: Integración con Firebase para la autenticación.....	57
Introducción.....	57

Examples.....	57
Reaccionar nativo - ListView con Firebase.....	57
Autenticación en React Native usando Firebase.....	58
Capítulo 21: Mejores Prácticas de Navegador.....	60
Examples.....	60
Navegador.....	60
Utilice react-navigation para navegar en reaccionar aplicaciones nativas.....	62
navegación nativa de reacción con flujo de enrutador nativo de reacción.....	63
Capítulo 22: Mejores Prácticas de Render.....	65
Introducción.....	65
Examples.....	65
Funciones en JSX.....	65
Capítulo 23: Modal.....	67
Introducción.....	67
Parámetros.....	67
Examples.....	67
Ejemplo Modal Básico.....	67
Ejemplo Modal Transparente.....	68
Capítulo 24: Módulo de plataforma.....	70
Examples.....	70
Encuentra el tipo de sistema operativo / versión.....	70
Capítulo 25: Módulos nativos.....	71
Examples.....	71
Crea tu módulo nativo (IOS).....	71
Introducción.....	71
Ejemplo.....	71
Capítulo 26: Navegador con botones inyectados desde páginas.....	73
Examples.....	73
Introducción.....	73
Ejemplo completo comentado.....	73
Capítulo 27: Notificación de inserción.....	77

Introducción.....	77
Observaciones.....	77
Examples.....	77
Configuración simple de notificaciones push.....	77
Navegando a la escena desde Notificación.....	79
Capítulo 28: RefreshControl con ListView.....	82
Observaciones.....	82
Examples.....	82
Control de actualización.....	82
onRefresh function Ejemplo.....	82
Refresque el control con ListView Ejemplo completo.....	82
Capítulo 29: Representación de múltiples apoyos.....	85
Examples.....	85
render multiples variables.....	85
Capítulo 30: Solicitudes HTTP.....	86
Sintaxis.....	86
Observaciones.....	86
Examples.....	86
Websockets.....	86
HTTP con la API fetch.....	86
Redes con XMLHttpRequest.....	87
Usando Promesas con la API de búsqueda y Redux.....	87
Web Socket con Socket.io.....	88
Http con axios.....	89
Capítulo 31: Vista de la lista.....	91
Examples.....	91
Ejemplo simple.....	91
Capítulo 32: WebView.....	92
Introducción.....	92
Examples.....	92
Componente simple utilizando webview.....	92
Creditos.....	93

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [react-native](#)

It is an unofficial and free react-native ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official react-native.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con react-native

Observaciones

React Native te permite crear aplicaciones móviles usando solo JavaScript. Utiliza el mismo diseño que React, lo que le permite componer una rica interfaz de usuario móvil a partir de componentes declarativos.

Con React Native, no crea una "aplicación web móvil", una "aplicación HTML5" o una "aplicación híbrida". Usted construye una aplicación móvil real que no se puede distinguir de una aplicación creada con Objective-C o Java. React Native utiliza los mismos bloques de construcción fundamentales de UI que las aplicaciones normales de iOS y Android. Simplemente pones esos bloques de construcción juntos usando JavaScript y React.

Es de código abierto y mantenido por Facebook.

- [Sitio web](#)
- [Documentación](#)
- [Repositorio GitHub](#)

Fuente: [sitio nativo React Native](#)

Examples

Configuración para Mac

Instalación del gestor de paquetes Homebrew `brew`

Pega eso en un aviso de Terminal.

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install) "
```

Instalación de Xcode IDE

Descárguelo utilizando el enlace a continuación o búsquelo en Mac App Store

<https://developer.apple.com/download/>

NOTA: Si tiene **Xcode-beta.app** instalado junto con la versión de producción de **Xcode.app**, asegúrese de estar usando la versión de producción de la herramienta `xcodebuild`. Puedes configurarlo con:

```
sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer/
```

Instalación del entorno Android

- `Git git`

* Si ha instalado XCode, Git ya está instalado, de lo contrario ejecute lo siguiente

```
brew install git
```

- [Último JDK](#)
- [Android Studio](#)

Elija una instalación personalizada



Install Type

Choose the type of setup you want for Android Studio

Standard

Android Studio will be installed with the most recommended settings.
Recommended for most users.

Custom

You can customize installation settings and

Elija tanto el rendimiento como el dispositivo virtual de Android



SDK Component

Check the components you want






- Android SDK – (installed)
- Android SDK Platform
- API 23: Android 6.0 (Ma
- Performance (Intel ® HAXM
- Android Virtual Device – (i

Después de la instalación, elija Configurar -> Administrador de SDK en la ventana de bienvenida de Android Studio.



Android Studio

Version 2.3.3

-  Start a new Android Studio project
-  Open an existing Android Studio project
-  Check out project from version control
-  Import project (Eclipse or Gradle)
-  Import an Android code repository

En la ventana de Plataformas del SDK, elija Mostrar detalles del paquete y en Android 6.0 (Marshmallow), asegúrese de que las API de Google, la imagen del sistema Intel x86 Atom, la imagen del sistema Intel x86 Atom_64 y la API del sistema Intel x86 Atom_64 estén marcadas.

En la ventana de Herramientas del SDK, elija Mostrar detalles del paquete y en Herramientas de compilación del SDK de Android, asegúrese de que la herramienta de construcción del SDK de Android 23.0.1 esté seleccionada.

- Variable de entorno `ANDROID_HOME`

Asegúrese de que la variable de entorno `ANDROID_HOME` apunte a su SDK de Android existente. Para hacer eso, agregue esto a su `~ / .bashrc`, `~ / .bash_profile` (o lo que sea que use su shell) y vuelva a abrir su terminal:

Si instaló el SDK sin Android Studio, entonces podría ser algo como: `/usr/local/opt/android-sdk`

```
export ANDROID_HOME=~/.Library/Android/sdk
```

Dependencias para Mac

Necesitará Xcode para iOS y Android Studio para Android, node.js, las herramientas de línea de comandos React Native y Watchman.

Recomendamos la instalación de node y watchman a través de Homebrew.

```
brew install node
brew install watchman
```

[Watchman](#) es una herramienta de Facebook para observar los cambios en el sistema de archivos. Es altamente recomendable que lo instale para un mejor rendimiento. Es opcional.

El nodo viene con npm, que le permite instalar la interfaz de línea de comandos de React Native.

```
npm install -g react-native-cli
```

Si obtiene un error de permiso, intente con sudo:

```
sudo npm install -g react-native-cli.
```

Para iOS, la forma más fácil de instalar Xcode es a través de la Mac App Store. Y para Android descargar e instalar Android Studio.

Si planea realizar cambios en el código Java, recomendamos Gradle Daemon, que acelera la compilación.

Probando su instalación nativa React

Use las herramientas de línea de comando React Native para generar un nuevo proyecto React Native llamado "AwesomeProject", luego ejecute run-ios nativo reactivo dentro de la carpeta recién creada.

```
react-native init AwesomeProject
cd AwesomeProject
react-native run-ios
```

Debería ver su nueva aplicación ejecutándose en el simulador de iOS en breve. react-native run-ios es solo una forma de ejecutar su aplicación: también puede ejecutarla directamente desde Xcode o Nuclide.

Modificar tu aplicación

Ahora que has ejecutado exitosamente la aplicación, modifiquémosla.

- Abra index.ios.js o index.android.js en el editor de texto de su elección y edite algunas líneas.
- ¡Presiona Command + R en tu simulador de iOS para recargar la aplicación y ver tu cambio! ¡Eso es!

¡Felicidades! Has ejecutado y modificado con éxito tu primera aplicación React Native.

fuelle: [Getting Started - React-Native](#)

Configuración para Windows

Nota: no puede desarrollar aplicaciones nativas de reacción para iOS en Windows, solo aplicaciones android nativas de reacción.

Los documentos de configuración oficiales para reaccion-native en windows se pueden [encontrar aquí](#) . Si necesitas más detalles hay una [guía granular aquí](#) .

Herramientas / Medio Ambiente

- Windows 10
- herramienta de línea de comandos (por ejemplo, Powershell o línea de comandos de Windows)
- [Chocolatey](#) ([pasos para configurar via PowerShell](#))
- El JDK (versión 8)
- Android Studio
- Una máquina Intel con tecnología de virtualización habilitada para HAXM (opcional, solo necesaria si desea usar un emulador)

1) Configure su máquina para reaccionar al desarrollo nativo

Inicie la línea de comandos como administrador, ejecute los siguientes comandos:

```
choco install nodejs.install
choco install python2
```

Reinicie la línea de comandos como administrador para que pueda ejecutar npm

```
npm install -g react-native-cli
```

Después de ejecutar el último comando, copie el directorio en el que se instaló reaccion-native. Lo necesitará para el Paso 4. Lo intenté en dos computadoras en un caso: C:\Program Files (x86)\Nodist\v-x64\6.2.2

. En el otro era: C:\Users\admin\AppData\Roaming\npm

2) Establezca sus variables de entorno

Puede encontrar una guía paso a paso con imágenes aquí para esta sección.

Abra la ventana Variables de entorno navegando a:

[Clic derecho] Menú "Inicio" -> Sistema -> Configuración avanzada del sistema -> Variables de entorno

En la sección inferior, busque la variable del sistema "Ruta" y agregue la ubicación donde se instaló react-native en el paso 1.

Si no ha agregado una variable de entorno ANDROID_HOME, también deberá hacerlo aquí. Mientras aún se encuentre en la ventana "Variables de entorno", agregue una nueva Variable del sistema con el nombre "ANDROID_HOME" y valore como la ruta a su SDK de Android.

Luego reinicie la línea de comandos como administrador para que pueda ejecutar comandos react-native en ella.

3) Cree su proyecto En la línea de comandos, navegue a la carpeta donde desea colocar su proyecto y ejecute el siguiente comando:

```
react-native init ProjectName
```

4) Ejecute su proyecto Inicie un emulador desde android studio Navegue hasta el directorio raíz de su proyecto en la línea de comandos y ejecútelo:

```
cd ProjectName  
react-native run-android
```

Puede encontrarse con problemas de dependencia. Por ejemplo, puede haber un error que no tenga la versión correcta de las herramientas de compilación. Para solucionar este problema, deberá abrir [el administrador sdk en Android Studio](#) y descargar las herramientas de compilación desde allí.

Felicidades

Para actualizar la interfaz de usuario, puede presionar la tecla `r` dos veces mientras está en el emulador y ejecuta la aplicación. Para ver las opciones de desarrollador puedes presionar `ctrl + m`.

Configuración para Linux (Ubuntu)

1) Setup Node.JS

Inicie el terminal y ejecute los siguientes

comandos para instalar nodeJS:

```
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash -  
sudo apt-get install nodejs
```

Si el comando de nodo no está disponible

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Alternativas de instalaciones de NodeJS:

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

o

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

comprueba si tienes la versión actual

```
node -v
```

Ejecuta el npm para instalar el reactivo-nativo

```
sudo npm install -g react-native-cli
```

2) Configurar Java

```
sudo apt-get install lib32stdc++6 lib32z1 openjdk-7-jdk
```

3) Configurar Android Studio:

Android SDK o Android Studio

```
http://developer.android.com/sdk/index.html
```

Android SDK y ENV

```
export ANDROID_HOME=/YOUR/LOCAL/ANDROID/SDK
```

```
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

4) Emulador de configuración:

En la terminal ejecuta el comando

```
android
```

Seleccione "Plataformas SDK" en el Administrador de SDK y debería ver una marca de verificación azul junto a "Android 7.0 (Nougat)". En caso de que no lo sea, haga clic en la casilla de verificación y luego en "Aplicar".

Appearance & Behavior > System Settings > Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location:

[Edit](#)

SDK Platforms

SDK Tools

SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
<input checked="" type="checkbox"/> Android 7.0 (Nougat)	24	2	Installed
<input type="checkbox"/> Android 6.0 (Marshmallow)	23	3	Not installed
<input type="checkbox"/> Android 5.1 (Lollipop)	22	2	Not installed
<input type="checkbox"/> Android 5.0 (Lollipop)	21	2	Not installed
<input type="checkbox"/> Android 4.4 (KitKat Wear)	20	2	Not installed
<input type="checkbox"/> Android 4.4 (KitKat)	19	4	Not installed
<input type="checkbox"/> Android 4.3 (Jelly Bean)	18	3	Not installed
<input type="checkbox"/> Android 4.2 (Jelly Bean)	17	3	Not installed
<input type="checkbox"/> Android 4.1 (Jelly Bean)	16	5	Not installed
<input type="checkbox"/> Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
<input type="checkbox"/> Android 4.0 (IceCreamSandwich)	14	4	Not installed
<input type="checkbox"/> Android 3.2 (Honeycomb)	13	1	Not installed
<input type="checkbox"/> Android 3.1 (Honeycomb)	12	3	Not installed
<input type="checkbox"/> Android 3.0 (Honeycomb)	11	2	Not installed
<input type="checkbox"/> Android 2.3.3 (Gingerbread)	10	2	Not installed
<input type="checkbox"/> Android 2.3 (Gingerbread)	9	2	Not installed
<input type="checkbox"/> Android 2.2 (Froyo)	8	3	Not installed

Show Package Details

[Launch Standalone SDK Manager](#)

5) Iniciar un proyecto.

Ejemplo de inicio de aplicación

```
react-native init ReactNativeDemo && cd ReactNativeDemo
```


Obs: siempre verifique si la versión de `android/app/build.gradle` es la misma que la de las herramientas de construcción descargadas en su SDK de Android

```
android {  
  compileSdkVersion XX  
  buildToolsVersion "XX.X.X"  
  ...  
}
```

6) Ejecutar el proyecto

**Abre Android AVD para configurar un androide virtual.
Ejecutar la línea de comando:**

```
android avd
```

Sigue las instrucciones para crear un dispositivo virtual e iniciarlo.

Abre otro terminal y ejecuta las líneas de comando:

```
react-native run-android  
react-native start
```

Lea **Empezando con react-native en línea**: <https://riptutorial.com/es/react-native/topic/857/empezando-con-react-native>

Capítulo 2: Accesorios

Introducción

Los apoyos, o propiedades, son datos que se pasan a componentes secundarios en una aplicación React. Los componentes de React generan elementos de la interfaz de usuario en función de sus propiedades y su estado interno. Los accesorios que un componente toma (y usa) definen cómo se puede controlar desde el exterior.

Examples

¿Qué son los accesorios?

Los apoyos se utilizan para transferir datos del componente principal al secundario. Los apoyos son de solo lectura. El componente hijo solo puede obtener los accesorios pasados de los padres utilizando **this.props.keyName** . Usando puntales uno puede hacer su componente reutilizable.

Uso de accesorios

Una vez finalizada la configuración. Copie el código siguiente a `index.android.js` o al archivo `index.ios.js` para usar los accesorios.

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

class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}

AppRegistry.registerComponent('LotsOfGreetings', () => LotsOfGreetings);
```

Usando puntales uno puede hacer su componente genérico. Por ejemplo, tienes un componente Button. Puede pasar diferentes apoyos a ese componente, de modo que uno pueda colocar ese botón en cualquier lugar en su vista.

PropTypes

El paquete `prop-types` permite agregar una verificación de tipo de tiempo de ejecución a su componente que garantice que los tipos de props pasados al componente son correctos. Por ejemplo, si no pasa un `name` o una `isYummy` al componente debajo, se producirá un error en el modo de desarrollo. En el modo de producción no se realizan las comprobaciones del tipo de apoyo. La definición de `propTypes` puede hacer que su componente sea más fácil de leer y mantener.

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { AppRegistry, Text, View } from 'react-native';

import styles from './styles.js';

class Recipe extends Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    isYummy: PropTypes.bool.isRequired
  }
  render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.name}</Text>
        {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
      </View>
    )
  }
}

AppRegistry.registerComponent('Recipe', () => Recipe);

// Using the component
<Recipe name="Pancakes" isYummy={true} />
```

PropTypes múltiples

También puede tener varios `propTypes` de `propTypes` para uno. Por ejemplo, los accesorios de nombre que estoy tomando también pueden ser un objeto, puedo escribirlo como.

```
static propTypes = {
  name: PropTypes.oneOfType([
    PropTypes.string,
    PropTypes.object
  ])
}
```

Accesorios para niños

También hay un accesorio especial llamado `children`, que **no se** transmite como

```
<Recipe children={something}/>
```

En su lugar, deberías hacer esto

```
<Recipe>
  <Text>Hello React Native</Text>
</Recipe>
```

entonces puedes hacer esto en el render de la receta:

```
return (
  <View style={styles.container}>
    {this.props.children}
    {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
  </View>
)
```

Tendrá un componente `<Text>` en su `Recipe` dice: "¿ Hello React Native , Hello React Native ?

Y el `propTipo` de niños es

```
children: PropTypes.node
```

Props por defecto

`defaultProps` le permite establecer valores de `prop` predeterminados para su componente. En el ejemplo a continuación, si no pasa los accesorios de nombre, mostrará a John; de lo contrario, mostrará el valor pasado

```
class Example extends Component {
  render() {
    return (
      <View>
        <Text>{this.props.name}</Text>
      </View>
    )
  }
}

Example.defaultProps = {
  name: 'John'
}
```

Lea Accesorios en línea: <https://riptutorial.com/es/react-native/topic/1271/accesorios>

Capítulo 3: Android - Botón Atrás de Hardware

Examples

Detecta pulsaciones del botón Atrás del hardware en Android

```
BackAndroid.addEventListener('hardwareBackPress', function() {
  if (!this.onMainScreen()) {
    this.goBack();
    return true;
  }
  return false;
});
```

Nota: `this.onMainScreen()` y `this.goBack()` no están integradas en funciones, también debe implementarlas. (<https://github.com/immidi/react-native/commit/ed7e0fb31d842c63e8b8dc77ce795fac86e0f712>)

Ejemplo de BackAndroid junto con Navigator

Este es un ejemplo de cómo usar el `BackAndroid` React Native junto con el `Navigator` .

`componentWillMount` registra un detector de eventos para manejar los toques en el botón Atrás. Comprueba si hay otra vista en la pila de historial, y si la hay, retrocede; de lo contrario, mantiene el comportamiento predeterminado.

Más información sobre la [documentación de BackAndroid](#) y la [documentación de Navigator](#) .

```
import React, { Component } from 'react'; // eslint-disable-line no-unused-vars

import {
  BackAndroid,
  Navigator,
} from 'react-native';

import SceneContainer from './Navigation/SceneContainer';
import RouteMapper from './Navigation/RouteMapper';

export default class AppContainer extends Component {

  constructor(props) {
    super(props);

    this.navigator;
  }

  componentWillMount() {
    BackAndroid.addEventListener('hardwareBackPress', () => {
      if (this.navigator && this.navigator.getCurrentRoutes().length > 1) {
```

```

        this.navigator.pop();
        return true;
    }
    return false;
});
}

renderScene(route, navigator) {
    this.navigator = navigator;

    return (
        <SceneContainer
            title={route.title}
            route={route}
            navigator={navigator}
            onBack={() => {
                if (route.index > 0) {
                    navigator.pop();
                }
            }}
            {...this.props} />
    );
}

render() {
    return (
        <Navigator
            initialRoute={<View />}
            renderScene={this.renderScene.bind(this)}
            navigationBar={
                <Navigator.NavigationBar
                    style={{backgroundColor: 'gray'}}
                    routeMapper={RouteMapper} />
            } />
    );
}
};

```

Ejemplo de detección de botón de retroceso de hardware utilizando BackHandler

Desde BackAndroid está en desuso. Utilice BackHandler en lugar de BackAndroid.

```

import { BackHandler } from 'react-native';

{...}
ComponentWillMount() {
    BackHandler.addEventListener('hardwareBackPress', ()=>{
        if (!this.onMainScreen()) {
            this.goBack();
            return true;
        }
        return false;
    });
}

```

Manejo del botón de retroceso del hardware usando BackHandler y

propiedades de navegación (sin usar BackAndroid y Navigator en desuso)

Este ejemplo le mostrará la navegación anterior que se espera generalmente en la mayoría de los flujos. Deberá agregar el siguiente código a cada pantalla dependiendo del comportamiento esperado. Hay 2 casos:

1. Si hay más de 1 pantalla en la pila, el botón de retroceso del dispositivo mostrará la pantalla anterior.
2. Si solo hay 1 pantalla en la pila, el botón de retroceso del dispositivo saldrá de la aplicación.

Caso 1: Mostrar pantalla anterior

```
import { BackHandler } from 'react-native';

constructor(props) {
  super(props)
  this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
}

componentWillMount() {
  BackHandler.addEventListener('hardwareBackPress', this.handleBackButtonClick);
}

componentWillUnmount() {
  BackHandler.removeEventListener('hardwareBackPress', this.handleBackButtonClick);
}

handleBackButtonClick() {
  this.props.navigation.goBack(null);
  return true;
}
```

Importante: No olvide enlazar el método en el constructor y eliminar la escucha en `componentWillUnmount`.

Caso 2: Salir de la aplicación

En este caso, no es necesario manejar nada en esa pantalla en la que desea salir de la aplicación.

Importante: Esto debe ser solo pantalla en la pila.

Lea Android - Botón Atrás de Hardware en línea: <https://riptutorial.com/es/react-native/topic/4668/android---boton-atras-de-hardware>

Capítulo 4: API de animación

Examples

Animar una imagen

```
class AnimatedImage extends Component {
  constructor(props) {
    super(props)
    this.state = {
      logoMarginTop: new Animated.Value(200)
    }
  }
  componentDidMount() {
    Animated.timing(
      this.state.logoMarginTop,
      { toValue: 100 }
    ).start()
  }
  render () {
    return (
      <View>
        <Animated.Image source={require('../images/Logo.png')} style={[baseStyles.logo, {
          marginTop: this.state.logoMarginTop
        }]} />
      </View>
    )
  }
}
```

Este ejemplo anima la posición de la imagen cambiando el margen.

Lea API de animación en línea: <https://riptutorial.com/es/react-native/topic/4415/api-de-animacion>

Capítulo 5: Componentes

Examples

Componente basico

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

class Example extends Component {
  render () {
    return (
      <View>
        <Text> I'm a basic Component </Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

Componente de estado

Estos componentes tendrán estados cambiantes.

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

class Example extends Component {
  constructor (props) {
    super(props)
    this.state = {
      name: "Sriraman"
    }
  }
  render () {
    return (
      <View>
        <Text> Hi, {this.state.name}</Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

Componente sin estado

Como su nombre lo indica, Stateless Components no tiene ningún estado local. También son conocidos como **componentes tontos** . Sin ningún estado local, estos componentes no necesitan métodos de ciclo de vida o gran parte de la placa de calderas que viene con un componente de estado.

La sintaxis de clase no es obligatoria, simplemente puedes hacer `const name = ({props}) => (...)` . En general, los componentes sin estado son más concisos como resultado.

Debajo hay un ejemplo de dos componentes sin estado, la `App` y el `Title` , con una demostración de aprobación de apoyos entre componentes:

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

const Title = ({Message}) => (
  <Text>{Message}</Text>
)

const App = () => (
  <View>
    <Title title='Example Stateless Component' />
  </View>
)

AppRegistry.registerComponent('App', () => App)
```

Este es el patrón recomendado para los componentes, cuando sea posible. Como en el futuro, se pueden hacer optimizaciones para estos componentes, reduciendo las asignaciones de memoria y las verificaciones innecesarias.

Lea Componentes en línea: <https://riptutorial.com/es/react-native/topic/5532/componentes>

Capítulo 6: Crear una APK compatible para Android

Introducción

Pasos para crear un APK (firmado y sin firmar) que puede instalar en un dispositivo usando CLI y compartir también:

Declaración de problema: he creado mi aplicación, puedo ejecutarla en mi emulador local (y también en mi dispositivo Android cambiando el servidor de depuración). Pero, quiero crear una apk que pueda enviar a alguien sin acceso al servidor de desarrollo y quiero que puedan probar la aplicación.

Observaciones

Una descripción más detallada también se menciona aquí: <https://facebook.github.io/react-native/docs/signed-apk-android.html>

Examples

Creando una clave para firmar el APK

```
keytool -genkey -v -keystore my-app-key.keystore -alias my-app-alias -keyalg RSA -keysize 2048 -validity 10000
```

Use una contraseña cuando se le solicite

Una vez que se genera la clave, úsela para generar la compilación instalable:

```
react-native bundle --platform android --dev false --entry-file index.android.js \
--bundle-output android/app/src/main/assets/index.android.bundle \
--assets-dest android/app/src/main/res/
```

Genera la construcción usando gradle

```
cd android && ./gradlew assembleRelease
```

Sube o comparte el APK generado.

Sube el APK a tu teléfono. La bandera -r reemplazará la aplicación existente (si existe)

```
adb install -r ./app/build/outputs/apk/app-release-unsigned.apk
```

El APK firmado compatible se encuentra en:

```
./app/build/outputs/apk/app-release.apk
```

Lea [Crear una APK compatible para Android en línea](https://riptutorial.com/es/react-native/topic/8964/crear-una-apk-compartible-para-android): <https://riptutorial.com/es/react-native/topic/8964/crear-una-apk-compartible-para-android>

Capítulo 7: Depuración

Sintaxis

- depurador

Examples

Iniciar la depuración remota de JS en Android

Puede iniciar la depuración remota desde el menú Desarrollador. Después de seleccionar habilitar la depuración remota, se abrirá Google Chrome, para que pueda registrar el resultado en su consola. También puede escribir la sintaxis del depurador en su código js.

Utilizando `console.log ()`

Puede imprimir un mensaje de registro en el terminal usando `console.log()` . Para hacerlo, abre una nueva terminal y ejecuta el siguiente comando para Android:

```
react-native log-android
```

o siguiendo el comando si está utilizando iOS:

```
react-native log-ios
```

Ahora empezará a ver todos los mensajes de registro en este terminal.

Lea Depuración en línea: <https://riptutorial.com/es/react-native/topic/5105/depuracion>

Capítulo 8: Diseño

Examples

Flexbox

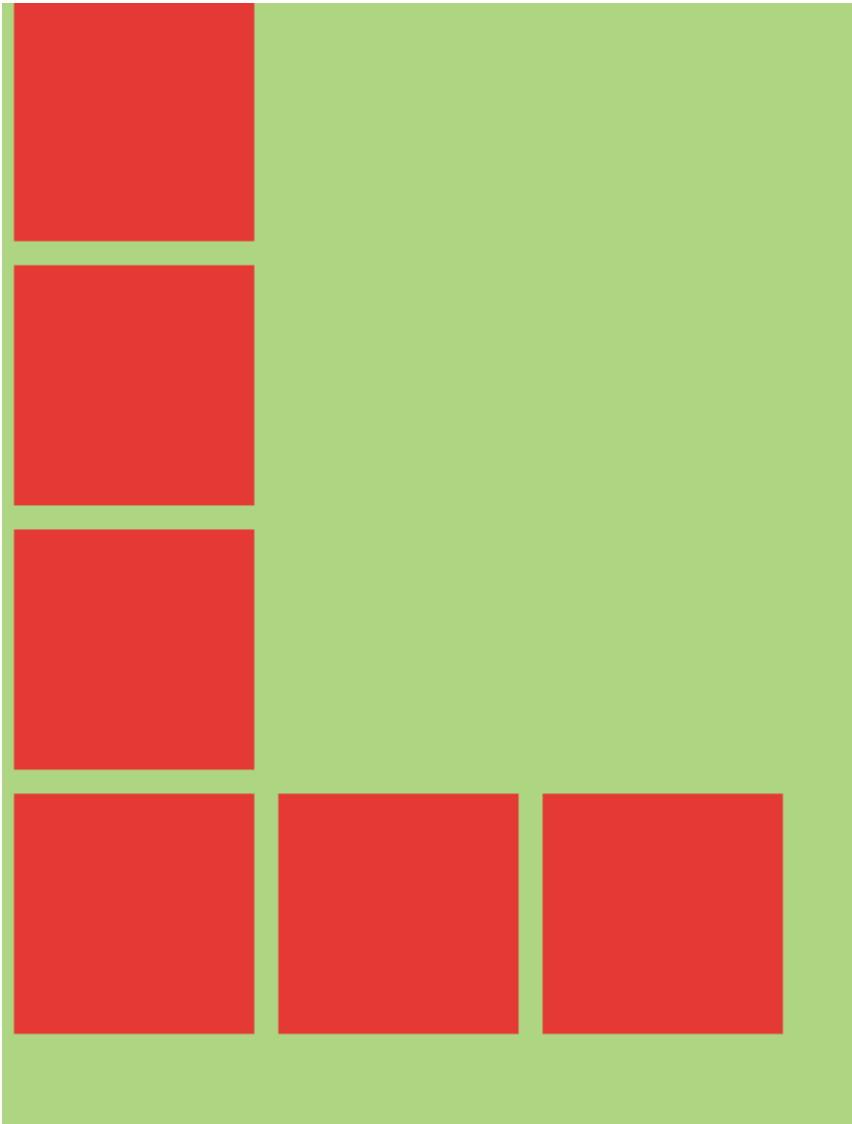
Flexbox es un modo de diseño que proporciona la disposición de elementos en una página, de manera que los elementos se comportan de manera predecible cuando el diseño de la página debe adaptarse a diferentes tamaños de pantalla y diferentes dispositivos de visualización. Por defecto, flexbox organiza los niños en una columna. Pero puede cambiarlo a fila usando

`flexDirection: 'row'`.

direccion flex

```
const Direction = (props) => {
  return (
    <View style={styles.container}>
      <Box/>
      <Box/>
      <Box/>
      <View style={{flexDirection:'row'}}>
        <Box/>
        <Box/>
        <Box/>
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: '#AED581',
  }
});
```



Eje de alineación

```
const AlignmentAxis = (props) => {
  return (
    <View style={styles.container}>
      <Box />
      <View style={{flex:1, alignItems:'flex-end', justifyContent:'flex-end'}}>
        <Box />
        <Box />
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: `#69B8CC`,
  },
  text: {
    color: 'white',
    textAlign: 'center'
  }
})
```

```
});
```


AlignmentAx



Capítulo 9: Ejecutar una aplicación en el dispositivo (versión de Android)

Observaciones

Resolución de problemas:

Could not connect to development server => Haga esto: `adb reverse tcp:8081 tcp:8081` , asegúrese de que su teléfono esté conectado (dispositivos adb). Verifique también que haya un servidor local iniciado, si no se ejecuta `react-native start`

Examples

Ejecutar una aplicación en el dispositivo Android.

1. `adb devices`
 - ¿Se está mostrando su teléfono? De lo contrario, habilite el modo de desarrollador en su teléfono y conéctelo por USB.
2. `adb reverse tcp:8081 tcp:8081` :
 - Para vincular correctamente su teléfono y que React-Native lo reconozca durante la compilación. (**NOTA: Android Version 5 o superior**) .
3. `react-native run-android` :
 - Para ejecutar la aplicación en su teléfono.
4. `react-native start` :
 - Con el fin de iniciar un servidor local para el desarrollo (obligatorio). Este servidor se inicia automáticamente si utiliza la última versión de React-native.

Lea Ejecutar una aplicación en el dispositivo (versión de Android) en línea:

<https://riptutorial.com/es/react-native/topic/5135/ejecutar-una-aplicacion-en-el-dispositivo--version-de-android->

Capítulo 10: Enlace de API nativa

Introducción

La API de enlaces le permite enviar y recibir enlaces entre aplicaciones. Por ejemplo, abrir la aplicación Teléfono con el número marcado o abrir Google Maps e iniciar una navegación a un destino elegido. También puede utilizar Vinculación para que su aplicación pueda responder a los enlaces que la abren desde otras aplicaciones.

Para usar el `Linking`, primero debes importarlo desde `react-native`

```
import {Linking} from 'react-native'
```

Examples

Enlaces salientes

Para abrir un enlace llame a `openURL`.

```
Linking.openURL(url)
  .catch(err => console.error('An error occurred ', err))
```

El método preferido es verificar si alguna aplicación instalada puede manejar una URL dada de antemano.

```
Linking.canOpenURL(url)
  .then(supported => {
    if (!supported) {
      console.log('Unsupported URL: ' + url)
    } else {
      return Linking.openURL(url)
    }
  }).catch(err => console.error('An error occurred ', err))
```

Esquemas URI

Aplicación de destino	Ejemplo	Referencia
Navegador web	<code>https://stackoverflow.com</code>	
Teléfono	<code>tel:1-408-555-5555</code>	manzana
Correo	<code>mailto:email@example.com</code>	manzana
SMS	<code>sms:1-408-555-1212</code>	manzana

Aplicación de destino	Ejemplo	Referencia
Mapas de apple	http://maps.apple.com/?ll=37.484847,-122.148386	manzana
mapas de Google	geo:37.7749,-122.4194	Google
iTunes	Ver iTunes Link Maker	manzana
Facebook	fb://profile	Desbordamiento de pila
Youtube	http://www.youtube.com/v/oHg5SJYRHA0	manzana
Facetime	facetime://user@example.com	manzana
calendario de iOS	calshow:514300000 [1]	iPhoneDevWiki

[1] Abre el calendario en el número de segundos indicado desde 1. 1. 2001 (¿UTC?). Por alguna razón, esta API no está documentada por Apple.

Enlaces entrantes

Puede detectar cuando su aplicación se inicia desde una URL externa.

```
componentDidMount() {
  const url = Linking.getInitialURL()
  .then((url) => {
    if (url) {
      console.log('Initial url is: ' + url)
    }
  }).catch(err => console.error('An error occurred ', err))
}
```

Para habilitar esto en iOS [Link RCTLinking a su proyecto](#) .

Para habilitar esto en Android, [siga estos pasos](#) .

Lea [Enlace de API nativa en línea](#): <https://riptutorial.com/es/react-native/topic/9687/enlace-de-api-nativa>

Capítulo 11: Enrutamiento

Introducción

El enrutamiento o navegación permite aplicaciones entre diferentes pantallas. Es vital para una aplicación móvil, ya que proporciona un contexto para el usuario acerca de dónde se encuentra, desacopla las acciones de los usuarios entre las pantallas y se mueve entre ellas, proporciona una máquina de estados como modelo de toda la aplicación.

Examples

Componente navegador

Navigator funciona tanto para iOS como para Android.

```
import React, { Component } from 'react';
import { Text, Navigator, TouchableHighlight } from 'react-native';

export default class NavAllDay extends Component {
  render() {
    return (
      <Navigator
        initialRoute={{ title: 'Awesome Scene', index: 0 }}
        renderScene={(route, navigator) =>
          <Text>Hello {route.title}!</Text>
        }
        style={{padding: 100}}
      />
    );
  }
}
```

Las rutas a `Navigator` se proporcionan como objetos. También proporciona una función `renderScene` que representa la escena para cada objeto de ruta. `initialRoute` se utiliza para especificar la primera ruta.

Lea Enrutamiento en línea: <https://riptutorial.com/es/react-native/topic/8279/enrutamiento>

Capítulo 12: ESLint en reaccion-nativo

Introducción

Este es el tema para la explicación de las reglas de ESLint para reaccion-native.

Examples

Cómo empezar

Es altamente recomendable utilizar ESLint en su proyecto en react-native. ESLint es una herramienta para la validación de códigos que utiliza reglas específicas proporcionadas por la comunidad.

Para react-native puede usar conjuntos de reglas para javascript, reaccionar y reaccion-native.

Las reglas comunes de ESLint con motivación y explicaciones para javascript se pueden encontrar aquí: <https://github.com/eslint/eslint/tree/master/docs/rules> . Simplemente puede agregar un conjunto de reglas listo de los desarrolladores de ESLint agregando su .eslintrc.json al nodo 'extendido' 'eslint: recomendado'. ("extiende": ["eslint: recomendado"]) Puede leer más sobre la configuración de ESLint aquí: <http://eslint.org/docs/developer-guide/development-environment> . Se recomienda leer el documento completo sobre esta herramienta extremadamente útil.

A continuación, los documentos completos sobre las reglas para el complemento de reacción de ES Lint se pueden encontrar aquí: <https://github.com/yannickcr/eslint-plugin-react/tree/master/docs/rules> . Nota importante: no todas las reglas de reacción son relativas a la reacción nativa. Por ejemplo: reaccionar / mostrar-nombre y reaccionar / no-desconocido-propiedad por ejemplo. Otras reglas son 'debe tener' para cada proyecto en reaccion-native, como react / jsx-no-bind y reaccion / jsx-key.

Tenga mucho cuidado al elegir su propio conjunto de reglas.

Y finalmente, hay un complemento explícitamente para reaccion-native:

<https://github.com/intellicode/eslint-plugin-react-native> Nota: Si divide sus estilos en un archivo separado, descarte la regla reaccion-native / no-inline- Los estilos no funcionarán.

Para el correcto funcionamiento de esta herramienta en env nativo reactivo, es posible que necesite establecer un valor o 'env' en su configuración a esto: "env": {"browser": true, "es6": true, "amd": true} ,

ESLint es una herramienta clave para el desarrollo de productos de alta calidad.

Lea ESLint en reaccion-nativo en línea: <https://riptutorial.com/es/react-native/topic/10650/eslint-en-reaccion-nativo>

Capítulo 13: Estado

Sintaxis

- `void setState (function | object nextState, [función callback])`

Examples

setState

Para cambiar la vista en su aplicación, puede usar `setState`, esto volverá a representar su componente y cualquiera de sus componentes secundarios. `setState` realiza una fusión superficial entre el estado nuevo y el anterior, y desencadena una nueva representación del componente.

`setState` toma un objeto clave-valor o una función que devuelve un objeto clave-valor

Objeto clave-valor

```
this.setState({myKey: 'myValue'});
```

Función

El uso de una función es útil para actualizar un valor basado en el estado o las propiedades existentes.

```
this.setState((previousState, currentProps) => {
  return {
    myInteger: previousState.myInteger+1
  }
})
```

También puede pasar una devolución de llamada opcional a `setState` que se activará cuando el componente se haya procesado nuevamente con el nuevo estado.

```
this.setState({myKey: 'myValue'}, () => {
  // Component has re-rendered... do something amazing!
});
```

Ejemplo completo

```
import React, { Component } from 'react';
import { AppRegistry, StyleSheet, Text, View, TouchableOpacity } from 'react-native';

export default class MyParentComponent extends Component {
  constructor(props) {
    super(props);
  }
}
```

```

    this.state = {
      myInteger: 0
    }
  }
  getRandomInteger() {
    const randomInt = Math.floor(Math.random()*100);

    this.setState({
      myInteger: randomInt
    });
  }
  incrementInteger() {

    this.setState((previousState, currentProps) => {
      return {
        myInteger: previousState.myInteger+1
      }
    });
  }
  render() {

    return <View style={styles.container}>

      <Text>Parent Component Integer: {this.state.myInteger}</Text>

      <MyChildComponent myInteger={this.state.myInteger} />

      <Button label="Get Random Integer" onPress={this.getRandomInteger.bind(this)} />
      <Button label="Increment Integer" onPress={this.incrementInteger.bind(this)} />

    </View>

  }
}

export default class MyChildComponent extends Component {
  constructor(props) {
    super(props);
  }
  render() {

    // this will get updated when "MyParentComponent" state changes
    return <View>
      <Text>Child Component Integer: {this.props.myInteger}</Text>
    </View>

  }
}

export default class Button extends Component {
  constructor(props) {
    super(props);
  }
  render() {

    return <TouchableOpacity onPress={this.props.onPress}>
      <View style={styles.button}>

```



```

        <Text style={styles.buttonText}>{this.props.label}</Text>
      </View>
    </TouchableOpacity>

  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  button: {
    backgroundColor: '#444',
    padding: 10,
    marginTop: 10
  },
  buttonText: {
    color: '#fff'
  }
});

AppRegistry.registerComponent('MyApp', () => MyParentComponent);

```

Inicializar estado

Debería inicializar el estado dentro de la función constructora de su componente de esta manera:

```

export default class MyComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      myInteger: 0
    }
  }
  render() {
    return (
      <View>
        <Text>Integer: {this.state.myInteger}</Text>
      </View>
    )
  }
}

```

Usando `setState` uno puede actualizar la vista.

Lea Estado en línea: <https://riptutorial.com/es/react-native/topic/3596/estado>

Capítulo 14: Estilo

Introducción

Los estilos se definen dentro de un objeto JSON con nombres de atributos de estilo similares como en CSS. Dicho objeto se puede poner en línea en el estilo prop de un componente o se puede pasar a la función `StyleSheet.create(StyleObject)` y se puede almacenar en una variable para un acceso en línea más corto utilizando un nombre de selector para él similar a una clase en CSS.

Sintaxis

- `<Component style={styleFromStyleSheet} />`
- `<Component style={styleObject} />`
- `<Component style={[style1, style2]} />`

Observaciones

La mayoría de los estilos nativos de React son sus formularios CSS, pero en camel case. Así, el `text-decoration` convierte en `textDecoration`.

A diferencia de CSS, los estilos no se heredan. Si desea que los componentes secundarios hereden un cierto estilo, debe proporcionárselos explícitamente al niño. Esto significa que no puede establecer una familia de fuentes para una `View` completa.

La única excepción a esto es el componente `Text`: los `Text` anidados heredan sus estilos principales.

Examples

Estilismo usando estilos en línea

Cada componente React Native puede tener un `style` prop. Puedes pasarle un objeto JavaScript con propiedades de estilo CSS:

```
<Text style={{color:'red'}}>Red text</Text>
```

Esto puede ser ineficiente ya que tiene que recrear el objeto cada vez que se renderiza el componente. Se prefiere usar una hoja de estilo.

Estilo usando una hoja de estilo

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

const styles = StyleSheet.create({
```

```

    red: {
      color: 'red'
    },
    big: {
      fontSize: 30
    }
  });

class Example extends Component {
  render() {
    return (
      <View>
        <Text style={styles.red}>Red</Text>
        <Text style={styles.big}>Big</Text>
      </View>
    );
  }
}

```

`StyleSheet.create()` devuelve un objeto donde los valores son números. React Native sabe convertir estos ID numéricos en el objeto de estilo correcto.

Añadiendo múltiples estilos

Puede pasar una matriz a la propiedad de `style` para aplicar varios estilos. Cuando hay un conflicto, el último de la lista tiene prioridad.

```

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

const styles = StyleSheet.create({
  red: {
    color: 'red'
  },
  greenUnderline: {
    color: 'green',
    textDecoration: 'underline'
  },
  big: {
    fontSize: 30
  }
});

class Example extends Component {
  render() {
    return (
      <View>
        <Text style={[styles.red, styles.big]}>Big red</Text>
        <Text style={[styles.red, styles.greenUnderline]}>Green underline</Text>
        <Text style={[styles.greenUnderline, styles.red]}>Red underline</Text>
        <Text style={[styles.greenUnderline, styles.red, styles.big]}>Big red
underline</Text>
        <Text style={[styles.big, {color:'yellow'}]}>Big yellow</Text>
      </View>
    );
  }
}

```

Estilo condicional

```
<View style={[this.props.isTrue ? styles.backgroundColorBlack : styles.backgroundColorWhite]}>
```

Si el valor de `isTrue` es `true`, tendrá un color de fondo negro; de lo contrario, será blanco.

Lea Estilo en línea: <https://riptutorial.com/es/react-native/topic/7757/estilo>

Capítulo 15: Examen de la unidad

Introducción

La prueba de unidad es una práctica de prueba de bajo nivel donde se prueban las unidades o componentes más pequeños del código.

Examples

Pruebas unitarias con broma.

Jest es un marco de pruebas de javascript ampliamente utilizado para probar aplicaciones de reacción. Es soportado por facebook.

Aquí hay una prueba

```
import 'react-native';
import React from 'react';
import Index from '../index.android.js';

import renderer from 'react-test-renderer';

it('renders correctly', () => {
  const tree = renderer.create(
    <Index />
  );
});
```

Aquí hay un código para hacerlo pasar

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

export default class gol extends Component {
  render() {
    return (
      <View>
        <Text>
          Welcome to React Native!
        </Text>
        <Text>
          To get started, edit index.android.js
        </Text>
        <Text>
          Double tap R on your keyboard to reload,{'\n'}
          Shake or press menu button for dev menu
        </Text>
      </View>
    );
  }
}
```

```
        </View>
      );
    }
  }

AppRegistry.registerComponent('gol', () => gol);
```

Prueba de unidad en reaccionar nativo usando bromia

A partir de la versión reactiva-nativa 0.38, se incluye una configuración de Jest de forma predeterminada cuando se ejecuta react-native init. La siguiente configuración se debe agregar automáticamente a su archivo package.json:

```
"scripts": {
  "start": "node node_modules/react-native/local-cli/cli.js start",
  "test": "jest"
},
"jest": {
  "preset": "react-native"
}
```

Puede ejecutar `run npm test` or `jest` para probar en reaccionar nativo. Por ejemplo de código:

[Enlace](#)

Lea Examen de la unidad en línea: <https://riptutorial.com/es/react-native/topic/8281/examen-de-la-unidad>

Capítulo 16: Fuentes personalizadas

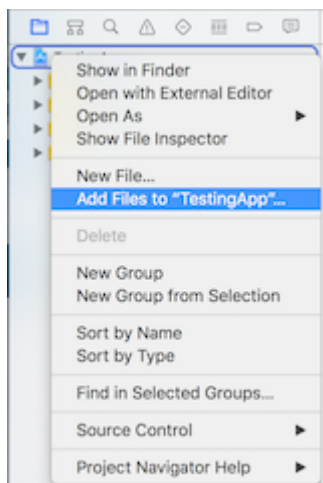
Examples

Pasos para usar fuentes personalizadas en React Native (Android)

1. Pegue su archivo de fuentes en `android/app/src/main/assets/fonts/font_name.ttf`
2. Recompila la aplicación de Android ejecutando `react-native run-android`
3. Ahora, puedes usar `fontFamily: 'font_name'` en tus React Native Styles

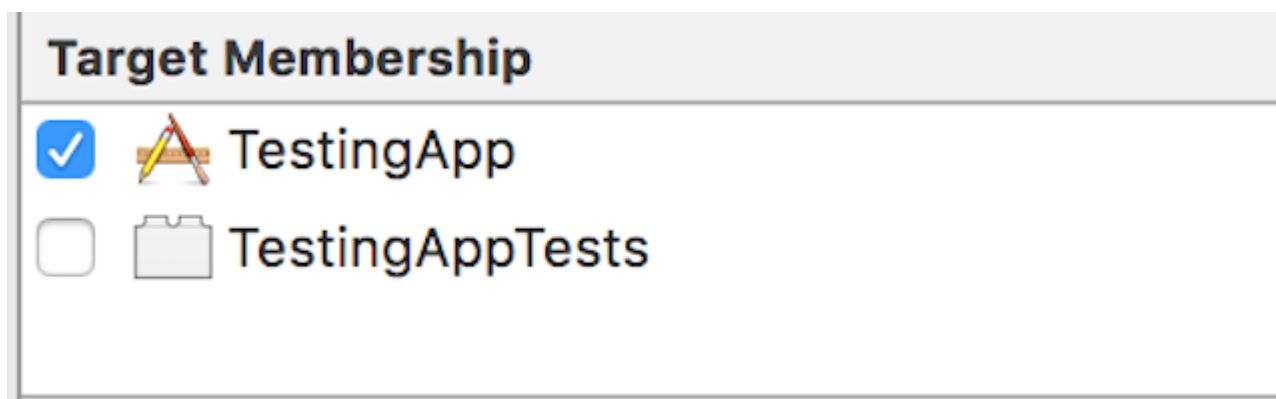
Pasos para usar fuentes personalizadas en React Native (iOS)

1. Incluye la fuente en tu proyecto Xcode.



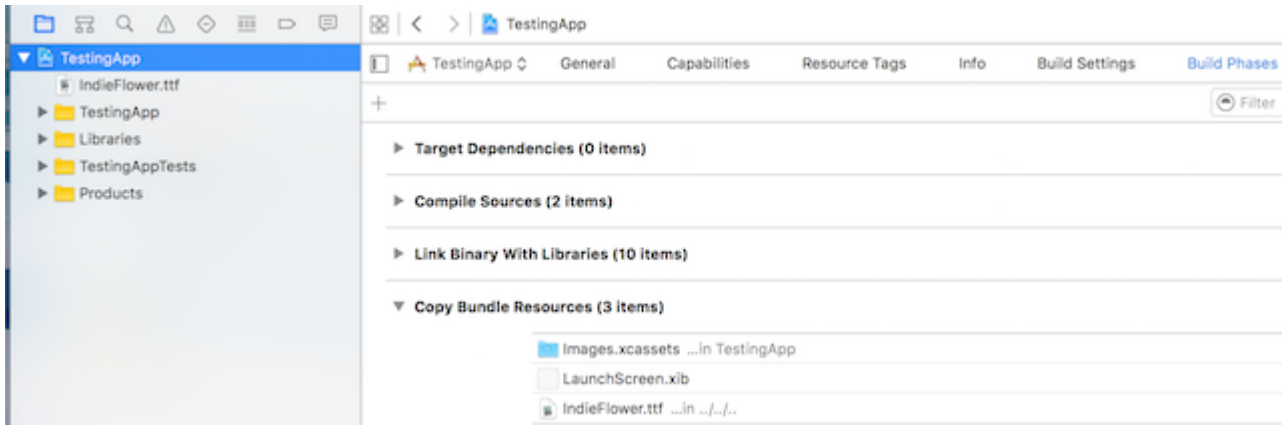
2. Asegúrese de que estén incluidos en la columna de membresía de destino

Haga clic en la fuente del navegador y compruebe si la fuente está incluida.



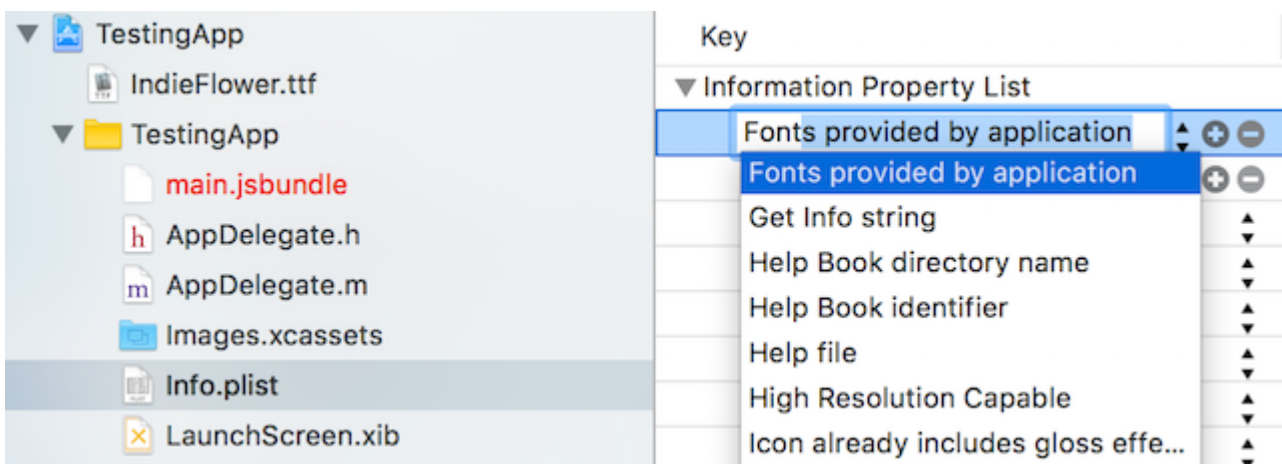
3. Compruebe si la fuente incluida como recurso en su paquete

haga clic en el archivo de proyecto de Xcode, seleccione "Crear fases, seleccione" Copiar recursos del paquete ". Compruebe si se ha agregado la fuente.



4. Incluir la fuente en Application Plist (Info.plist)

desde la carpeta principal de la aplicación, abra Info.plist, haga clic en "Lista de propiedades de información" y luego haga clic en el signo más (+). De la lista desplegable, elija "Fuentes proporcionadas por la aplicación".



5. Añadir el nombre de la fuente en las fuentes proporcionadas por la aplicación

expanda las fuentes proporcionadas por la aplicación y agregue el nombre de la fuente exactamente a la columna de valor

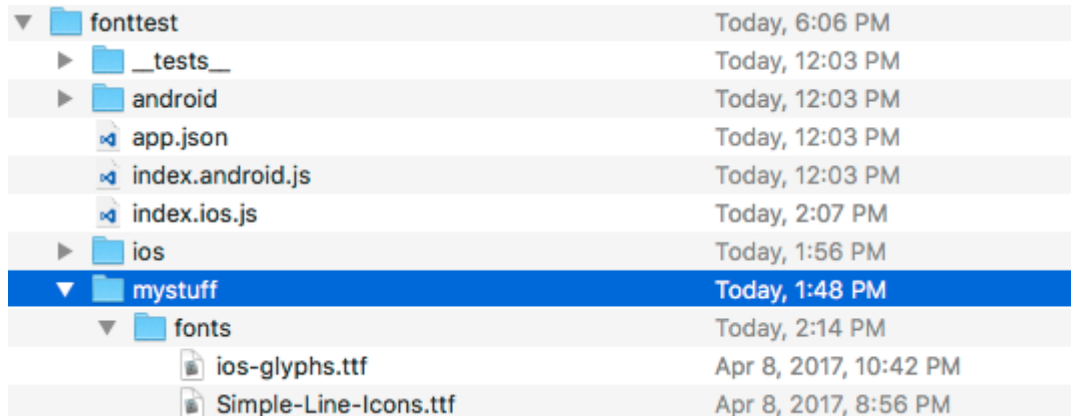
Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
▼ Fonts provided by application	Array	(1 item)
Item 0	String	IndieFlower.ttf

6. Utilízalo en la aplicación.

```
<Text style={{fontFamily:'IndieFlower'}}>
  Welcome to React Native!
</Text>
```

Fuentes personalizadas tanto para Android como para iOS

- Cree una carpeta en la carpeta de su proyecto y agregue sus fuentes a ella. Ejemplo:
 - Ejemplo: Aquí agregamos una carpeta en la raíz llamada "mystuff", luego "fuentes", y dentro de ella colocamos nuestras fuentes:



- Agregue el siguiente código en `package.json`.

```
{
  ...

  "rnpm": {
    "assets": [
      "path/to/fontfolder"
    ]
  },
  ...
}
```

- Para el ejemplo anterior, nuestro `package.json` ahora tendría una ruta de "mystuff / fonts":

```
"rnpm": {
  "assets": [
    "mystuff/fonts"
  ]
}
```

- Ejecutar el comando de `react-native link`.
- Usando fuentes personalizadas en el proyecto debajo del código

```
<Text style={{ fontFamily: 'FONT-NAME' }}>
  My Text
</Text>
```

Donde `FONT-NAME` es el prefijo específico de la plataforma.

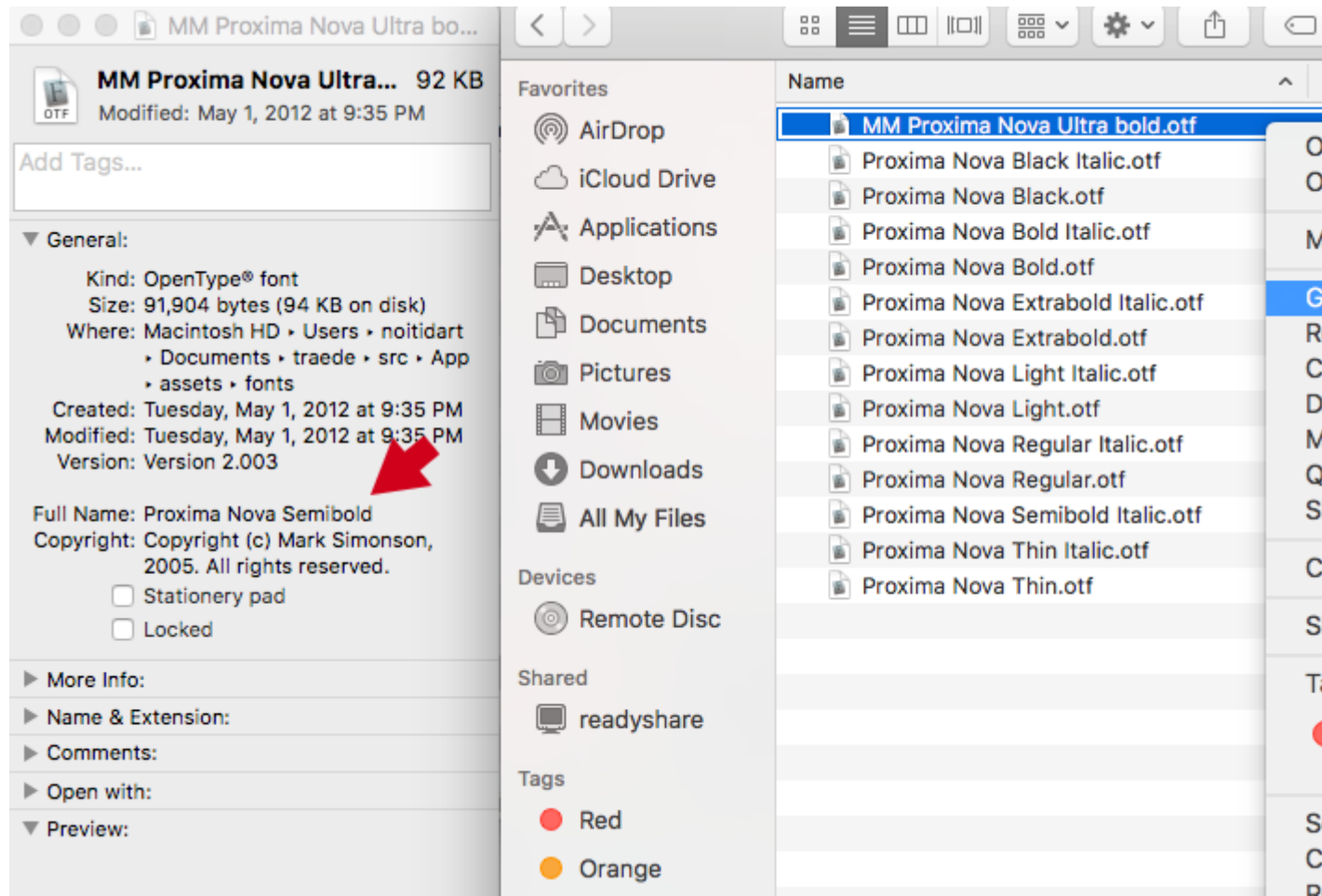
Androide

`FONT-NAME` son las palabras antes de la extensión en el archivo. Ejemplo: el nombre de

archivo de su fuente es `Roboto-Regular.ttf` , por lo que debe establecer `fontFamily: Roboto-Regular` .

iOS

FONT-NAME es el "Nombre completo" que se encuentra después de hacer clic derecho, en el archivo de fuente, y luego hacer clic en "Obtener información". (Fuente: <https://stackoverflow.com/a/16788493/2529614>), en la captura de pantalla siguiente, el nombre del archivo es `MM Proxima Nova Ultra bold.otf` , sin embargo "Nombre completo" es "Proxima Nova Semibold", por lo que set `fontFamily: Proxima Nova Semibold` . Captura de pantalla -



- Ejecute `react-native run-ios` o `react-native run-android` nuevamente (esto se volverá a compilar con los recursos)

Lea Fuentes personalizadas en línea: <https://riptutorial.com/es/react-native/topic/4341/fuentes-personalizadas>

Capítulo 17: Hola Mundo

Examples

Editando index.ios.js o index.android.js

Abra `index.ios.js` o `index.android.js` y elimine todo lo que haya entre `<View>` `</View>` . Después de eso, escribe `<Text> Hello World! </Text>` y ejecuta el emulador.

Deberías ver `Hello World!` Escrito en la pantalla!

Felicidades ¡Has escrito con éxito tu primer Hello World!

¡Hola Mundo!

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

class HelloWorldApp extends Component {
  render() {
    return (
      <Text>Hello world!</Text>
    );
  }
}

AppRegistry.registerComponent('HelloWorldApp', () => HelloWorldApp);
```

Lea **Hola Mundo en línea**: <https://riptutorial.com/es/react-native/topic/3779/hola-mundo>

Capítulo 18: Imágenes

Examples

Módulo de imagen

Vas a tener que importar la `Image` del paquete `react-native`, así que úsala:

```
import { Image } from 'react';

<Image source={{uri: 'https://image-souce.com/awesomeImage'}} />
```

También puede usar una imagen local con una sintaxis ligeramente diferente pero con la misma lógica, como por ejemplo:

```
import { Image } from 'react';

<Image source={require('./img/myCoolImage.png')} />
```

Nota: - Debes dar altura, ancho a la imagen, de lo contrario no se mostrará.

Ejemplo de imagen

```
class ImageExample extends Component {
  render() {
    return (
      <View>
        <Image style={{width: 30, height: 30}}
          source={{uri: 'http://facebook.github.io/react/img/logo_og.png'}}
        />
      </View>
    );
  }
}
```

Fuente de imagen condicional

```
<Image style={ [this.props.imageStyle] }
  source={this.props.imagePath
    ? this.props.imagePath
    : require('../theme/images/resource.png')}
 />
```

Si la ruta está disponible en `imagePath`, se asignará a la fuente; de lo contrario, se asignará la ruta de la imagen predeterminada.

Usando variable para ruta de imagen

```
let imagePath = require(".././assets/list.png");  
  
<Image style={{height: 50, width: 50}} source={imagePath} />
```

Del recurso externo:

```
<Image style={{height: 50, width: 50}} source={{uri: userData.image}} />
```

Para encajar una imagen

```
<Image  
  resizeMode="contain"  
  style={{height: 100, width: 100}}  
  source={require('../assets/image.png')} />
```

Intenta también **cubrir** , **estirar** , **repetir** y **centrar los** parámetros.

Lea Imágenes en línea: <https://riptutorial.com/es/react-native/topic/3956/imagenes>

Capítulo 19: Instrucciones de línea de comando

Examples

Comprobar versión instalada

```
$ react-native -v
```

Ejemplo de salida

```
react-native-cli: 0.2.0
react-native: n/a - not inside a React Native project directory //Output from different folder
react-native: react-native: 0.30.0 // Output from the react native project directory
```

Actualizar el proyecto existente a la última versión de RN

En la carpeta de la aplicación, busque `package.json` y modifique la siguiente línea para incluir la última versión, guarde el archivo y ciérrelo.

```
"react-native": "0.32.0"
```

En la terminal:

```
$ npm install
```

Seguido por

```
$ react-native upgrade
```

Explotación florestal

Androide

```
$ react-native log-android
```

iOS

```
$ react-native log-ios
```

Inicializa y comienza con el proyecto React Native.

Inicializar

```
react-native init MyAwesomeProject
```

Inicializar con una versión específica de React Native.

```
react-native init --version="0.36.0" MyAwesomeProject
```

Para ejecutar para Android

```
cd MyAwesomeProject  
react-native run-android
```

Para ejecutar para iOS

```
cd MyAwesomeProject  
react-native run-ios
```

Iniciar React Packing nativo

```
$ react-native start
```

En la última versión de React Native, no es necesario ejecutar el empaquetador. Se ejecutará automáticamente.

Por defecto, esto inicia el servidor en el puerto 8081. Para especificar en qué puerto está el servidor

```
$ react-native start --port PORTNUMBER
```

Añadir proyecto de Android para su aplicación

Si tienes aplicaciones generadas con soporte pre-android o simplemente lo hiciste a propósito, siempre puedes agregar proyectos de Android a tu aplicación.

```
$ react-native android
```

Esto generará la carpeta de `android` e `index.android.js` dentro de tu aplicación.

Lea Instrucciones de línea de comando en línea: <https://riptutorial.com/es/react-native/topic/2117/instrucciones-de-linea-de-comando>

Capítulo 20: Integración con Firebase para la autenticación

Introducción

// Reemplace los valores de base de fuego con los valores api de su aplicación importe base de fuego desde 'base de fuego';

```
componentWillMount () {firebase.initializeApp ({apiKey: "yourAPIKey", authDomain: "authDomainName", databaseURL: "yourDomainBaseURL", projectId: your firebase.auth ().signInWithEmailAndPassword (correo electrónico, contraseña) .then (this.onLoginSuccess)}}}
```

Examples

Reaccionar nativo - ListView con Firebase

Esto es lo que hago cuando trabajo con Firebase y quiero usar ListView.

Utilice un componente principal para recuperar los datos de Firebase (Posts.js):

Posts.js

```
import PostsList from './PostsList';

class Posts extends Component{
  constructor(props) {
    super(props);
    this.state = {
      posts: []
    }
  }

  componentWillMount() {
    firebase.database().ref('Posts/').on('value', function(data) {
      this.setState({ posts: data.val() });
    });
  }

  render() {
    return <PostsList posts={this.state.posts}/>
  }
}
```

PostsList.js

```
class PostsList extends Component {
  constructor(props) {
    super(props);
    this.state = {
```



```

        dataSource: new ListView.DataSource({
            rowHasChanged: (row1, row2) => row1 !== row2
        }),
    },
}

getDataSource(posts: Array<any>): ListView.DataSource {
    if(!posts) return;
    return this.state.dataSource.cloneWithRows(posts);
}

componentDidMount() {
    this.setState({dataSource: this.getDataSource(this.props.posts)});
}

componentWillReceiveProps(props) {
    this.setState({dataSource: this.getDataSource(props.posts)});
}

renderRow = (post) => {
    return (
        <View>
            <Text>{post.title}</Text>
            <Text>{post.content}</Text>
        </View>
    );
}

render() {
    return(
        <ListView
            dataSource={this.state.dataSource}
            renderRow={this.renderRow}
            enableEmptySections={true}
        />
    );
}
}

```

Quiero señalar que en `Posts.js`, no estoy importando `firebase` porque solo necesita importarlo una vez, en el componente principal de su proyecto (donde tiene el navegador) y usarlo en cualquier lugar.

Esta es la solución que alguien sugirió en una pregunta que hice cuando estaba luchando con ListView. Pensé que sería bueno compartirlo.

Fuente: [<http://stackoverflow.com/questions/38414289/react-native-listview-not-rendering-data-from-firebase> ♦ []

Autenticación en React Native usando Firebase

Reemplace los valores de base de fuego con los valores api de su aplicación:

```

import firebase from 'firebase';
componentWillMount() {
    firebase.initializeApp({
        apiKey: "yourAPIKey",

```

```
    authDomain: "authDomainName",
    databaseURL: "yourDomainBaseURL",
    projectId: "yourProjectID",
    storageBucket: "storageBUcketValue",
    messagingSenderId: "senderIdValue"
  });
  firebase.auth().signInWithEmailAndPassword(email, password)
    .then(this.onLoginSuccess)
    .catch(() => {
      firebase.auth().createUserWithEmailAndPassword(email, password)
        .then(this.onLoginSuccess)
        .catch(this.onLoginFail)
    })
}
```

Lea Integración con Firebase para la autenticación en línea: <https://riptutorial.com/es/react-native/topic/6391/integracion-con-firebase-para-la-autenticacion>

Capítulo 21: Mejores Prácticas de Navegador

Examples

Navegador

`Navigator` es el navegador predeterminado de React Native. Un componente `Navigator` administra una *pila* de objetos de ruta y proporciona métodos para administrar esa pila.

```
<Navigator
  ref={(navigator) => { this.navigator = navigator }}
  initialRoute={{ id: 'route1', title: 'Route 1' }}
  renderScene={this.renderScene.bind(this)}
  configureScene={(route) => Navigator.SceneConfigs.FloatFromRight}
  style={{ flex: 1 }}
  navigationBar={
    // see "Managing the Navigation Bar" below
    <Navigator.NavigationBar routeMapper={this.routeMapper} />
  }
/>
```

Gestión de la pila de ruta

En primer lugar, observe el prop `initialRoute`. Una ruta es simplemente un objeto javascript y puede tomar la forma que desee y tener los valores que desee. Es la forma principal de pasar valores y métodos entre los componentes de la pila de navegación.

El `Navigator` sabe qué representar en función del valor devuelto por su prop `renderScene`.

```
renderScene(route, navigator) {
  if (route.id === 'route1') {
    return <ExampleScene navigator={navigator} title={route.title} />; // see below
  } else if (route.id === 'route2') {
    return <ExampleScene navigator={navigator} title={route.title} />; // see below
  }
}
```

Imaginemos una implementación de `ExampleScene` en este ejemplo:

```
function ExampleScene(props) {

  function forward() {
    // this route object will passed along to our `renderScene` function we defined above.
    props.navigator.push({ id: 'route2', title: 'Route 2' });
  }

  function back() {
    // `pop` simply pops one route object off the `Navigator`'s stack
    props.navigator.pop();
  }

  return (
```

```

<View>
  <Text>{props.title}</Text>
  <TouchableOpacity onPress={forward}>
    <Text>Go forward!</Text>
  </TouchableOpacity>
  <TouchableOpacity onPress={back}>
    <Text>Go Back!</Text>
  </TouchableOpacity>
</View>
);
}

```

Configurando el navegador

Puede configurar las transiciones del `Navigator` con la opción `configureScene` prop. Esta es una función que pasa el objeto de `route` y necesita devolver un objeto de configuración. Estos son los objetos de configuración disponibles:

- `Navigator.SceneConfigs.PushFromRight` (predeterminado)
- `Navigator.SceneConfigs.FloatFromRight`
- `Navigator.SceneConfigs.FloatFromLeft`
- `Navigator.SceneConfigs.FloatFromBottom`
- `Navigator.SceneConfigs.FloatFromBottomAndroid`
- `Navigator.SceneConfigs.FadeAndroid`
- `Navigator.SceneConfigs.HorizontalSwipeJump`
- `Navigator.SceneConfigs.HorizontalSwipeJumpFromRight`
- `Navigator.SceneConfigs.VerticalUpSwipeJump`
- `Navigator.SceneConfigs.VerticalDownSwipeJump`

Puede devolver uno de estos objetos sin modificación, o puede modificar el objeto de configuración para personalizar las transiciones de navegación. Por ejemplo, para modificar la anchura hit borde para emular más de cerca el iOS `UINavigationController`'s

`interactivePopGestureRecognizer` :

```

configureScene=(route) => {
  return {
    ...Navigator.SceneConfigs.FloatFromRight,
    gestures: {
      pop: {
        ...Navigator.SceneConfigs.FloatFromRight.gestures.pop,
        edgeHitWidth: Dimensions.get('window').width / 2,
      },
    },
  };
}

```

Manejando la barra de navegación

El componente `Navigator` viene con un prop de barra de `navigationBar`, que teóricamente puede tomar cualquier componente React configurado correctamente. Pero la implementación más común utiliza el `Navigator.NavigationBar` predeterminado. Esto requiere un prop de `routeMapper` que puede usar para configurar el aspecto de la barra de navegación según la ruta.

Un `routeMapper` es un objeto javascript normal con tres funciones: `Title` , `RightButton` y `LeftButton` .
Por ejemplo:

```
const routeMapper = {

  LeftButton(route, navigator, index, navState) {
    if (index === 0) {
      return null;
    }

    return (
      <TouchableOpacity
        onPress={() => navigator.pop()}
        style={styles.navBarLeftButton}
      >
        <Text>Back</Text>
      </TouchableOpacity>
    );
  },

  RightButton(route, navigator, index, navState) {
    return (
      <TouchableOpacity
        onPress={route.handleRightButtonClick}
        style={styles.navBarRightButton}
      >
        <Text>Next</Text>
      </TouchableOpacity>
    );
  },

  Title(route, navigator, index, navState) {
    return (
      <Text>
        {route.title}
      </Text>
    );
  },
};
```

Ver más

Para obtener una documentación más detallada de cada propuesta, consulte la [documentación oficial de React Native para el Navigator](#) y la guía de React Native sobre el [uso de los navegadores](#) .

Utilice react-navigation para navegar en reaccionar aplicaciones nativas

Con la ayuda de [react-navigation](#) , puede agregar la navegación a su aplicación realmente fácil.

Instalar react-navigation

```
npm install --save react-navigation
```

Ejemplo:

```

import { Button, View, Text, AppRegistry } from 'react-native';
import { StackNavigator } from 'react-navigation';

const App = StackNavigator({
  FirstPage: {screen: FirstPage},
  SecondPage: {screen: SecondPage},
});

class FirstPage extends React.Component {
  static navigationOptions = {
    title: 'Welcome',
  };
  render() {
    const { navigate } = this.props.navigation;

    return (
      <Button
        title='Go to Second Page'
        onPress={() =>
          navigate('SecondPage', { name: 'Awesomepankaj' })
        }
      />
    );
  }
}

class SecondPage extends React.Component {
  static navigationOptions = ({navigation}) => ({
    title: navigation.state.params.name,
  });

  render() {
    const { goBack } = this.props.navigation;
    return (
      <View>
        <Text>Welcome to Second Page</Text>
        <Button
          title="Go back to First Page"
          onPress={() => goBack()}
        />
      </View>
    );
  }
}

```

navegación nativa de reacción con flujo de enrutador nativo de reacción

Instale usando `npm install --save react-native-router-flux`

En react-native-router-flux, cada ruta se denomina `<Scene>`

```
<Scene key="home" component={LogIn} title="Home" initial />
```

`key` Una cadena única que se puede usar para referirse a la escena en particular.

`component` que componente mostrar, aquí está

`title` crea una barra de navegación y dale un título 'Inicio'

initial ¿Es esta la primera pantalla de la aplicación?

Ejemplo:

```
import React from 'react';
import { Scene, Router } from 'react-native-router-flux';
import LogIn from './components/LogIn';
import SecondPage from './components/SecondPage';

const RouterComponent = () => {
  return (
    <Router>
      <Scene key="login" component={LogIn} title="Login Form" initial />
      <Scene key="secondPage" component={SecondPage} title="Home" />
    </Router>
  );
};

export default RouterComponent;
```

Importe este archivo en el App.js principal (archivo de índice) y renderícelo. Para más información puede visitar este [enlace](#) .

Lea Mejores Prácticas de Navegador en línea: <https://riptutorial.com/es/react-native/topic/2559/mejores-practicas-de-navegador>

Capítulo 22: Mejores Prácticas de Render

Introducción

Tema para notas importantes sobre el comportamiento específico del método `Component.render`.

Examples

Funciones en JSX

Para un mejor rendimiento es importante evitar el uso de la función array (lambda) en JSX.

Como se explica en <https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md> :

Una llamada de enlace o una función de flecha en un prop JSX creará una nueva función en cada render. Esto es malo para el rendimiento, ya que hará que el recolector de basura sea invocado más de lo necesario. También puede causar repeticiones innecesarias si se pasa una nueva función como apoyo a un componente que utiliza la verificación de igualdad de referencia en el elemento para determinar si debería actualizarse.

Así que si tienes bloque de código jsx como este:

```
<TextInput
  onChangeValue={ value => this.handleValueChanging(value) }
/>
```

o

```
<button onClick={ this.handleClick.bind(this) }></button>
```

puedes mejorarlo

```
<TextInput
  onChangeValue={ this.handleValueChanging }
/>
```

y

```
<button onClick={ this.handleClick }></button>
```

Para el contexto correcto dentro de la función `handleValueChanging`, puede aplicarlo en el constructor del componente:

```
constructor() {
```



```
    this.handleChangeing = this.handleChangeing.bind(this)
  }
```

más en [vincular una función pasada a un componente](#)

O puede usar soluciones como esta: <https://github.com/andreypopp/autobind-decorator> y simplemente agregue @autobind decorator a cada method que desee enlazar a:

```
@autobind
handleChangeing(newValue)
{
  //processing event
}
```

Lea **Mejores Prácticas de Render en línea**: <https://riptutorial.com/es/react-native/topic/10649/mejores-practicas-de-render>

Capítulo 23: Modal

Introducción

El componente modal es una forma sencilla de presentar contenido por encima de una vista adjunta.

Parámetros

Apuntalar	detalles
tipo de animación	es una enumeración de (' none ', ' slide ', ' fade ') y controla la animación modal.
visible	Es un bool que controla la visibilidad modal.
en el programa	permite pasar una función que se llamará una vez que se muestre el modal.
transparente	bool para establecer la transparencia.
onRequestClose (Android)	Siempre se define un método que se llamará cuando el usuario haga clic en el botón Atrás
onOrientationChange (IOS)	Siempre definiendo un método que será llamado cuando cambie la orientación.
Orientaciones soportadas (IOS)	enum ('portrait', 'portrait-upside-down', 'landscape', 'landscape-left', 'landscape-right')

Examples

Ejemplo Modal Básico

```
import React, { Component } from 'react';
import {
  Modal,
  Text,
  View,
  Button,
  StyleSheet,
} from 'react-native';

const styles = StyleSheet.create({
  mainContainer: {
    marginTop: 22,
  },
});
```

```

    modalContainer: {
      marginTop: 22,
    },
  });

class Example extends Component {
  constructor() {
    super();
    this.state = {
      visibility: false,
    };
  }

  setModalVisibility(visible) {
    this.setState({
      visibility: visible,
    });
  }

  render() {
    return (
      <View style={styles.mainContainer}>
        <Modal
          animationType={'slide'}
          transparent={false}
          visible={this.state.visibility}
        >
          <View style={styles.modalContainer}>
            <View>
              <Text>I'm a simple Modal</Text>
              <Button
                color="#000"
                onPress={() => this.setModalVisibility(!this.state.visibility)}
                title="Hide Modal"
              />
            </View>
          </View>
        </Modal>

        <Button
          color="#000"
          onPress={() => this.setModalVisibility(true)}
          title="Show Modal"
        />
      </View>
    );
  }
}

export default Example;

```

Ejemplo Modal Transparente

Veamos este ejemplo [aquí](#) .

```

import React, { Component } from 'react';
import { Text, View, StyleSheet, Button, Modal } from 'react-native';
import { Constants } from 'expo';

```

```

export default class App extends Component {
  state = {
    modalVisible: false,
  };

  _handleButtonPress = () => {
    this.setModalVisible(true);
  };

  setModalVisible = (visible) => {
    this.setState({modalVisible: visible});
  }

  render() {
    var modalBackgroundStyle = {
      backgroundColor: 'rgba(0, 0, 0, 0.5)'
    };
    var innerContainerTransparentStyle = {backgroundColor: '#fff', padding: 20};
    return (
      <View style={styles.container}>
        <Modal
          animationType='fade'
          transparent={true}
          visible={this.state.modalVisible}
          onRequestClose={() => this.setModalVisible(false)}
        >
          <View style={[styles.container, modalBackgroundStyle]}>
            <View style={innerContainerTransparentStyle}>
              <Text>This is a modal</Text>
              <Button title='close'
                onPress={this.setModalVisible.bind(this, false)}>
            </View>
          </View>
        </Modal>
        <Button
          title="Press me"
          onPress={this._handleButtonPress}
        />

      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#ecf0f1',
  }
});

```

Lea Modal en línea: <https://riptutorial.com/es/react-native/topic/8253/modal>

Capítulo 24: Módulo de plataforma

Examples

Encuentra el tipo de sistema operativo / versión

El primer paso es importar la plataforma desde el paquete 'react-native' así:

```
import { Platform } from 'react-native'
```

Después de hacer eso, puede seguir adelante y acceder al tipo de sistema operativo a través de `Platform.OS` lo que le permite usarlo en sentencias condicionales como

```
const styles = StyleSheet.create({
  height: (Platform.OS === 'ios') ? 200 : 100,
})
```

Si desea detectar la versión de Android, puede usar `Platform.Version` así:

```
if (Platform.Version === 21) {
  console.log('Running on Lollipop!');
}
```

Para iOS, `Platform.Version` está devolviendo un String, por una condición compleja no olvide analizarlo.

```
if (parseInt(Platform.Version, 10) >= 9) {
  console.log('Running version higher than 8');
}
```

Si la lógica específica de la plataforma es compleja, se pueden representar dos archivos diferentes según la plataforma. Ex:

- `MyTask.android.js`
- `MyTask.ios.js`

y lo requieren usando

```
const MyTask = require('./MyTask')
```

Lea Módulo de plataforma en línea: <https://riptutorial.com/es/react-native/topic/3593/modulo-de-plataforma>

Capítulo 25: Módulos nativos

Examples

Crea tu módulo nativo (IOS)

Introducción

de <http://facebook.github.io/react-native/docs/native-modules-ios.html>

A veces, una aplicación necesita acceso a la API de la plataforma, y React Native todavía no tiene un módulo correspondiente. Tal vez desee reutilizar algún código existente de Objective-C, Swift o C++ sin tener que volverlo a implementar en JavaScript, o escribir algún código de alto rendimiento y multiproceso, como para el procesamiento de imágenes, una base de datos o cualquier número de extensiones avanzadas.

Un módulo nativo es simplemente una clase Objective-C que implementa el protocolo

`RCTBridgeModule`.

Ejemplo

En su proyecto Xcode cree un nuevo archivo y seleccione **Cocoa Touch Class**, en el asistente de creación elija un nombre para su clase (*por ejemplo, NativeModule*), conviértalo en una **subclase de** `NSObject` y elija `Objective-C` para el idioma.

Esto creará dos archivos `NativeModuleEx.h` y `NativeModuleEx.m`

Deberá importar `RCTBridgeModule.h` a su archivo `NativeModuleEx.h` siguiente manera:

```
#import <Foundation/Foundation.h>
#import "RCTBridgeModule.h"

@interface NativeModuleEx : NSObject <RCTBridgeModule>

@end
```

En su `NativeModuleEx.m` agregue el siguiente código:

```
#import "NativeModuleEx.h"

@implementation NativeModuleEx

RCT_EXPORT_MODULE();
```

```
RCT_EXPORT_METHOD(testModule:(NSString *)string )
{
  NSLog(@"The string '%@' comes from JavaScript! ", string);
}

@end
```

`RCT_EXPORT_MODULE()` hará que su módulo sea accesible en JavaScript, puede pasarle un argumento opcional para especificar su nombre. Si no se proporciona ningún nombre, coincidirá con el nombre de clase de Objective-C.

`RCT_EXPORT_METHOD()` expondrá su método a JavaScript, solo los métodos que exporte utilizando esta macro serán accesibles en JavaScript.

Finalmente, en tu JavaScript puedes llamar a tu método como sigue:

```
import { NativeModules } from 'react-native';

var NativeModuleEx = NativeModules.NativeModuleEx;

NativeModuleEx.testModule('Some String !');
```

Lea Módulos nativos en línea: <https://riptutorial.com/es/react-native/topic/6155/modulos-nativos>

Capítulo 26: Navegador con botones inyectados desde páginas.

Examples

Introducción

En lugar de hinchar su archivo js principal que contiene su navegador con botones. Es más limpio simplemente inyectar botones a pedido en cualquier página que necesite.

```
//In the page "Home", I want to have the right nav button to show
//a settings modal that resides in "Home" component.

componentWillMount() {
  this.props.route.navbarTitle = "Home";

  this.props.route.rightNavButton = {
    text: "Settings",
    onPress: this._ShowSettingsModal.bind(this)
  };
}
```

Ejemplo completo comentado

```
'use strict';

import React, {Component} from 'react';
import ReactNative from 'react-native';

const {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Navigator,
  Alert,
  TouchableHighlight
} = ReactNative;

//This is the app container that contains the navigator stuff
class AppContainer extends Component {

  renderScene(route, navigator) {
    switch(route.name) {
      case "Home":
        //You must pass route as a prop for this trick to work properly
        return <Home route={route} navigator={navigator} {...route.passProps} />
      default:
        return (
          <Text route={route}
            style={styles.container}>

```



```

        Your route name is probably incorrect {JSON.stringify(route)}
    </Text>
    );
  }
}

render() {
  return (
    <Navigator
      navigationBar={
        <Navigator.NavigationBar
          style={ styles.navbar }
          routeMapper={ NavigationBarRouteMapper } />
        </Navigator.NavigationBar>
      }
      initialRoute={{ name: 'Home' }}
      renderScene={ this.renderScene }
    />
  );
}

//Nothing fancy here, except for checking for injected buttons.
//Notice how we are checking if there are injected buttons inside the route object.
//Also, we are showing a "Back" button when the page is not at index-0 (e.g. not home)
var NavigationBarRouteMapper = {
  LeftButton(route, navigator, index, navState) {
    if(route.leftNavButton) {
      return (
        <TouchableHighlight
          style={styles.leftNavButton}
          underlayColor="transparent"
          onPress={route.leftNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.leftNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
    else if(route.enableBackButton) {
      return (
        <TouchableHighlight
          style={styles.leftNavButton}
          underlayColor="transparent"
          onPress={() => navigator.pop() }>
          <Text style={styles.navbarButtonText}>Back</Text>
        </TouchableHighlight>
      );
    }
  },
  RightButton(route, navigator, index, navState) {
    if(route.rightNavButton) {
      return (
        <TouchableHighlight
          style={styles.rightNavButton}
          underlayColor="transparent"
          onPress={route.rightNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.rightNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
  }
}

```

```

    },
    Title(route, navigator, index, navState) {
      //You can inject the title aswell.  If you don't we'll use the route name.
      return (<Text style={styles.navbarTitle}>{route.navbarTitle || route.name}</Text>);
    }
  };

//This is considered a sub-page that navigator is showing
class Home extends Component {

  //This trick depends on that componentWillMount fires before the navbar is created
  componentWillMount() {
    this.props.route.navbarTitle = "Home";

    this.props.route.rightNavButton = {
      text: "Button",
      onPress: this._doSomething.bind(this)
    };
  }

  //This method will be invoked by pressing the injected button.
  _doSomething() {
    Alert.alert(
      'Awesome, eh?',
      null,
      [
        {text: 'Indeed'},
      ]
    )
  }

  render() {
    return (
      <View style={styles.container}>
        <Text>You are home</Text>
      </View>
    );
  }
}

var styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
    marginTop: 66
  },
  navbar: {
    backgroundColor: '#ffffff',
  },
  navbarTitle: {
    marginVertical: 10,
    fontSize: 17
  },
  leftNavButton: {
    marginVertical: 10,
    paddingLeft: 8,
  },
  rightNavButton: {
    marginVertical: 10,
  }
});

```

```
paddingRight: 8,  
},  
navbarButtonText: {  
  fontSize: 17,  
  color: "#007AFF"  
}  
});  
  
AppRegistry.registerComponent('AppContainer', () => AppContainer);
```

Lea Navegador con botones inyectados desde páginas. en línea: <https://riptutorial.com/es/react-native/topic/6416/navegador-con-botones-inyectados-desde-paginas->

Capítulo 27: Notificación de inserción

Introducción

Podemos agregar notificaciones push para reaccionar a la aplicación nativa usando el módulo npm [react-native-push-notification](#) de [zo0r](#) . Esto permite un desarrollo multiplataforma.

Instalación

```
npm install --save reaccion-native-push-notification
```

enlace nativo reactivo

Observaciones

Consulte [GitHub Repo](#) de este módulo para más detalles.

Examples

Configuración simple de notificaciones push

Crear nuevo proyecto PushNotification

```
react-native init PushNotification
```

Poner siguiente en index.android.js

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

import {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Button
} from 'react-native';

import PushNotification from 'react-native-push-notification';

export default class App extends Component {

  constructor(props) {
    super(props);

    this.NewNotification = this.NewNotification.bind(this);
  }

  componentDidMount () {

    PushNotification.configure({
```

```

    // (required) Called when a remote or local notification is opened or received
    onNotification: function(notification) {
        console.log( 'NOTIFICATION:', notification );
    },

    // Should the initial notification be popped automatically
    // default: true
    popInitialNotification: true,

    /**
     * (optional) default: true
     * - Specified if permissions (ios) and token (android and ios) will requested or
not,
     * - if not, you must call PushNotificationsHandler.requestPermissions() later
     */
    requestPermissions: true,
  });
}

NewNotification(){

    let date = new Date(Date.now() + (this.state.seconds * 1000));

    //Fix for IOS
    if(Platform.OS == "ios"){
        date = date.toISOString();
    }

    PushNotification.localNotificationSchedule({
        message: "My Notification Message", // (required)
        date: date, // (optional) for setting delay
        largeIcon:"" // set this blank for removing large icon
        //smallIcon: "ic_notification", // (optional) default: "ic_notification" with
fallback for "ic_launcher"
    });
}

render() {

    return (
        <View style={styles.container}>
            <Text style={styles.welcome}>
                Push Notification
            </Text>
            <View style={styles.Button} >
                <Button
                    onPress={()=>{this.NewNotification()}}
                    title="Show Notification"
                    style={styles.Button}
                    color="#841584"
                    accessibilityLabel="Show Notification"
                />
            </View>
        </View>
    );
}

}

const styles = StyleSheet.create({

```

```

container: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: '#F5FCFF',
},
welcome: {
  fontSize: 20,
  textAlign: 'center',
  margin: 10,
},
Button:{
  margin: 10,
}
});

AppRegistry.registerComponent('PushNotification', () => App);

```

Navegando a la escena desde Notificación

Este es un ejemplo simple para demostrar que, ¿cómo podemos saltar / abrir una pantalla específica en función de la notificación? Por ejemplo, cuando un usuario hace clic en la notificación, la aplicación debería abrirse y pasar directamente a la página de notificaciones en lugar de a la página de inicio.

```

'use strict';

import React, { Component } from 'react';
import {
  StyleSheet,
  Text,
  View,
  Navigator,
  TouchableOpacity,
  AsyncStorage,
  BackAndroid,
  Platform,
} from 'react-native';
import PushNotification from 'react-native-push-notification';

let initialRoute = { id: 'loginview' }

export default class MainClass extends Component
{
  constructor(props)
  {
    super(props);

    this.handleNotification = this.handleNotification.bind(this);
  }

  handleNotification(notification)
  {
    console.log('handleNotification');
    var notificationId = ''
    //your logic to get relevant information from the notification

    //here you navigate to a scene in your app based on the notification info
  }
}

```

```

    this.navigator.push({ id: Constants.ITEM_VIEW_ID, item: item });
}

componentDidMount()
{
    var that = this;

    PushNotification.configure({

        // (optional) Called when Token is generated (iOS and Android)
        onRegister: function(token) {
            console.log( 'TOKEN:', token );
        },

        // (required) Called when a remote or local notification is opened or received
        onNotification(notification) {
            console.log('onNotification')
            console.log( notification );

            that.handleNotification(notification);
        },

        // ANDROID ONLY: (optional) GCM Sender ID.
        senderID: "Vizido",

        // IOS ONLY (optional): default: all - Permissions to register.
        permissions: {
            alert: true,
            badge: true,
            sound: true
        },

        // Should the initial notification be popped automatically
        // default: true
        popInitialNotification: true,

        /**
         * (optional) default: true
         * - Specified if permissions (ios) and token (android and ios) will requested or
not,
         * - if not, you must call PushNotificationsHandler.requestPermissions() later
         */
        requestPermissions: true,
    });
}

render()
{
    return (
        <Navigator
            ref={(nav) => this.navigator = nav }
            initialRoute={initialRoute}
            renderScene={this.renderScene.bind(this)}
            configureScene={(route) =>
                {
                    if (route.sceneConfig)
                    {
                        return route.sceneConfig;
                    }
                    return Navigator.SceneConfigs.FadeAndroid;
                }
            }
        >

```

```
        }
    }
    />
  );
}

renderScene(route, navigator)
{
  switch (route.id)
  {
    // do your routing here
    case 'mainview':
      return ( <MainView navigator={navigator} /> );

    default:
      return ( <MainView navigator={navigator} /> );
  }
}
}
```

Lea Notificación de inserción en línea: <https://riptutorial.com/es/react-native/topic/9674/notificacion-de-insercion>

Capítulo 28: RefreshControl con ListView

Observaciones

Referencias:

RefreshControl: <https://facebook.github.io/react-native/docs/refreshcontrol.html>

ListView: <https://facebook.github.io/react-native/docs/listview.html>

Examples

Control de actualización

```
_refreshControl() {
  return (
    <RefreshControl
      refreshing={this.state.refreshing}
      onRefresh={()=>this._refreshListView()} />
  )
}
```

refreshante: es el estado del spinner (verdadero, falso).

onRefresh: esta función invocará cuando actualice ListView / ScrollView.

onRefresh function Ejemplo

```
_refreshListView() {
  //Start Rendering Spinner
  this.setState({refreshing:true})
  this.state.cars.push(
    {name:'Fusion',color:'Black'},
    {name:'Yaris',color:'Blue'}
  )
  //Updating the dataSource with new data
  this.setState({ dataSource:
    this.state.dataSource.cloneWithRows(this.state.cars) })
  this.setState({refreshing:false}) //Stop Rendering Spinner
}
```

Aquí estamos actualizando la matriz y después de eso actualizaremos la fuente de datos. podemos usar [fetch](#) para solicitar algo del servidor y usar `async / await`.

Refresque el control con ListView Ejemplo completo

RefreshControl se utiliza dentro de un ScrollView o ListView para agregar la funcionalidad pull para actualizar. En este ejemplo lo usaremos con ListView.

```

'use strict'
import React, { Component } from 'react';
import { StyleSheet, View, ListView, RefreshControl, Text } from 'react-native'

class RefreshControlExample extends Component {
  constructor () {
    super()
    this.state = {
      refreshing: false,
      dataSource: new ListView.DataSource({
        rowHasChanged: (row1, row2) => row1 !== row2 }),
      cars : [
        {name:'Datsun',color:'White'},
        {name:'Camry',color:'Green'}
      ]
    }
  }

  componentWillMount(){
    this.setState({ dataSource:
      this.state.dataSource.cloneWithRows(this.state.cars) })
  }

  render() {
    return (
      <View style={{flex:1}}>
        <ListView
          refreshControl={this._refreshControl()}
          dataSource={this.state.dataSource}
          renderRow={(car) => this._renderListView(car)}>
        </ListView>
      </View>
    )
  }

  _renderListView(car){
    return(
      <View style={styles.listView}>
        <Text>{car.name}</Text>
        <Text>{car.color}</Text>
      </View>
    )
  }

  _refreshControl(){
    return (
      <RefreshControl
        refreshing={this.state.refreshing}
        onRefresh={()=>this._refreshListView()} />
    )
  }

  _refreshListView(){
    //Start Rendering Spinner
    this.setState({refreshing:true})
    this.state.cars.push(
      {name:'Fusion',color:'Black'},
      {name:'Yaris',color:'Blue'}
    )
    //Updating the dataSource with new data
  }
}

```

```
    this.setState({ dataSource:
      this.state.dataSource.cloneWithRows(this.state.cars) })
    this.setState({refreshing:false}) //Stop Rendering Spinner
  }
}

const styles = StyleSheet.create({

  listView: {
    flex: 1,
    backgroundColor:'#fff',
    marginTop:10,
    marginRight:10,
    marginLeft:10,
    padding:10,
    borderWidth:.5,
    borderColor:'#dddddd',
    height:70
  }
})

module.exports = RefreshControlExample
```

Lea RefreshControl con ListView en línea: <https://riptutorial.com/es/react-native/topic/6672/refreshcontrol-con-listview>

Capítulo 29: Representación de múltiples apoyos

Examples

render multiples variables

Para renderizar múltiples props o variables podemos usar `` ` ` .

```
render() {
  let firstName = 'test';
  let lastName = 'name';
  return (
    <View style={styles.container}>
      <Text>`${firstName} ${lastName}` </Text>
    </View>
  );
}
```

Salida: nombre de la prueba

Lea Representación de múltiples apoyos en línea: <https://riptutorial.com/es/react-native/topic/10781/representacion-de-multiples-apoyos>

Capítulo 30: Solicitudes HTTP

Sintaxis

- `fetch (url, opciones) [. luego (...)] [. catch (...)]`

Observaciones

- La API Fetch es la API más utilizada para las solicitudes HTTP. Es moderno, flexible y utiliza promesas.
- La API XMLHttpRequest también se usa para solicitudes HTTP y se incluye principalmente para que los desarrolladores puedan usar sus bibliotecas existentes favoritas, como [ApiSauce](#).
- La API de WebSocket se puede usar para datos "en vivo" en escenarios de tiempo real, como en aplicaciones de chat.

Examples

Websockets

```
var ws = new WebSocket('ws://host.com/path');

ws.onopen = () => {
  // connection opened

  ws.send('something'); // send a message
};

ws.onmessage = (e) => {
  // a message was received
  console.log(e.data);
};

ws.onerror = (e) => {
  // an error occurred
  console.log(e.message);
};

ws.onclose = (e) => {
  // connection closed
  console.log(e.code, e.reason);
};
```

HTTP con la API fetch

Cabe señalar que Fetch *no admite devoluciones de llamada de progreso*. Consulte: <https://github.com/github/fetch/issues/89>.

La alternativa es utilizar XMLHttpRequest <https://developer.mozilla.org/en->

```
fetch('https://mywebsite.com/mydata.json').then(json => console.log(json));

fetch('/login', {
  method: 'POST',
  body: form,
  mode: 'cors',
  cache: 'default',
}).then(session => onLogin(session), failure => console.error(failure));
```

Más detalles sobre fetch se pueden encontrar en [MDN](#)

Redes con XMLHttpRequest

```
var request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
  if (request.readyState !== 4) {
    return;
  }

  if (request.status === 200) {
    console.log('success', request.responseText);
  } else {
    console.warn('error');
  }
};

request.open('GET', 'https://mywebsite.com/endpoint/');
request.send();
```

Usando Promesas con la API de búsqueda y Redux

Redux es la biblioteca de administración de estado más común utilizada con React-Native. El siguiente ejemplo muestra cómo usar la API de obtención y el envío de cambios a su reductor de estado de aplicaciones usando redux-thunk.

```
export const fetchRecipes = (action) => {
  return (dispatch, getState) => {
    fetch('/recipes', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        recipeName,
        instructions,
        ingredients
      })
    })
    .then((res) => {
      // If response was successful parse the json and dispatch an update
      if (res.ok) {
        res.json().then((recipe) => {
          dispatch({
```

```

        type: 'UPDATE_RECIPe',
        recipe
    });
});
} else {
// response wasn't successful so dispatch an error
res.json().then((err) => {
    dispatch({
        type: 'ERROR_RECIPe',
        message: err.reason,
        status: err.status
    });
});
}
})
.catch((err) => {
// Runs if there is a general JavaScript error.
dispatch(error('There was a problem with the request.'));
});
});
};
};

```

Web Socket con Socket.io

Instalar *socket.io-client*

```
npm i socket.io-client --save
```

Módulo de importación

```
import SocketIOClient from 'socket.io-client/dist/socket.io.js'
```

Inicializa en tu constructor.

```

constructor(props) {
    super(props);
    this.socket = SocketIOClient('http://server:3000');
}

```

Ahora, para utilizar correctamente su conexión de socket, también debe enlazar sus funciones en el constructor. Supongamos que tenemos que crear una aplicación simple, que enviará un ping a un servidor a través de un socket cada 5 segundos (considere esto como ping), y luego la aplicación recibirá una respuesta del servidor. Para hacerlo, primero vamos a crear estas dos funciones:

```

_sendPing(){
//emit a dong message to socket server
socket.emit('ding');
}

_getReply(data){
//get reply from socket server, log it to console
console.log('Reply from server:' + data);
}

```

Ahora, necesitamos unir estas dos funciones en nuestro constructor:

```
constructor(props) {
  super(props);
  this.socket = SocketIOClient('http://server:3000');

  //bind the functions
  this._sendPing = this._sendPing.bind(this);
  this._getReply = this._getReply.bind(this);
}
```

Después de eso, también necesitamos vincular la función `_getReply` con el socket para recibir el mensaje del servidor socket. Para hacer esto necesitamos adjuntar nuestra función `_getReply` con el objeto socket. Agregue la siguiente línea a nuestro constructor:

```
this.socket.on('dong', this._getReply);
```

Ahora, siempre que el servidor de socket emita con el 'dong' su aplicación podrá recibirlo.

Http con axios

Configurar

Para solicitud web también puede utilizar la biblioteca de [axios](#) .

Es fácil de configurar. Para este propósito puedes crear el archivo `axios.js` por ejemplo:

```
import * as axios from 'axios';

var instance = axios.create();
instance.defaults.baseURL = serverURL;
instance.defaults.timeout = 20000;]
//...
//and other options

export { instance as default };
```

y luego usarlo en cualquier archivo que desee.

Peticiones

Para evitar el uso del patrón 'Swiss knife' para cada servicio en su backend, puede crear un archivo separado con métodos para esto dentro de la carpeta para la funcionalidad de integración:

```
import axios from '../axios';
import {
  errorHandler
} from '../common';

const UserService = {
  getCallToAction() {
    return axios.get('api/user/dosomething').then(response => response.data)
```



```
        .catch(errorHandling);
    },
}
export default UserService;
```

Pruebas

Hay una biblioteca especial para probar axios: [axios-mock-adapter](#) .

Con esta biblioteca puedes configurar para axios cualquier respuesta que desees para probarlo. También puedes configurar algunos errores especiales para tus métodos axios'es. Puede agregarlo a su archivo axios.js creado en el paso anterior:

```
import MockAdapter from 'axios-mock-adapter';

var mock = new MockAdapter(instance);
mock.onAny().reply(500);
```

por ejemplo.

Tienda Redux

A veces es necesario agregar token de autorización de encabezados, que probablemente almacene en su tienda de redux.

En este caso, necesitará otro archivo, interceptors.js con esta función:

```
export function getAuthToken(storeContainer) {
  return config => {
    let store = storeContainer.getState();
    config.headers['Authorization'] = store.user.accessToken;
    return config;
  };
}
```

A continuación en el constructor de su componente raíz puede agregar esto:

```
axios.interceptors.request.use(getAuthToken(this.state.store));
```

y luego todas sus solicitudes serán seguidas con su token de autorización.

Como puedes ver, axios es una biblioteca muy simple, configurable y útil para aplicaciones basadas en react-native.

Lea [Solicitudes HTTP en línea](https://riptutorial.com/es/react-native/topic/2375/solicitudes-http): <https://riptutorial.com/es/react-native/topic/2375/solicitudes-http>

Capítulo 31: Vista de la lista

Examples

Ejemplo simple

ListView: un componente central diseñado para la visualización eficiente de listas de desplazamiento vertical de datos cambiantes. La API mínima es crear un `ListView.DataSource`, rellenarlo con una simple matriz de blobs de datos e instanciar un componente `ListView` con esa fuente de datos y una devolución de llamada `renderRow` que toma un blob de la matriz de datos y devuelve un componente reproducible.

Ejemplo mínimo:

```
getInitialState: function() {
  var ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
  return {
    dataSource: ds.cloneWithRows(['row 1', 'row 2']),
  };
},

render: function() {
  return (
    <ListView
      dataSource={this.state.dataSource}
      renderRow={(rowData) => <Text>{rowData}</Text>}
    />
  );
},
```

ListView también admite funciones más avanzadas, que incluyen secciones con encabezados de sección adhesiva, encabezado y pie de página, devoluciones de llamadas al llegar al final de los datos disponibles (`onEndReached`) y en el conjunto de filas que están visibles en el cambio de la ventana del dispositivo (`onChangeVisibleRows`), y Varias optimizaciones de rendimiento.

Hay algunas operaciones de rendimiento diseñadas para hacer que `ListView` se desplace sin problemas mientras carga dinámicamente conjuntos de datos potencialmente muy grandes (o conceptualmente infinitos):

- Solo volver a renderizar filas modificadas: la función `rowHasChanged` proporcionada al origen de datos le dice a `ListView` si necesita volver a renderizar una fila porque los datos de origen han cambiado; consulte `ListViewDataSource` para obtener más detalles.
- Procesamiento de filas de velocidad limitada: de forma predeterminada, solo una fila se procesa por evento-bucle (personalizable con la propiedad `pageSize`). Esto divide el trabajo en partes más pequeñas para reducir la posibilidad de soltar marcos mientras se renderizan filas.

Lea Vista de la lista en línea: <https://riptutorial.com/es/react-native/topic/3112/vista-de-la-lista>

Capítulo 32: WebView

Introducción

WebView puede utilizarse para cargar páginas web externas o contenido html. Este componente está ahí por defecto.

Examples

Componente simple utilizando webview

```
import React, { Component } from 'react';
import { WebView } from 'react-native';

class MyWeb extends Component {
  render() {
    return (
      <WebView
        source={{uri: 'https://github.com/facebook/react-native'}}
        style={{marginTop: 20}}
      />
    );
  }
}
```

Lea WebView en línea: <https://riptutorial.com/es/react-native/topic/8763/webview>

Creditos

S. No	Capítulos	Contributors
1	Empezando con react-native	Adam , Community , Damien Varron , Dmitry Petukhov , Dr. Nitpick , Idan , Kaleb Portillo , Lucas Oliveira , manosim , Scimonster , Sivart , Tushar Khatiwada , xhg , Yevhen Dubinin
2	Accesorios	CallMeNorm , Chris Pena , corasan , fson , Gianfranco P. , henkimon , Hugo Dozois , Idan , Jagadish Upadhyay , Tobias Lins , Yevhen Dubinin , zhenjie ruan
3	Android - Botón Atrás de Hardware	Cássio Santos , manosim , Michael S , Pascal Le Merrer , Sriraman , Virat18
4	API de animación	Shashank Udupa , Sriraman , Tom Walters
5	Componentes	Michael Hancock , Sriraman , Tobias Lins
6	Crear una APK compatible para Android	Aditya Singh
7	Depuración	Jagadish Upadhyay , mostafiz rahman
8	Diseño	Alex Belets , gwint , Jagadish Upadhyay , Scimonster , sudo bangbang
9	Ejecutar una aplicación en el dispositivo (versión de Android)	Jagadish Upadhyay , Lwin Kyaw Myat , Mayeul
10	Enlace de API nativa	Viktor Seč
11	Enrutamiento	sudo bangbang
12	ESLint en reaccion-nativo	Alex Belets
13	Estado	Andyl , David , Jagadish Upadhyay , Tim Rijavec , Tobias Lins
14	Estilo	Jigar Shah , Martin Cup , Scimonster
15	Examen de la unidad	Ankit Sinha , sudo bangbang
16	Fuentes	Abdulaziz Alkharashi , Lwin Kyaw Myat , Noitidart , Olivia ,

	personalizadas	Sriraman
17	Hola Mundo	stereodenis , Zakaria Ridouh
18	Imágenes	Jagadish Upadhyay , Jigar Shah , Serdar Değirmenci , Zakaria Ridouh
19	Instrucciones de línea de comando	Dmitry Petukhov , epsilondelta , Idan , Jagadish Upadhyay , manosim , Mozak , Sriraman , Tim Rijavec
20	Integración con Firebase para la autenticación	Ankit Sinha , corasan
21	Mejores Prácticas de Navegador	Ankit Sinha , Michael Helvey , Pankaj Thakur
22	Mejores Prácticas de Render	Alex Belets
23	Modal	Ahmed Ali , Liron Yahdav , Tobias Lins
24	Módulo de plataforma	Florian Hämmerle , Gabriel Diez , Jagadish Upadhyay , Zakaria Ridouh
25	Módulos nativos	Andres C. Viesca
26	Navegador con botones inyectados desde páginas.	Ahmed Al Haddad
27	Notificación de inserción	shaN , Tejashwi Kalp Taru
28	RefreshControl con ListView	Abdulaziz Alkharashi
29	Representación de múltiples apoyos	Jigar Shah
30	Solicitudes HTTP	Alex Belets , Alireza Valizade , AntonB , Chris Pena , Daniel Schmidt , Dmitry Petukhov , Everettss , Jagadish Upadhyay , manosim , MauroPorrásP , respectTheCode , shaN , Tejashwi Kalp Taru , Tobias Lins
31	Vista de la lista	Kaleb Portillo
32	WebView	sudo bangbang