

# React Native



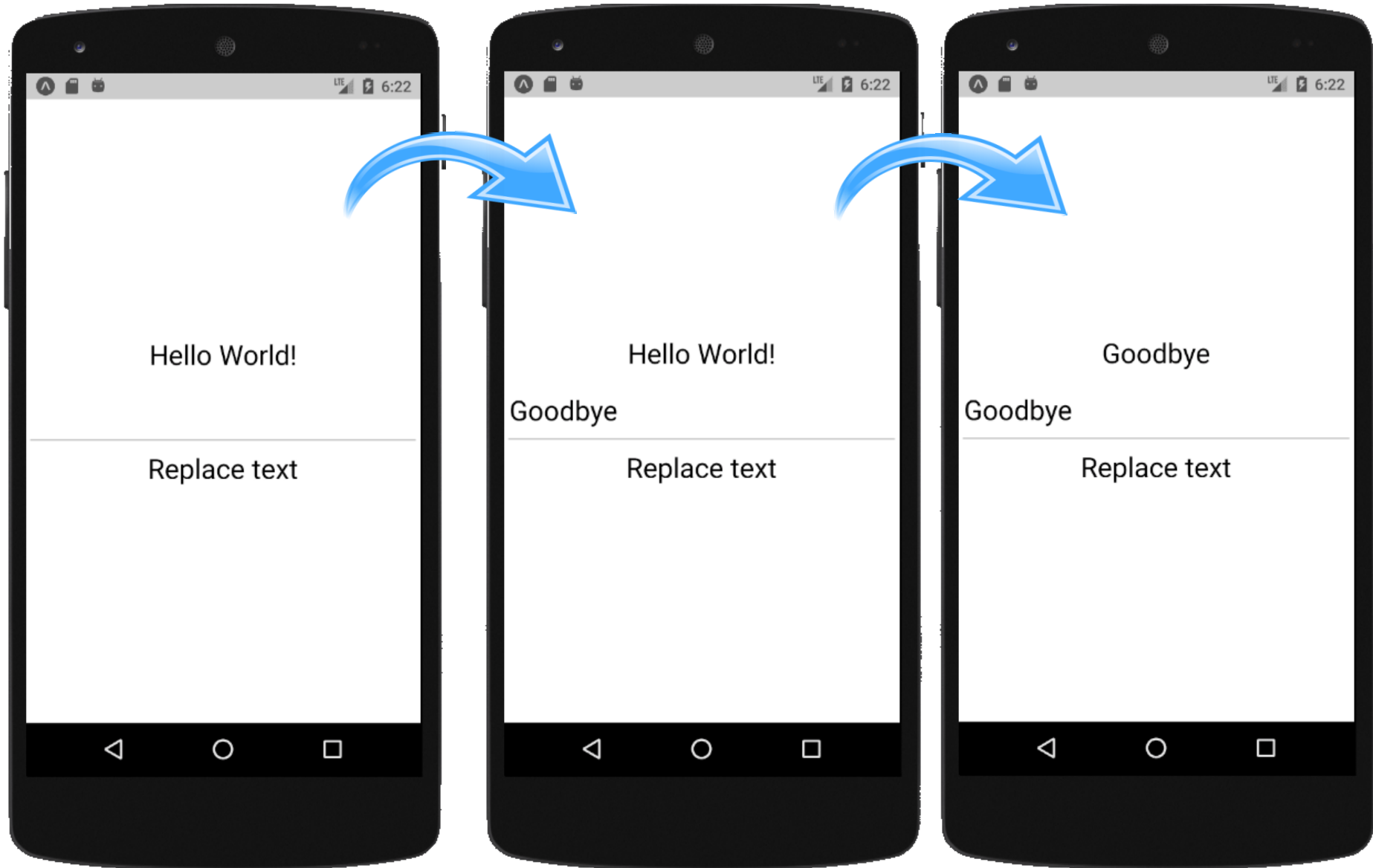
**Navegadores,  
Componentes y APIs**

# Ejercicio React Native

---

1. Instalar React Native
2. Crear y ejecutar la aplicación Hello World inicial
3. Modificar la aplicación Hello World como se indica en las transparencias 'Modificación de Hello World'
4. Realizar los siguientes cambios a la aplicación:
  - Añadir una caja de texto editable ***TextInput***
  - Añadir un botón ***TouchableHighlight*** que al pulsarlo reemplace el contenido del elemento ***Text*** que muestra inicialmente "Hello World!" por el texto escrito en el ***TextInput***
  - Posicionar todos los elementos usando **Flexbox** de modo que aparezcan dispuestos verticalmente de forma consecutiva y centrados en la pantalla (horizontal y verticalmente)

# Ejercicio React Native



# Ejercicio React Native: Solución

---

*App.js*

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

export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {valueText: "Hello World!", valueTextInput: ""}
  }
  _onTextInputChange(text){
    this.setState({valueTextInput: text});
  }
  _onPressButton(){
    this.setState({valueText: this.state.valueTextInput});
  }
}
```

# Ejercicio React Native: Solución

*App.js*

```
render() {  
  return (  
    <View style={{  
      flex:1,  
      flexDirection:'column',  
      alignItems:'center',  
      justifyContent:'center'  
    }}>  
      <Text style={{fontSize:25}}>{this.state.valueText}</Text>  
      <TextInput style={{height:70, alignSelf:'stretch', fontSize:25, padding:5}}  
        onChangeText={this._onTextInputChange.bind(this)}/>  
      <TouchableHighlight onPress={this._onPressButton.bind(this)}>  
        <Text style={{fontSize:25}}>Replace text</Text>  
      </TouchableHighlight>  
    </View>  
  )  
}
```

# React Native



## Navegadores

# Screens

---

- Son aquellos **componentes React** de una aplicación React Native que se renderizan a **pantalla completa**
- Representan las diferentes **pantallas** o páginas de una aplicación React Native
- Una aplicación está formada por una o más pantallas, y cada pantalla está formada por diferentes tipos de componentes: *Views, Texts, Images*, etc.
- Los usuarios navegarán por la aplicación transitando por las diferentes pantallas
- Los componentes encargados de la **renderización y transición de pantallas** se llaman **Navegadores**

# Screens: Hello World

---

Vamos a crear una pantalla en la aplicación Hello World

*App.js*

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

export default class App extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text style={{ fontSize: 30 }}>Hello World!</Text>
      </View>
    )
  }
}
```

# Screens: Hello World

---

Vamos a crear una pantalla en la aplicación Hello World

*App.js*

```
import React from 'react';  
// import {} from 'react-native';  
  
import IndexScreen from './index_screen';  
  
export default class App extends React.Component {  
  render() {  
    return (  
      <IndexScreen />  
    )  
  }  
}
```

# Screens: Hello World

---

Vamos a crear una pantalla en la aplicación Hello World

*index\_screen.js*

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

export default class IndexScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text style={{ fontSize: 30 }}>Hello World!</Text>
      </View>
    )
  }
}
```

# Navigators

---

- La **navegación entre pantallas (screens)** de una aplicación React Native se realiza mediante unos componentes llamados **Navegadores (Navigators)**
- Un **Navigator** es un componente encargado de la **renderización y transición de las pantallas**
- Existen varias soluciones para proporcionar navegación en una aplicación React Native:
  - **React Navigation:** solución oficial de la comunidad, implementación JavaScript  
<https://reactnavigation.org>
  - **react-native-navigation:** proporciona navegación 100% nativa en Android y iOS  
<https://github.com/wix/react-native-navigation>
  - **NavigatorIOS:** componente que proporciona un wrapper del navegador iOS (no válido para Android)

# React Navigation

---

- **Biblioteca JavaScript** desarrollada por la comunidad que ofrece una solución sencilla para proporcionar navegación en aplicaciones React Native
- Soporta 3 navegadores diferentes
  - **StackNavigator**: permite transición entre pantallas donde cada nueva pantalla se coloca encima de una pila
  - **TabNavigator**: permite crear pantallas con varias pestañas
  - **DrawerNavigator**: permite crear pantallas con navegación tipo cajón
- Instalación  
*npm install --save react-navigation*
- Documentación: <https://reactnavigation.org>

# StackNavigator

---

- Permite transición entre pantallas de modo que cada nueva pantalla se coloca encima de una **pila**
- Basado en **rutas**
  - Las **rutas** son objetos que contienen **información sobre una pantalla** (componente de React, título, estilo, ...) con el objetivo de permitir su renderización
- **Configurable**: rutas y opciones de visualización generales
  - Cada pantalla puede tener sus propias opciones de navegación y visualización
- Documentación:  
<https://reactnavigation.org/docs/navigators/stack>

# StackNavigator: Hello World

---

*App.js*

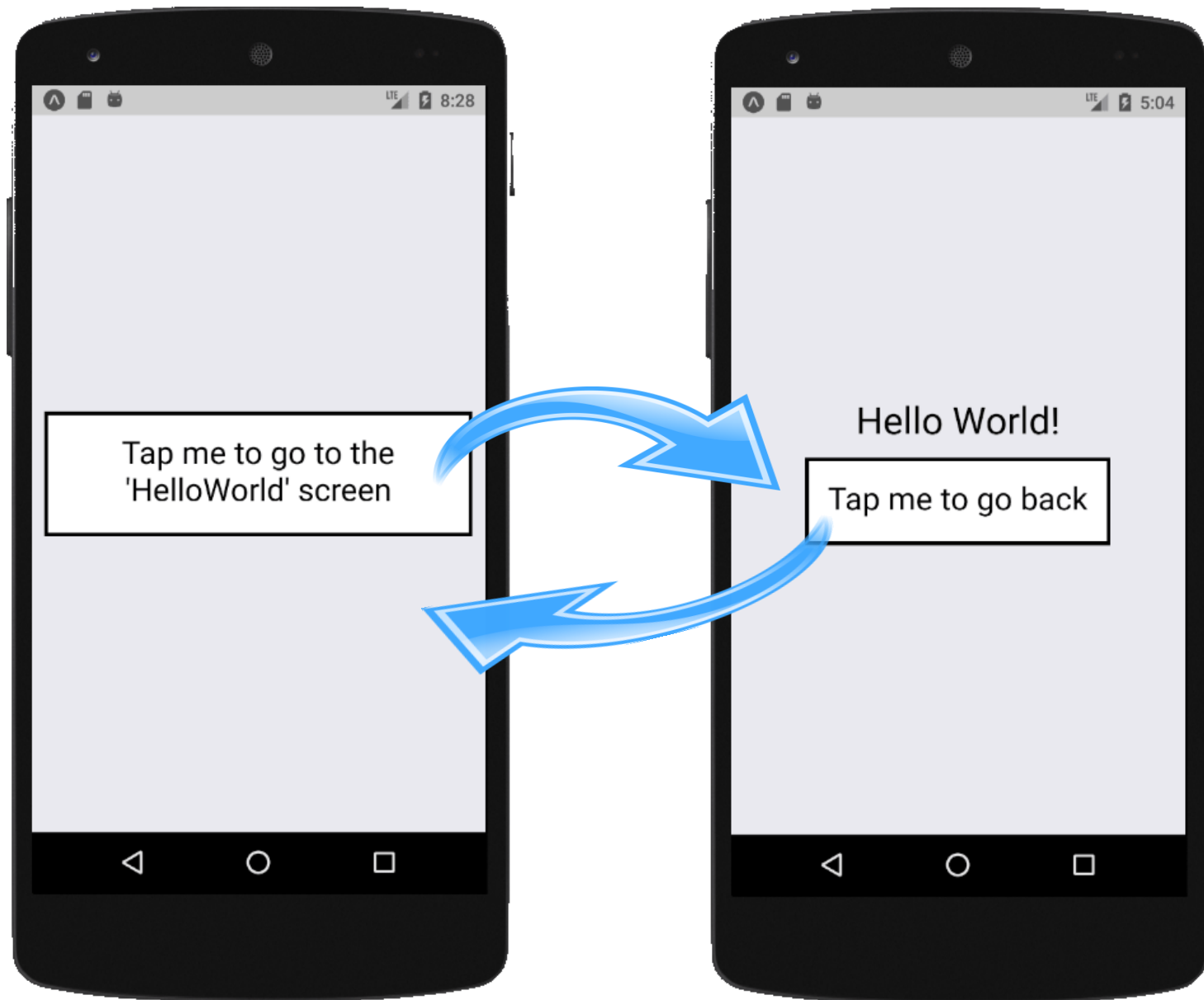
```
import React from 'react';
// import {} from 'react-native';
import { StackNavigator } from 'react-navigation';

import IndexScreen from './index_screen';

export default App = StackNavigator({
  Home: {
    screen: IndexScreen,
    navigationOptions: {
      header: null
    }
  }
})
```

# StackNavigator: Múltiples pantallas

---



# StackNavigator: Múltiples pantallas

---

*App.js*

```
import React from 'react';
import { StackNavigator } from 'react-navigation';
import IndexScreen from './index_screen';
import HelloWorldScreen from './hello_world_screen';
export default App = StackNavigator({
  Index: {
    screen: IndexScreen
  },
  HelloWorld: {
    screen: HelloWorldScreen
  }
}, {
  headerMode: 'none'
})
```

# StackNavigator: Múltiples pantallas

---

*index\_screen.js*

```
import React from 'react';
import { View } from 'react-native';
import MyButton from './my_button';

export default class IndexScreen extends React.Component {
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'center' }}>
        <MyButton
          onPress={() => this.props.navigation.navigate('HelloWorld')}
          text={"Tap me to go to the 'HelloWorld' screen"}/>
      </View>
    )
  }
}
```

# StackNavigator: Múltiples pantallas

---

*hello\_world\_screen.js*

```
import React from 'react';
import { Text, View } from 'react-native';
import MyButton from './my_button';

export default class HelloWorldScreen extends React.Component {
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'center' }}>
        <Text style={{ fontSize:30 }}>Hello World!</Text>
        <MyButton
          onPress={() => this.props.navigation.goBack(null)}
          text={"Tap me to go back"}/>
      </View>
    )
  }
}
```

# StackNavigator: Múltiples pantallas

---

*my\_button.js*

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

export default class MyButton extends React.Component {
  render() {
    return (
      <TouchableHighlight onPress={this.props.onPress}>
        <Text style={styles.text}>{this.props.text}</Text>
      </TouchableHighlight>
    )
  }
}
```

# StackNavigator: Múltiples pantallas

---

*my\_button.js*

```
const styles = StyleSheet.create({
  text: {
    padding: 15,
    margin: 10,
    backgroundColor: 'white',
    color: 'black',
    borderWidth: 3,
    borderColor: 'black',
    fontSize: 25,
    textAlign: 'center'
  }
})
```

# TabNavigator: Múltiples pantallas



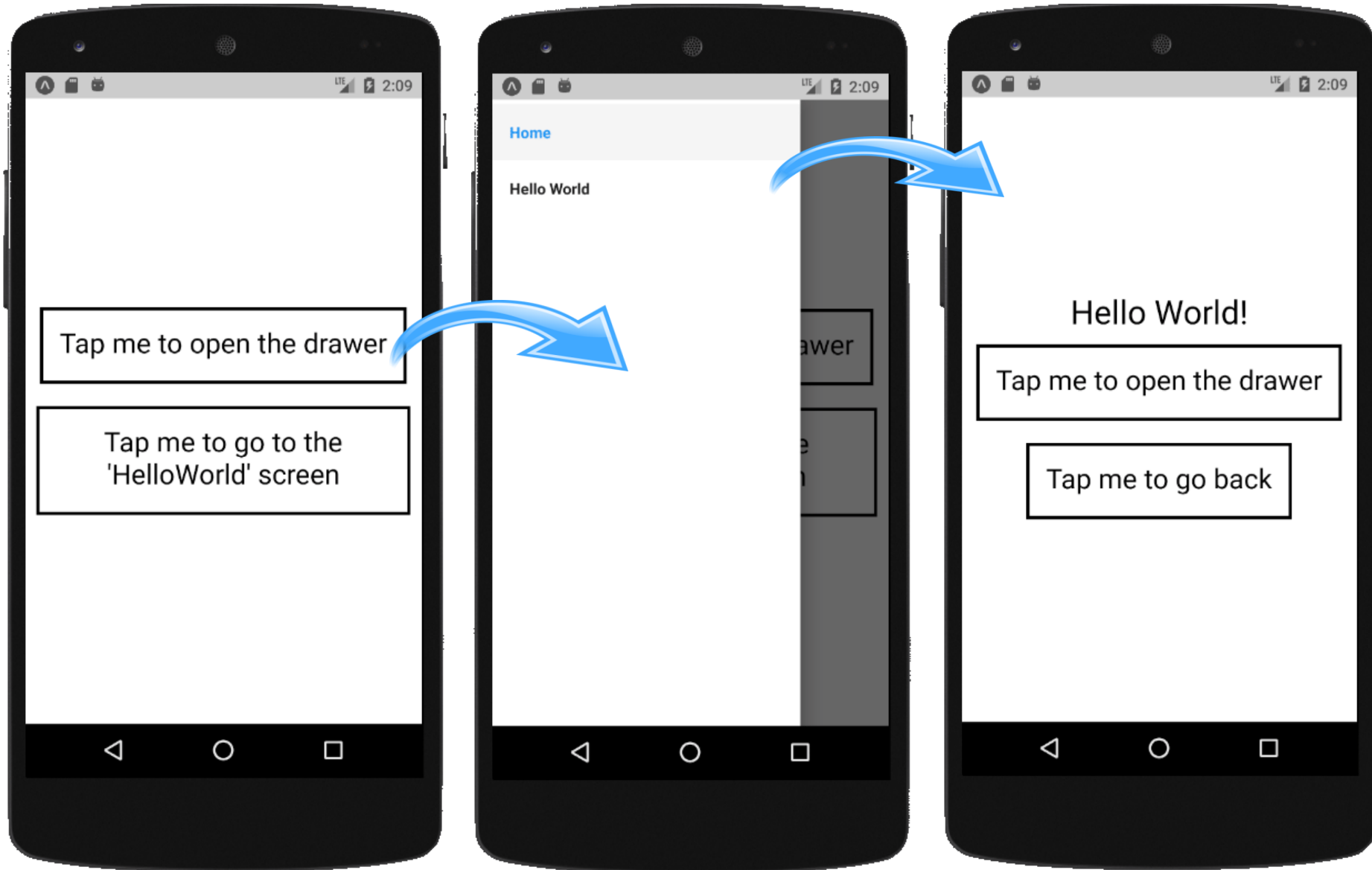
# TabNavigator: Múltiples pantallas

---

*App.js*

```
import React from 'react';
import { TabNavigator } from 'react-navigation';
import IndexScreen from './index_screen';
import HelloWorldScreen from './hello_world_screen';
export default App = TabNavigator({
  Home: {
    screen: IndexScreen,
    navigationOptions: { tabBarLabel: 'Home' }
  },
  HelloWorld: {
    screen: HelloWorldScreen,
    navigationOptions: { tabBarLabel: 'Hello World' }
  }
},{
  tabBarPosition: 'top',
  tabBarOptions: { showLabel: true, showIcon: false }
})
```

# DrawerNavigator: Múltiples pantallas



# DrawerNavigator: Múltiples pantallas

---

*App.js*

```
import React from 'react';
import { DrawerNavigator } from 'react-navigation';
import IndexScreen from './index_screen';
import HelloWorldScreen from './hello_world_screen';

export default App = DrawerNavigator({
  Home: {
    screen: IndexScreen,
    navigationOptions: { title: 'Home' }
  },
  HelloWorld: {
    screen: HelloWorldScreen,
    navigationOptions: { title: 'Hello World' }
  },
}, {
  drawerPosition: 'left'
})
```

# DrawerNavigator: Múltiples pantallas

*index\_screen.js*

```
import React from 'react';
import { View } from 'react-native';
import MyButton from './my_button';

export default class IndexScreen extends React.Component {
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'center' }}>
        <MyButton
          onPress={() => this.props.navigation.navigate('DrawerOpen')}
          text={"Tap me to open the drawer"}/>
        <MyButton
          onPress={() => this.props.navigation.navigate('HelloWorld')}
          text={"Tap me to go to the 'HelloWorld' screen"}/>
      </View>
    )
  }
}
```

# DrawerNavigator: Múltiples pantallas

*hello\_world\_screen.js*

```
import React from 'react';
import { Text, View } from 'react-native';
import MyButton from './my_button';

export default class HelloWorldScreen extends React.Component {
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'center' }}>
        <Text style={{ fontSize:30 }}>Hello World!</Text>
        <MyButton
          onPress={() => this.props.navigation.navigate('DrawerOpen')}
          text={"Tap me to open the drawer"}/>
        <MyButton
          onPress={() => this.props.navigation.goBack(null)}
          text={"Tap me to go back"}/>
      </View>
    )
  }
}
```

# React Native



## Más componentes

# Switch

---

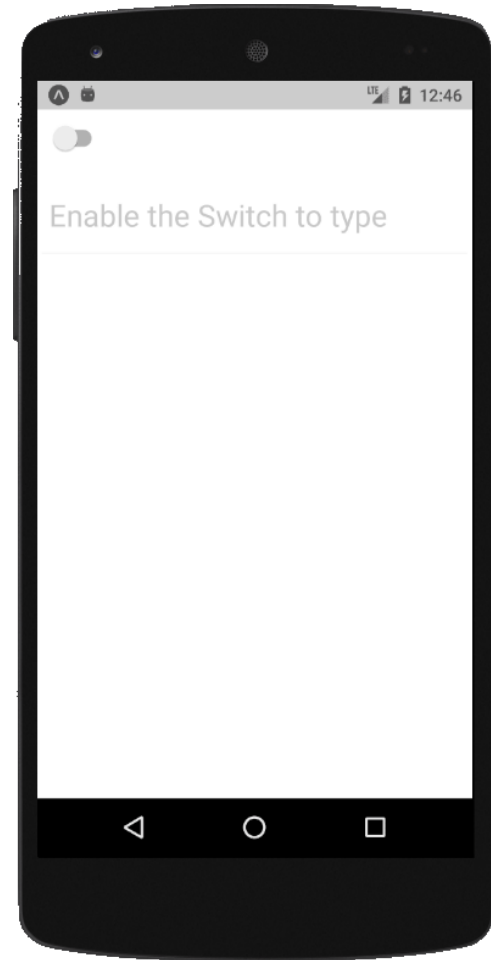
- El componente **Switch** permite incluir interruptores (*on/off*) en una aplicación
- Permite a los usuarios introducir valores **booleanos** (como los input de tipo *checkbox* en HTML)
- Es un **componente controlado\*** de React que requiere un callback **onValueChange** que actualice su propiedad **value**

\* Un componente controlado de React es un componente de entrada de datos con estado propio dependiente de la entrada de datos y cuyo valor es controlado por React

<https://reactjs.org/docs/forms.html>

- Documentación:

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



# Switch: Ejemplo

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

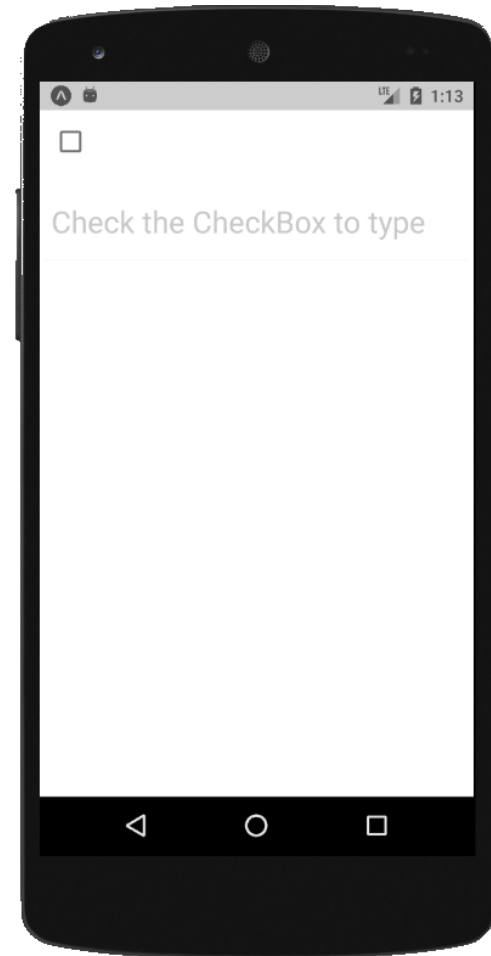
export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {switchValue: false};
  }
  _onSwitchValueChange(value){
    this.setState({switchValue: value});
  }
  render() {
    return (
      <View style={{flex:1, alignItems:'stretch'}}>
        <Switch
          value={this.state.switchValue}
          onValueChange={this._onSwitchValueChange.bind(this)}
          style={{alignSelf:'flex-start', margin: 10}} />
        <TextInput
          editable={this.state.switchValue}
          style={{height:80, padding:10, fontSize:25}}
          placeholder='Enable the Switch to type' />
      </View>
    )
  }
}
```



# CheckBox

---

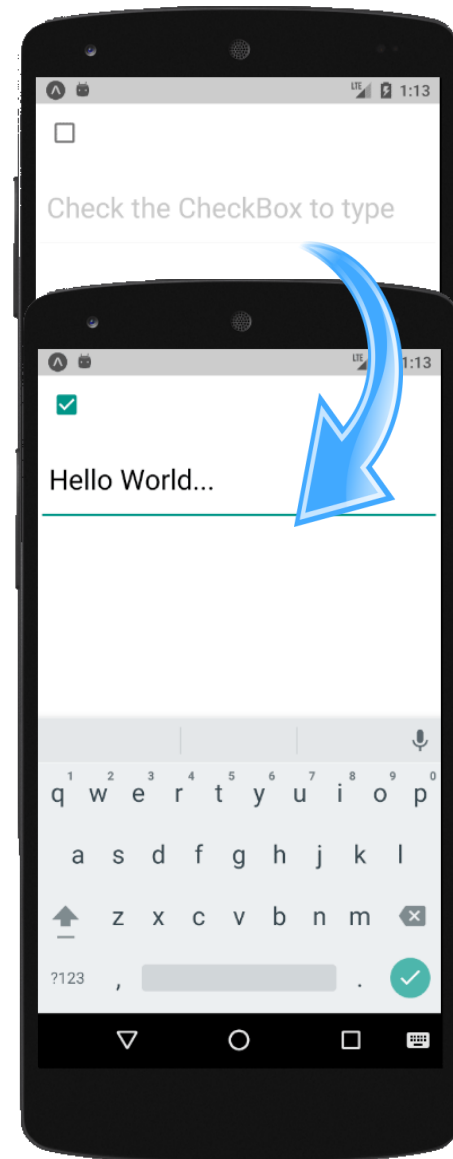
- El componente **CheckBox** permite incluir casillas en una aplicación para que los usuarios introduzcan valores **booleanos** (como los input de tipo *checkbox* en HTML)
- Muy similar a Switch pero con distinta interfaz gráfica
- Es un **componente controlado** de React que requiere un callback **onValueChange** que actualice su propiedad **value**
- Documentación:  
<https://facebook.github.io/react-native/docs/checkbox.html>



# CheckBox: Ejemplo

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

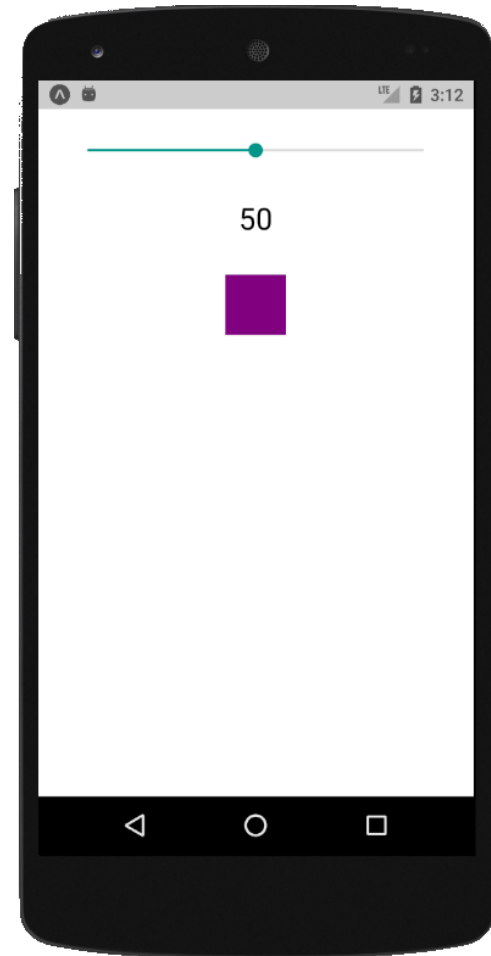
export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {checkBoxValue: false};
  }
  _onCheckBoxValueChange(value){
    this.setState({checkBoxValue: value});
  }
  render() {
    return (
      <View style={{flex:1, alignItems:'stretch'}}>
        <CheckBox
          value={this.state.checkBoxValue}
          onValueChange={this._onCheckBoxValueChange.bind(this)}
          style={{alignSelf:'flex-start', margin: 10}} />
        <TextInput
          editable={this.state.checkBoxValue}
          style={{height:80, padding:10, fontSize:25}}
          placeholder='Check the CheckBox to type' />
      </View>
    )
  }
}
```



# Slider

---

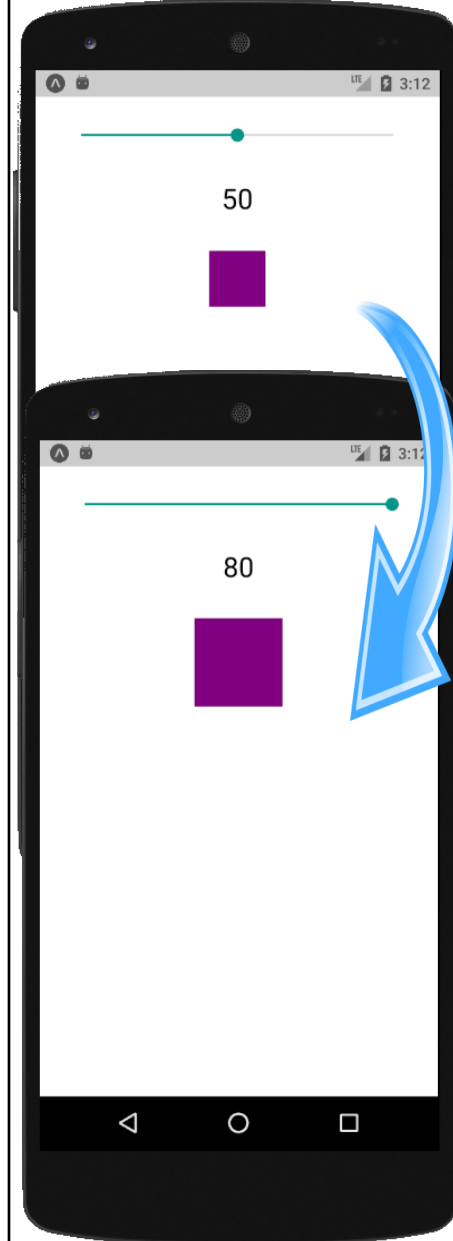
- El componente **Slider** permite a los usuarios seleccionar un único valor de una lista de **valores numéricos**
- Tiene una propiedad **value** y dos callbacks: **onValueChange** (llamado de forma continua mientras el usuario arrastra el slider) y **onSlidingComplete** (llamado cuando el usuario suelta el slider)
- Documentación:  
<https://facebook.github.io/react-native/docs/slider.html>



# Slider: Ejemplo

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

export default class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {sliderValue: 50, squareSide: 50};
  }
  _onSliderValueChange(value){this.setState({sliderValue: value});}
  _onSlidingComplete(value){this.setState({squareSide: this.state.sliderValue});}
  render() {
    return (
      <View style={{flex:1, alignItems:'center', justifyContent:'flex-start',
        padding: 25}}>
        <Slider style={{alignSelf:'stretch'}}
          step={1} minimumValue={20} maximumValue={80}
          value={this.state.sliderValue}
          onValueChange={this._onSliderValueChange.bind(this)}
          onSlidingComplete={this._onSlidingComplete.bind(this)}
        />
        <Text style={{fontSize: 25, padding: 30}}>{this.state.sliderValue}</Text>
        <View style={{width:this.state.squareSide, height:this.state.squareSide,
          backgroundColor:'purple'}}/>
      </View>
    )
  }
}
```



# Modal

---

- El componente **Modal** permite mostrar de una forma sencilla contenido sobre una vista o pantalla
- No sustituye a los **navegadores**
- Se pueden especificar varias **propiedades** (animación para mostrar el modal, estilo, ...) y **callbacks** para cuando el modal se muestre (*onShow*) o para cuando el usuario pulse el botón de retroceso (*onRequestClose*)
- Documentación:  
<https://facebook.github.io/react-native/docs/modal.html>

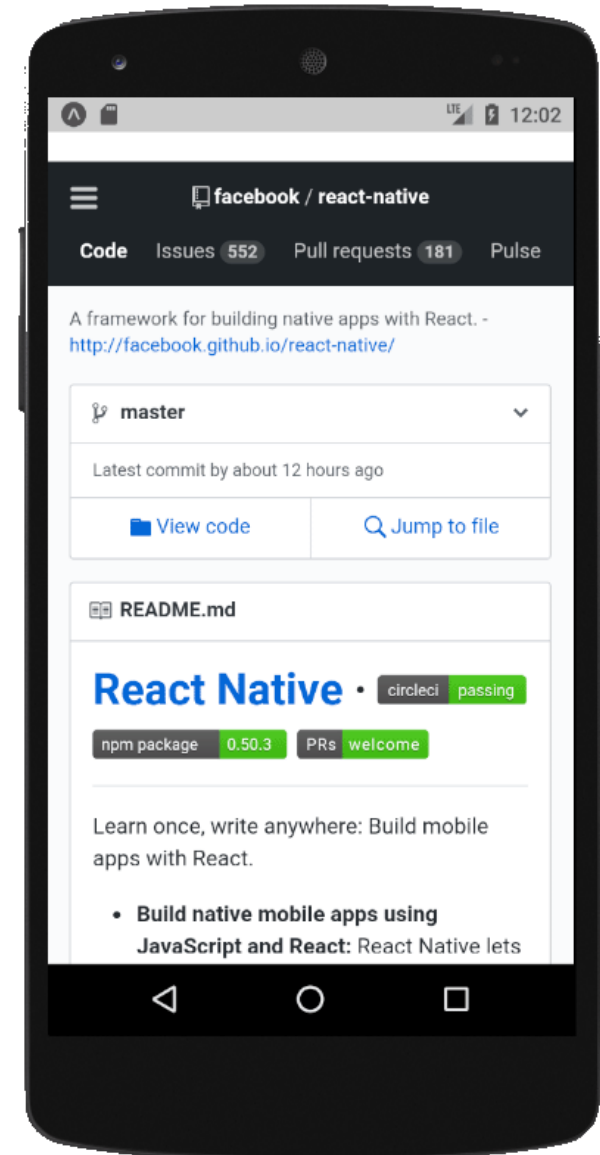
# Modal: Ejemplo

```
import React from 'react';
import { TouchableHighlight, Modal, Text, View } from 'react-native';
export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {modalVisible:false};
  }
  _setModalVisible(visible){ this.setState({modalVisible: visible}); }
  render() {
    return (
      <View style={{flex:1, alignItems:'center', justifyContent:'flex-start'}}>
        <Modal animationType="slide" transparent={false}
          visible={this.state.modalVisible}
          onRequestClose={() => {alert("User has tapped the back button")}}>
          <View style={{padding:30, alignItems:'center', justifyContent:'center'}}>
            <TouchableHighlight style={{alignSelf:'flex-end'}}
              onPress={() => {this._setModalVisible(false)}}>
              <Text style={{fontSize:20, marginBottom: 20}}>X</Text>
            </TouchableHighlight>
            <Text style={{fontSize:35}}>Hello World!</Text>
          </View>
        </Modal>
        <TouchableHighlight onPress={() => {this._setModalVisible(true)}}>
          <Text style={{fontSize:35}}>Show Modal</Text>
        </TouchableHighlight>
      </View>
    )
  }
}
```



# WebView

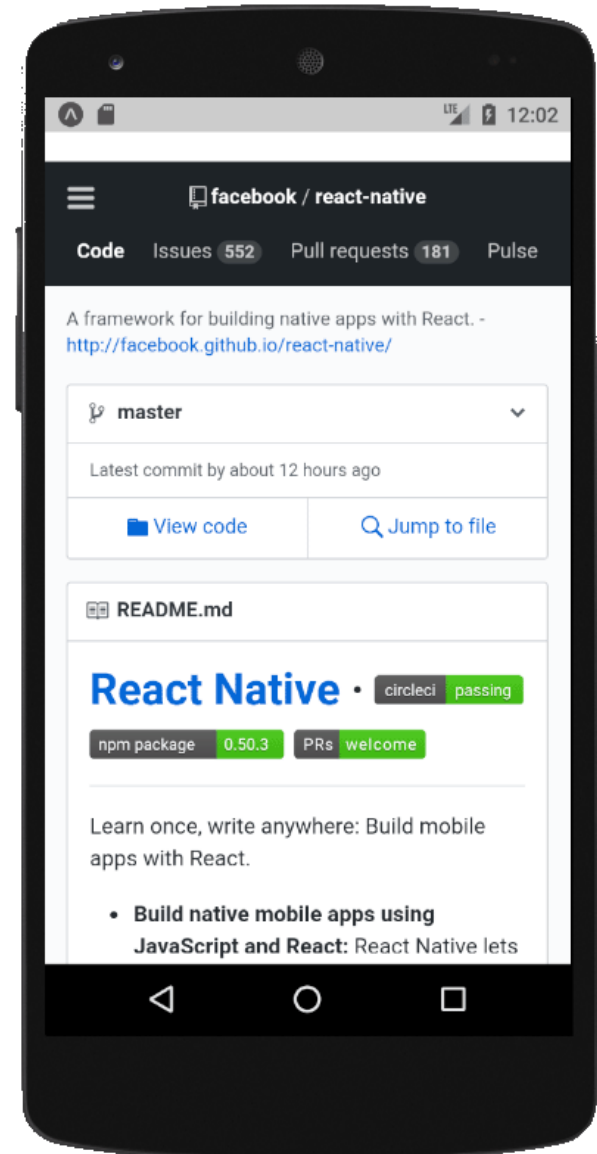
- El componente **WebView** permite renderizar **contenido web** en una vista nativa
- Permite especificar diversas **propiedades** para el contenido web: icono de carga, estilo, habilitación de JavaScript, etc.
- Permite inyectar **código JavaScript**
- Proporciona diversos **callbacks**: *onLoad*, *onError*, *onMessage*, *onNavigationStateChange*, etc.
- Documentación:  
<https://facebook.github.io/react-native/docs/webview.html>



# WebView: Ejemplo

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

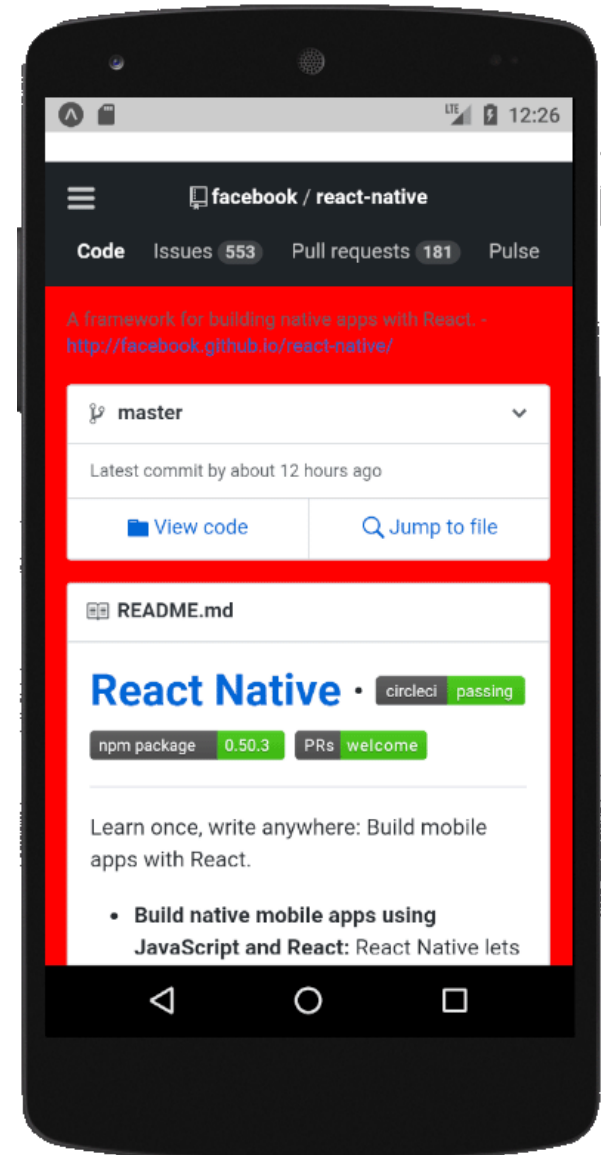
export default class App extends React.Component {
  render() {
    return (
      <View style={{flex:1}}>
        <WebView
          source={{uri:
            'https://github.com/facebook/react-native'}}
          startInLoadingState={true}
          style={{marginTop: 20}}
        />
      </View>
    )
  }
}
```



# WebView: Ejemplo II

```
import React from 'react';
import { WebView, View } from 'react-native';
export default class App extends React.Component {
  render() {
    var jsCode = "document.querySelector('body').
                  style.backgroundColor = 'red';";

    return (
      <View style={{flex:1}}>
        <WebView
          source={{uri:
            'https://github.com/facebook/react-native'}}
          startInLoadingState={true}
          injectedJavaScript={jsCode}
          style={{marginTop: 20}}
        />
      </View>
    )
  }
}
```



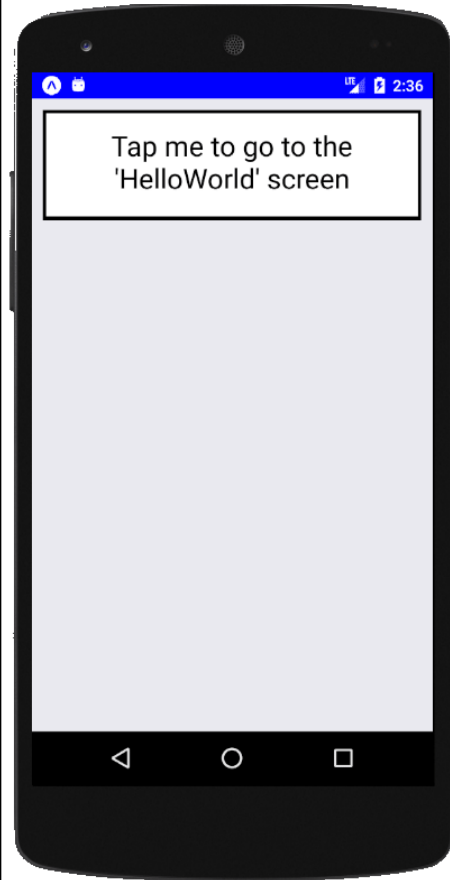
# StatusBar

---

- El componente **StatusBar** controla la **barra de estado** de la aplicación
- Podemos definir algunas **propiedades**: color de fondo, estilo, visible/oculta, etc.
- Las propiedades pueden ser establecidas directamente en el componente StatusBar (recomendado) o a través de una **API** ofrecida por él
- Pueden existir varios componentes StatusBar al mismo tiempo
  - Esto permite especificar **barras de estado diferentes** para diferentes pantallas de una misma aplicación
  - En este caso las propiedades se mezclarán siguiendo el orden en el que los componentes fueron montados
- Documentación:  
<https://facebook.github.io/react-native/docs/statusbar.html>

# StatusBar: Ejemplo con StackNavigator

```
import React from 'react';
import { StatusBar, View } from 'react-native';
import { StackNavigator } from 'react-navigation';
import IndexScreen from './index_screen';
import HelloWorldScreen from './hello_world_screen';
const AppContent = StackNavigator({
  Index: {
    screen: IndexScreen,
    navigationOptions: { header: null }
  },
  HelloWorld: {
    screen: HelloWorldScreen,
    navigationOptions: { header: null }
  }
});
export default class App extends React.Component {
  render(){
    return (
      <View style={{flex: 1}}>
        <StatusBar backgroundColor="blue" barStyle="light-content" />
        <AppContent />
      </View>
    )
  }
}
```



# StatusBar: Ejemplo con StackNavigator II

```
import React from 'react';
import { StatusBar, View } from 'react-native';
import { StackNavigator } from 'react-navigation';
[...]
```

```
export default class App extends React.Component {
  constructor(props) {
    super(props);
    StatusBar.setBackgroundColor("blue",false);
    setTimeout(() => {
      StatusBar.setBackgroundColor("red",true);
    },1500);
  }
  render(){
    return (
      <View style={{flex: 1}}>
        <StatusBar barStyle="light-content" />
```



# StatusBar: Ejemplo con StackNavigator III

*hello\_world\_screen.js*

```
import React from 'react';
import { StatusBar, Text, View } from 'react-native';
import MyButton from './my_button';

export default class HelloWorldScreen extends React.Component {
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'center' }}>
        <StatusBar hidden={true} />
        <Text style={{ fontSize:30 }}>Hello World!</Text>
        <MyButton
          onPress={() => this.props.navigation.goBack(null)}
          text={"Tap me to go back"}/>
      </View>
    )
  }
}
```



# React Native



**Promesas y  
funciones asíncronas**

# Promesas

---

- Las promesas son un tipo de **objeto** utilizado para la llamada y ejecución de **funciones asíncronas**
- Representan un valor **no necesariamente conocido** en el momento de su creación
- Permiten asociar **manejadores** para el éxito y fracaso de operaciones que serán completadas en el futuro
- Permiten a funciones asíncronas devolver valores de forma similar a las síncronas. Estas funciones **devolverán promesas** en lugar del valor final.
- Existen **diferentes implementaciones** (navegadores web, bibliotecas JavaScript, Node.js, React Native, ...)
- Más información:  
<https://developers.google.com/web/fundamentals/getting-started/primers/promises>  
[https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/Promise](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise)  
<http://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects>

# Promesas

---

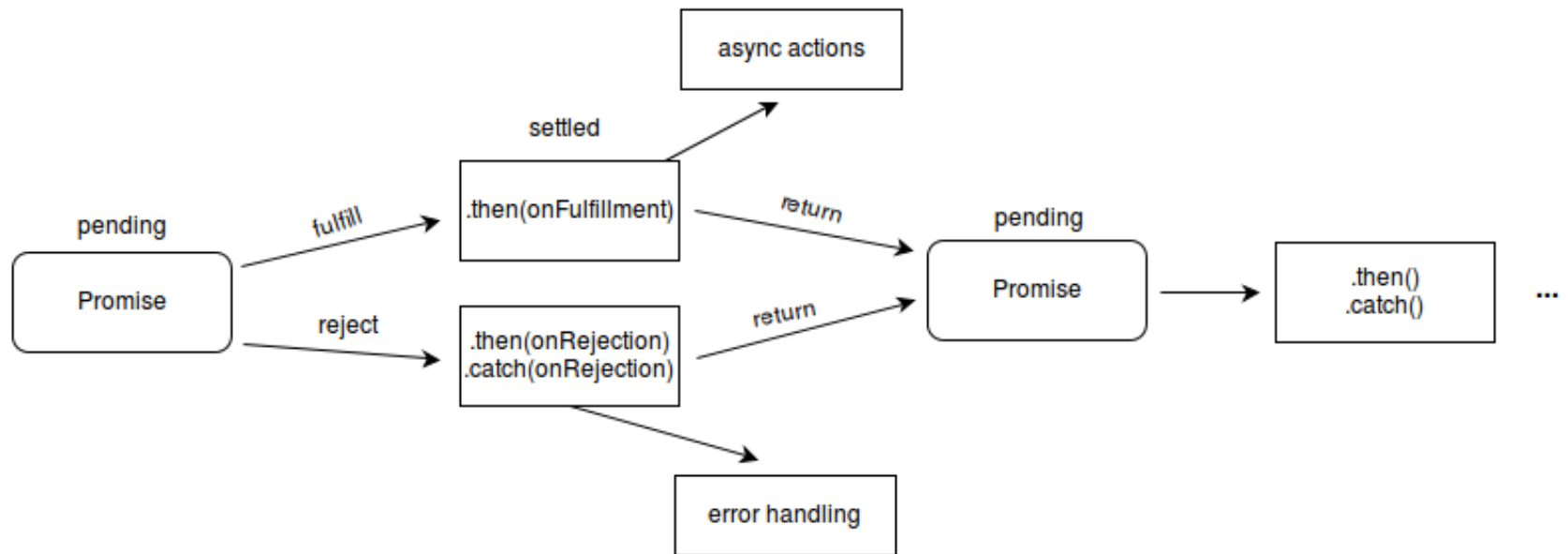
- Una promesa puede estar en uno de los siguientes estados:
  - **Pendiente** (*pending*): estado inicial, ni cumplida ni rechazada.
  - **Cumplida** (*fulfilled*): operación completada con éxito.
  - **Rechazada** (*rejected*): operación fallida.

Se dice que una promesa está **determinada** (*settled*) si se ha cumplido o rechazado, es decir, si no está pendiente.

Todas las promesas determinadas se consideran **resueltas** (*resolved*).

- Una **promesa pendiente** puede llegar a ser **cumplida** con un **valor** o **rechazada** con una **razón**
- Cuando se determina una promesa se invocan sus **manejadores** asociados mediante las funciones **then** y **catch**
- Las promesas se pueden **encadenar**

# Encadenamiento de Promesas



# Promesas y Eventos

---

- Las **promesas** se parecen a los **eventos**, pero tienen diferencias significativas:
  - **Una promesa solo puede tener éxito o fallar una vez.** Los manejadores de las promesas no son llamados en múltiples ocasiones mientras que los escuchadores de eventos son llamados cada vez que ocurre el evento.
  - Si una promesa se ha **determinado** y posteriormente se añade un **manejador**, este será llamado aunque el evento causante de la determinación de la promesa haya tenido lugar **antes** de añadir dicho manejador. Esto no ocurre con los eventos.
- Las promesas son más adecuadas que los eventos para gestionar **callbacks asíncronos de éxito/fracaso**

# Promesas en React Native

---

Las promesas de React Native se basan en **ECMAScript 6** pero también incorporan funcionalidades de **ECMAScript 7**:

- **async**: esta palabra clave permite declarar **funciones asíncronas: funciones que devuelven promesas y pueden usar la expresión await**.  
Si una función asíncrona **devuelve un valor** la promesa se **cumplirá** con el valor devuelto y si **lanza una excepción** la promesa se **rechazará** con dicha excepción.
- **await**: esta palabra clave permite indicar a la aplicación que **salga temporalmente de una función asíncrona** y reanude su ejecución cuando se complete una operación determinada. La aplicación puede ejecutar otro código mientras la operación está en progreso.  
**Se puede esperar (*await*) el resultado de cualquier función asíncrona (*async*), es decir, de cualquier promesa.**

# Ejemplo de función asíncrona en React Native

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

export default class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {text: "Hello World"};
  }
  componentDidMount() {
    this._getNewText().then(function(newText) {
      this.setState({text: newText});
    }.bind(this));
  }
  async _getNewText() {
    var newText = await this._fetchNewText();
    return newText;
  }
  async _fetchNewText() {
    return new Promise((resolve) => { // Simulate async operation
      setTimeout(function(){resolve("Goodbye World");},1500);
    })
  }
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'flex-start' }}>
        <Text style={{fontSize:30}}>{this.state.text}</Text>
      </View>
    )
  }
}
```



# React Native



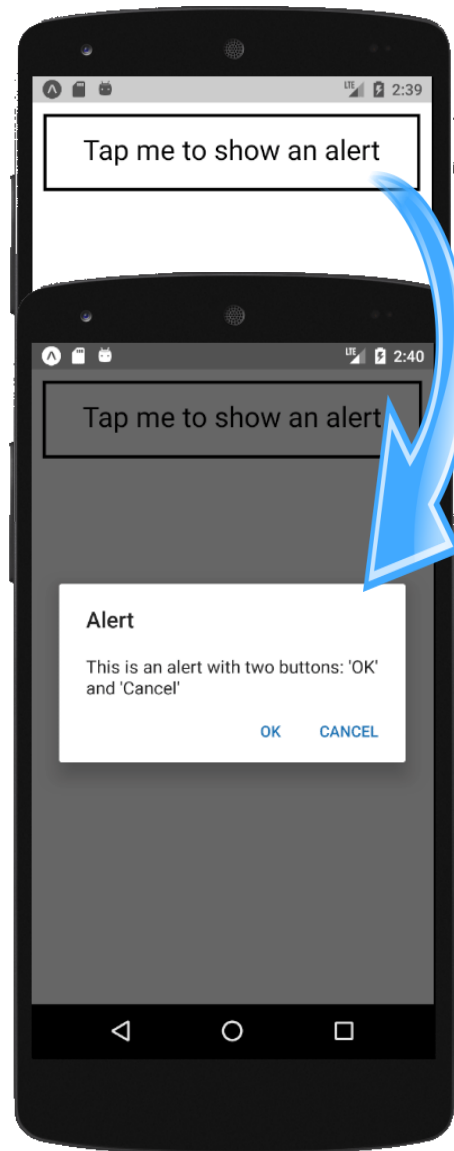
## APIs

# Alert

- Permite mostrar **diálogos de alerta**
- Documentación:  
<https://facebook.github.io/react-native/docs/alert.html>

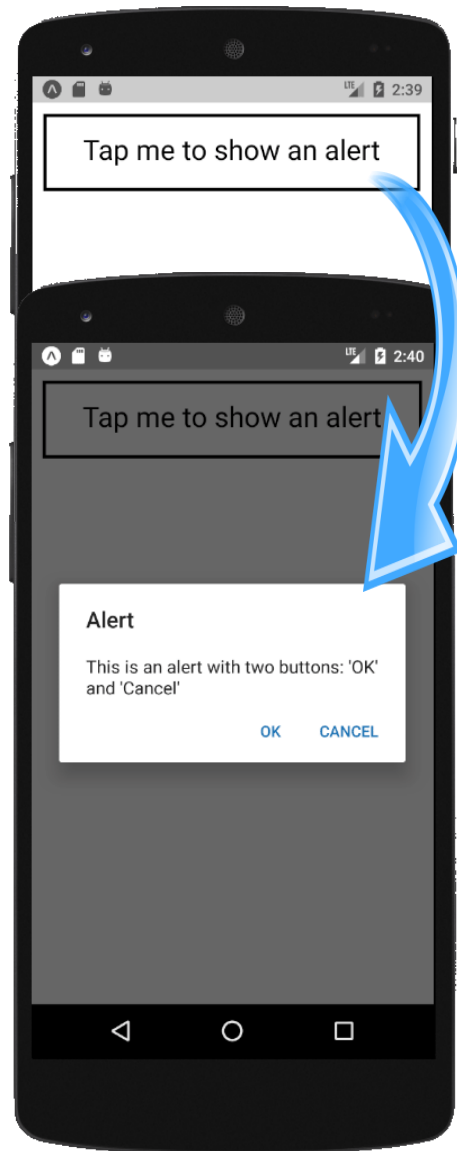
```
import React from 'react';
import { Alert, StyleSheet, Text, TouchableHighlight, View } from
'react-native';

export default class App extends React.Component {
  _onPressButton() {
    Alert.alert(
      "Alert",
      "This is an alert with two buttons: 'OK' and 'Cancel'",
      [
        {text: 'OK', onPress: () => console.log('OK pressed')},
        {text: 'Cancel', onPress: () => console.log('Cancel pressed')}
      ],
      { cancelable: false }
    )
  }
}
```



# Alert

```
render() {  
  return (  
    <View>  
      <TouchableHighlight onPress={this._onPressButton}>  
        <Text style={styles.text}>Tap me to show an alert</Text>  
      </TouchableHighlight>  
    </View>  
  )  
}  
}  
  
const styles = StyleSheet.create({  
  text: {  
    padding: 15,  
    margin: 10,  
    backgroundColor: 'white',  
    color: 'black',  
    borderWidth: 3,  
    borderColor: 'black',  
    fontSize: 25,  
    textAlign: 'center'  
  }  
})
```



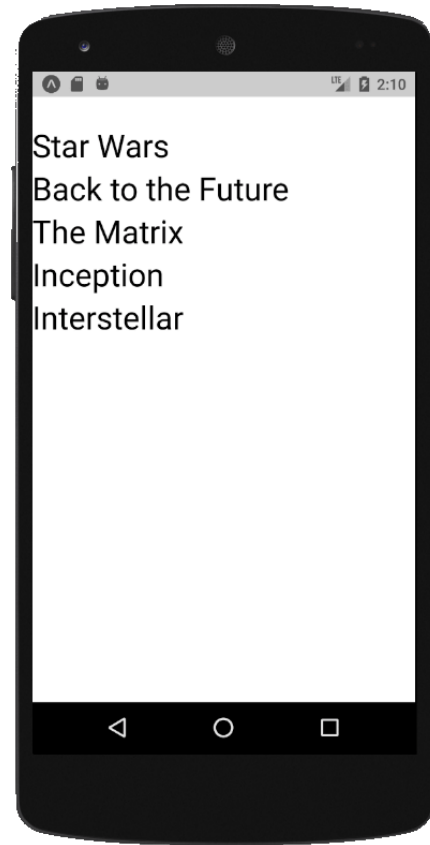
# Fetch

---

- **API JavaScript** para cargar recursos de una red como Internet a partir de sus URLs
- Permite realizar peticiones **HTTP**
- Similar a la API **XMLHttpRequest** soportada en la mayoría de navegadores web
- Las operaciones son **asíncronas**: las funciones de la API Fetch devuelven **promesas**
- React Native también incorpora la API **XMLHttpRequest**, por lo que se puede usar tanto esta API directamente como bibliotecas que dependan de ella (ej: axios)
- Documentación:  
<https://facebook.github.io/react-native/docs/network.html>  
[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)  
<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

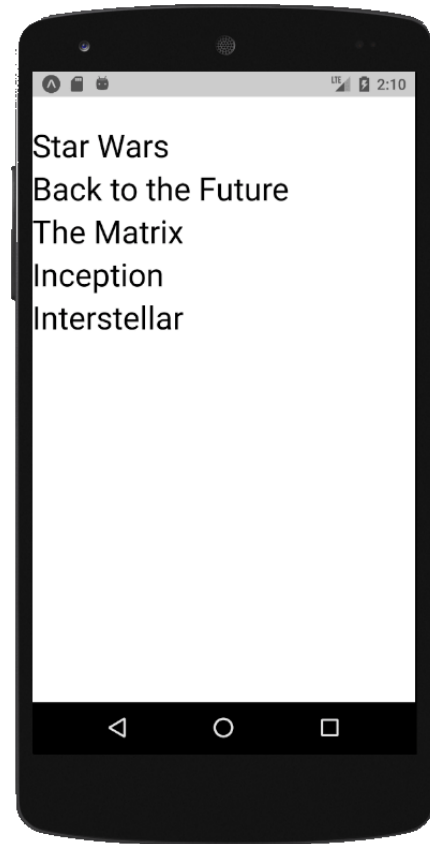
# Fetch: Ejemplo

```
async getMoviesFromAPI(){  
  var url = 'https://facebook.github.io/react-native/movies.json';  
  var response = await fetch(url,{  
    method: 'GET',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    }  
  });  
  var responseJson = await response.json();  
  return responseJson.movies;  
}
```



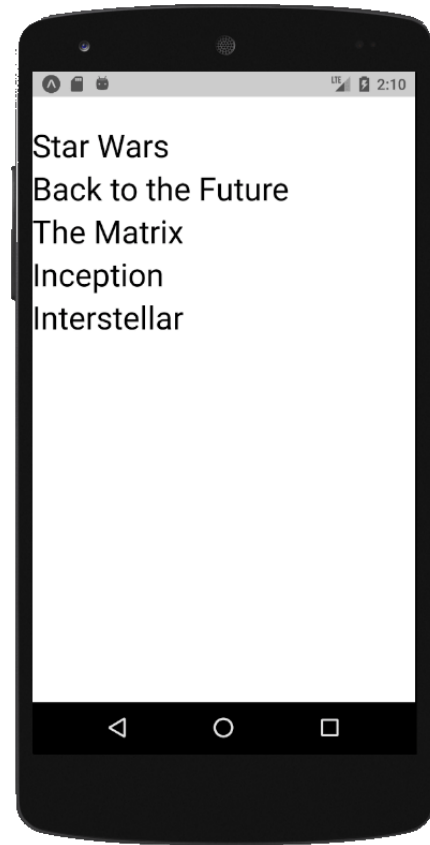
# Fetch: Ejemplo

```
import React from 'react';
import { Text, View } from 'react-native';
export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {moviesUI: []};
  }
  componentDidMount(){
    this.getMoviesFromAPI().then(function(movies){
      if((typeof movies == "object") && (movies instanceof Array)){
        var moviesUI = [];
        for(let i=0; i<movies.length; i++){
          moviesUI.push(<Text style={{fontSize:30}} key={i}>{movies[i].title}</Text>)
        }
        this.setState({moviesUI: moviesUI});
      }
    }).bind(this).catch(function(exception){
      alert("Exception: " + exception + "");
    });
  }
  async getMoviesFromAPI(){[...]}
  render() {
    return (<View style={{flex: 1, marginTop: 25}}>{this.state.moviesUI}</View>)
  }
}
```



# Fetch: Ejemplo con FlatList

```
import React from 'react';
import { FlatList, Text, View } from 'react-native';
export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {movies: []};
  }
  componentDidMount(){
    this.getMoviesFromAPI().then(function(movies){
      if((typeof movies == "object") && (movies instanceof Array)){
        for(let i=0; i<movies.length; i++){ movies[i].key = i; }
        this.setState({movies: movies});
      }
    }).bind(this)).catch(function(exception){
      alert("Exception: " + exception + "");
    });
  }
  async getMoviesFromAPI() {[...]}
  render() {
    return (<View style={{flex: 1, marginTop: 25}}>
      <FlatList data={this.state.movies}
        renderItem={({item}) => <Text style={{fontSize: 30}}>{item.title}</Text>}</FlatList>
    </View>)
  }
}
```



# Fetch: Ejemplo con HTTP POST

---

```
async sendPostToEndpoint() {  
  var url = 'https://mywebsite.com/endpoint/';  
  var response = await fetch(url, {  
    method: 'POST',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    },  
    body: JSON.stringify({  
      firstParam: 'firstParamValue',  
      secondParam: 'secondParamValue'  
    })  
  });  
  var parsedResponse = await response.json();  
  return parsedResponse;  
}
```

# AsyncStorage

---

- **API JavaScript** que proporciona a las aplicaciones React Native un sistema de **almacenamiento clave-valor** simple, asíncrono y persistente
- Misma función que proporciona la API HTML5 **LocalStorage** en aplicaciones web
- Todas las funciones de la API devuelven **promesas**
- Implementación
  - *Android*: SQLite o RocksDB (<http://rocksdb.org>)
  - *iOS*: diccionarios (valores pequeños) y ficheros (valores grandes)
- Documentación:  
<https://facebook.github.io/react-native/docs/asyncstorage.html>

# AsyncStorage: Funciones

---

Función	Descripción
<b>getItem</b> (key, callback?)	Obtiene el <b>valor</b> para la clave <b>key</b> e invoca un <b>callback</b> al finalizar.
<b>setItem</b> (key, value, callback?)	Establece el valor <b>value</b> para la clave <b>key</b> e invoca un <b>callback</b> al finalizar.
<b>removeItem</b> (key, callback?)	Borra el <b>valor</b> para la clave <b>key</b> e invoca un <b>callback</b> al finalizar.
<b>mergeItem</b> (key, value, callback?)	Hace un merge del valor de la clave <b>key</b> y el valor <b>value</b> pasado como parámetro asumiendo que ambos son objetos convertidos a <b>cadenas JSON</b> . Guarda el valor resultante en la clave <b>key</b> e invoca un <b>callback</b> al finalizar.
<b>Clear</b> (callback?)	Elimina todos los pares clave-valor almacenados con AsyncStorage, incluso los de otras aplicaciones.
<b>getAllKeys</b> (callback?)	Obtiene todas las claves conocidas por la aplicación.

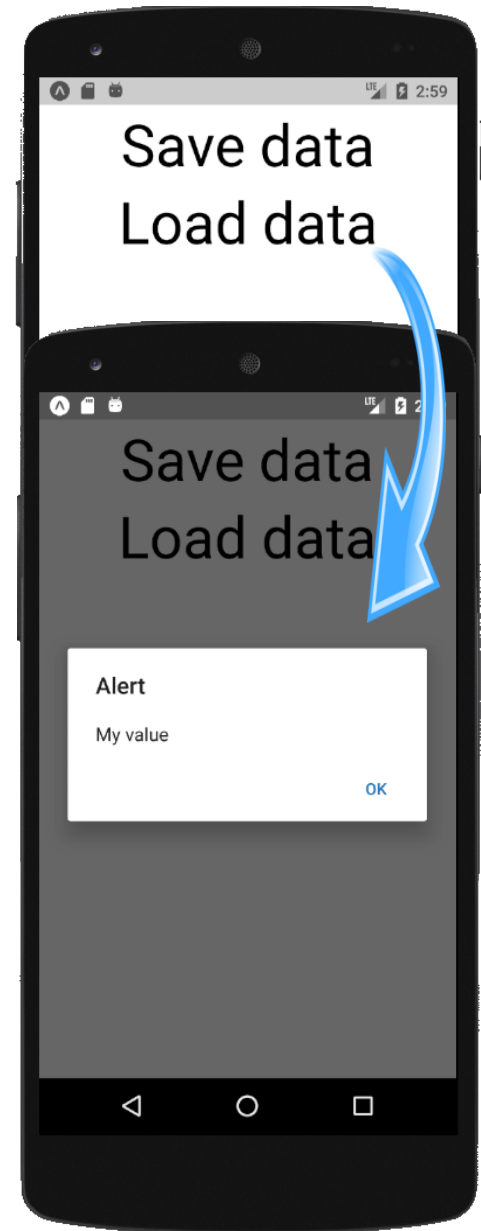
# AsyncStorage: Funciones

Función	Descripción
<b>multiGet</b> (keys, callback?)	Obtiene los <b>valores</b> correspondientes al array de claves <b>keys</b> . El <b>callback</b> será invocado con el array de los pares clave-valor encontrados.
<b>multiSet</b> (keyValuePairs, callback?)	Permite almacenar múltiples pares clave-valor proporcionados en el array <b>keyValuePairs</b> de la forma: <pre>[[ 'clave1', 'valor1' ], [ 'clave2', 'valor2' ]]</pre> Al finalizar, el <b>callback</b> se invoca con un array que contiene cualquier error encontrado durante la operación.
<b>multiRemove</b> (keys, callback?)	Borra los valores correspondientes a todas las claves contenidas en el array <b>keys</b> . Al finalizar, el <b>callback</b> se invoca con un array que contiene cualquier error encontrado durante la operación.
<b>multiMerge</b> (keyValuePairs, callback?)	Permite realizar varios <i>merges</i> con una sola operación. Las claves y valores para realizar los merges se pasan en el array <b>keyValuePairs</b> . Al finalizar, el <b>callback</b> se invoca con un array que contiene cualquier error encontrado durante la operación.

# AsyncStorage: Ejemplo

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

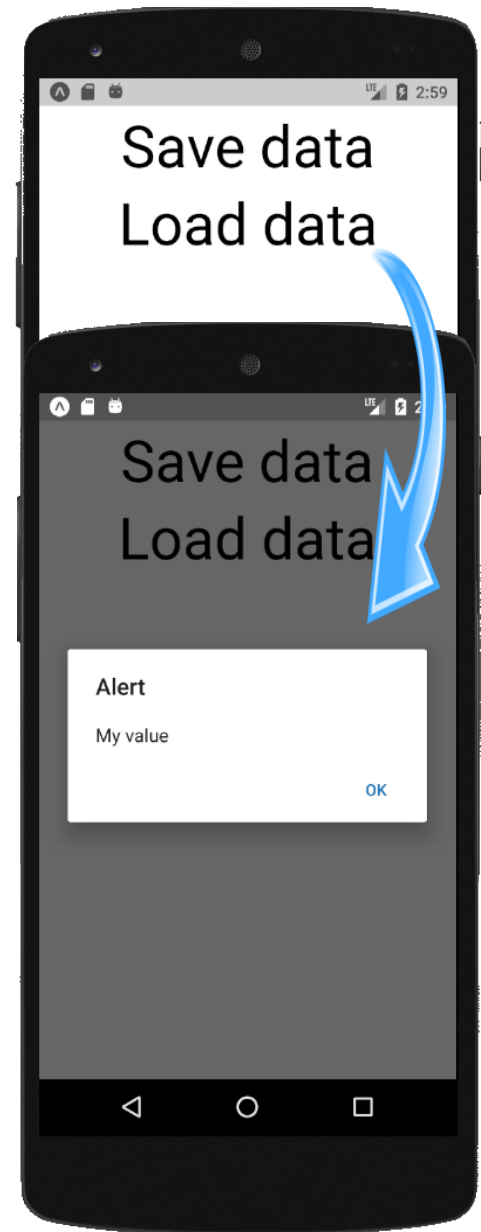
export default class App extends React.Component {
  async _saveData() {
    try {
      await AsyncStorage.setItem('@HelloWorld:myKey', 'My value');
    } catch (error) { // Error saving data }
  }
  async _loadData() {
    try {
      var value = await AsyncStorage.getItem('@HelloWorld:myKey');
      if (value !== null) {
        alert(value);
      }
    } catch (error) { // Error retrieving data }
  }
  [...]
}
```



# AsyncStorage: Ejemplo

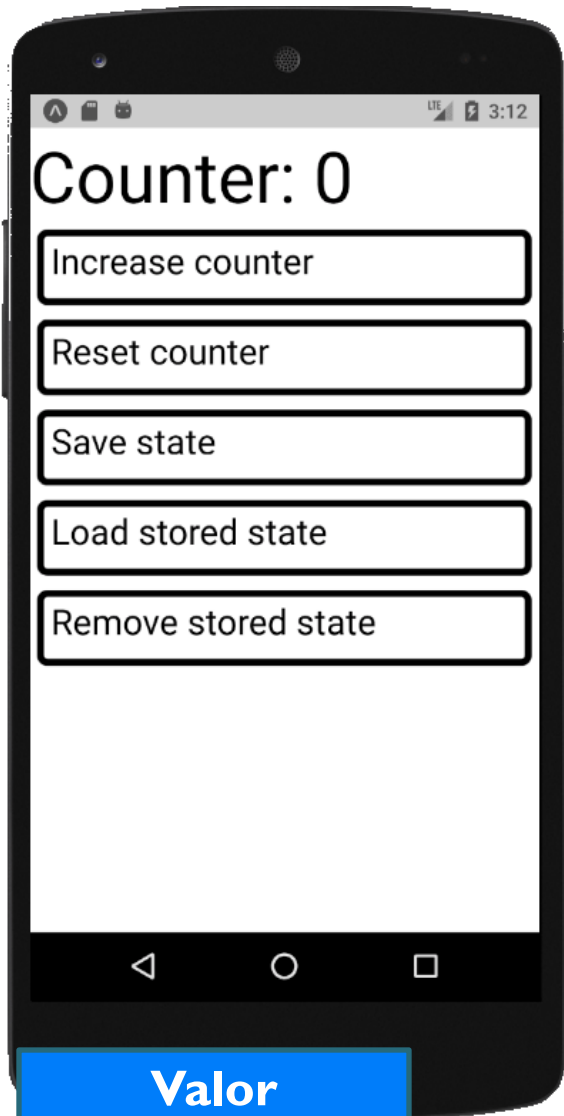
```
import React from 'react';
import { AsyncStorage, TouchableOpacity, Text, View } from
'react-native';
```

```
export default class App extends React.Component {
  [...]
  render(){
    return (
      <View style={{ flex:1, alignItems:'center',
        justifyContent:'flex-start' }}>
        <TouchableOpacity onPress={this._saveData}>
          <Text style={{fontSize: 50}}>Save data</Text>
        </TouchableOpacity>
        <TouchableOpacity onPress={this._loadData}>
          <Text style={{fontSize: 50}}>Load data</Text>
        </TouchableOpacity>
      </View>
    )
  }
}
```



# AsyncStorage: Ejemplo II

---



**Valor  
almacenado:**

-

# AsyncStorage: Ejemplo II



# AsyncStorage: Ejemplo II



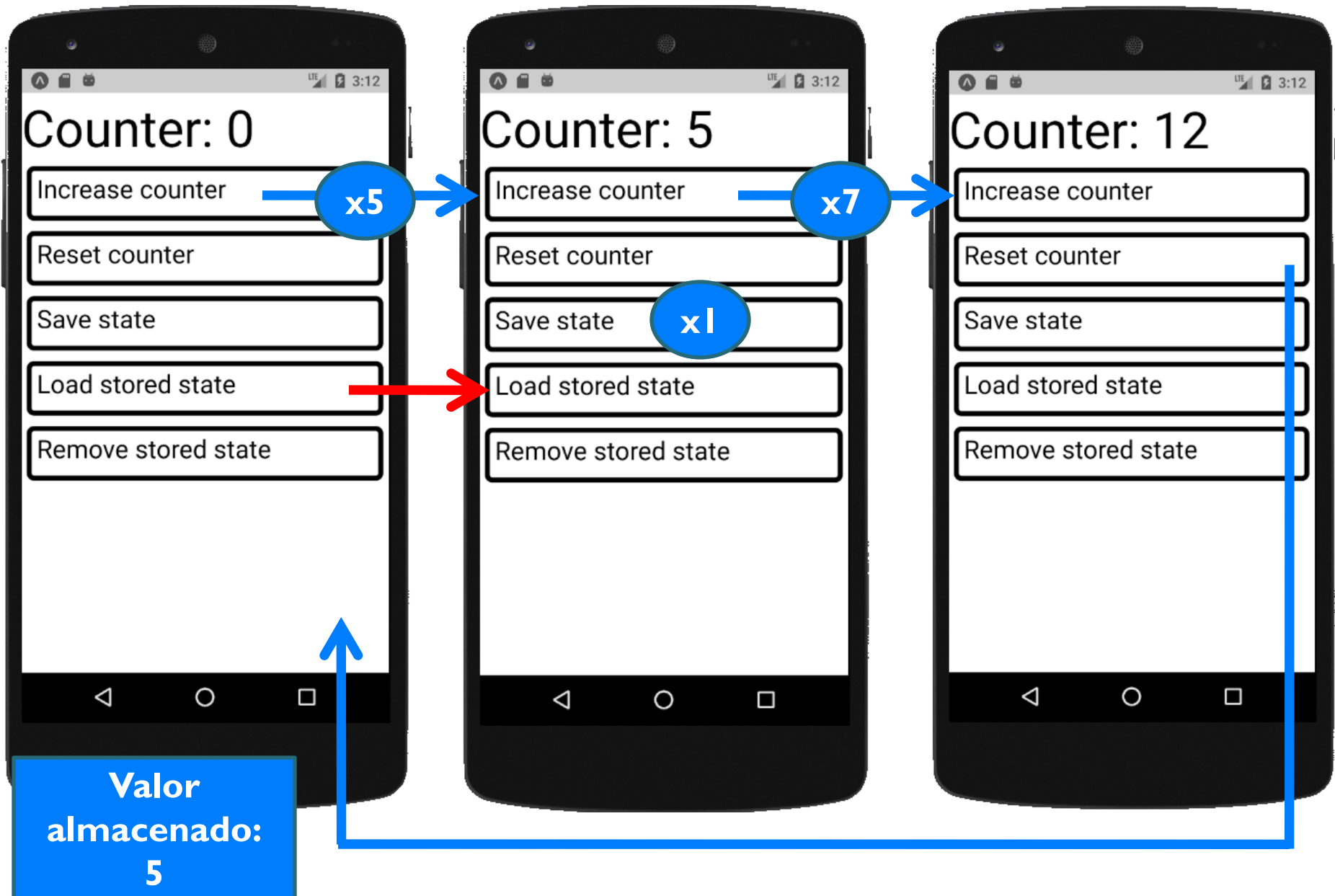
# AsyncStorage: Ejemplo II



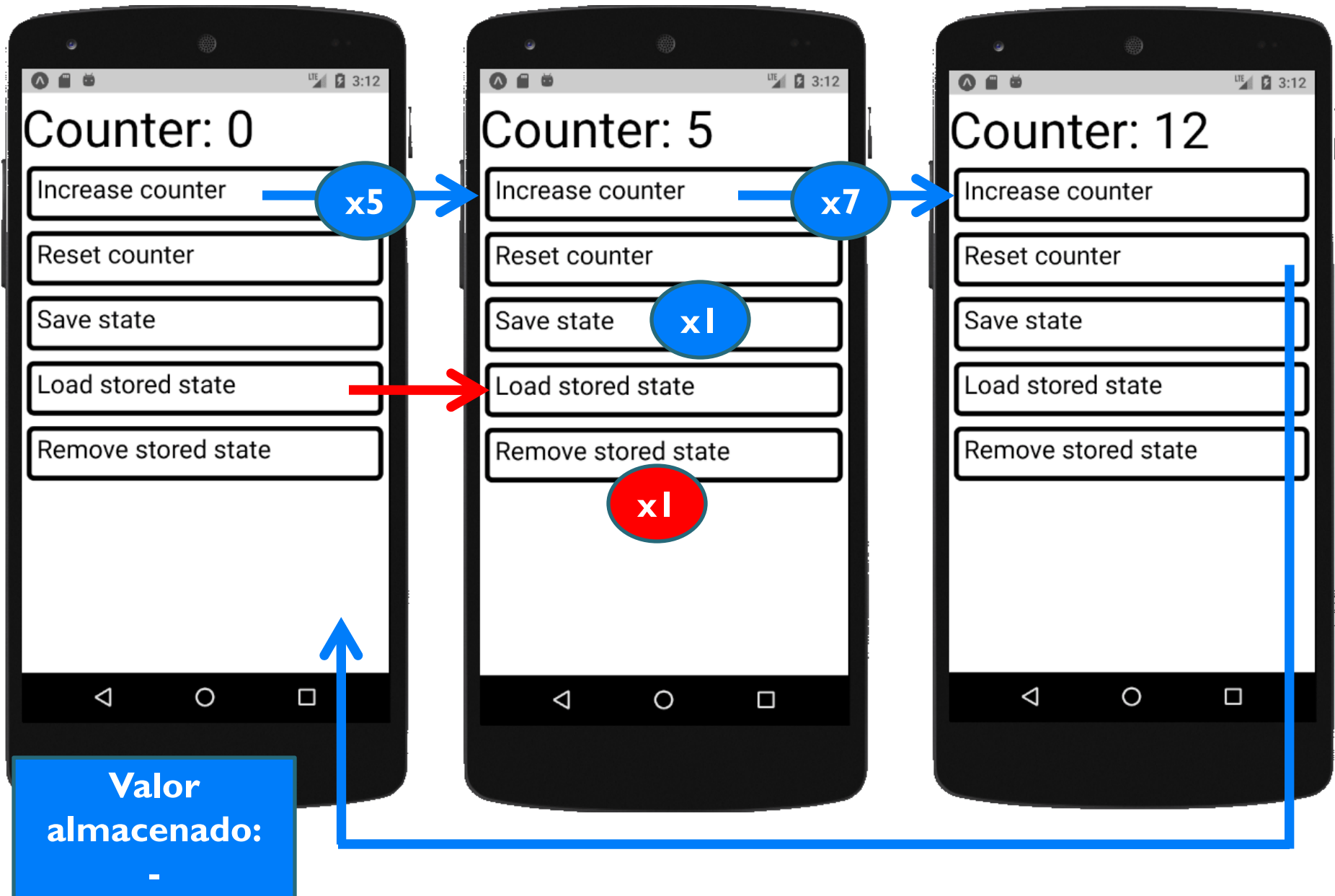
# AsyncStorage: Ejemplo II



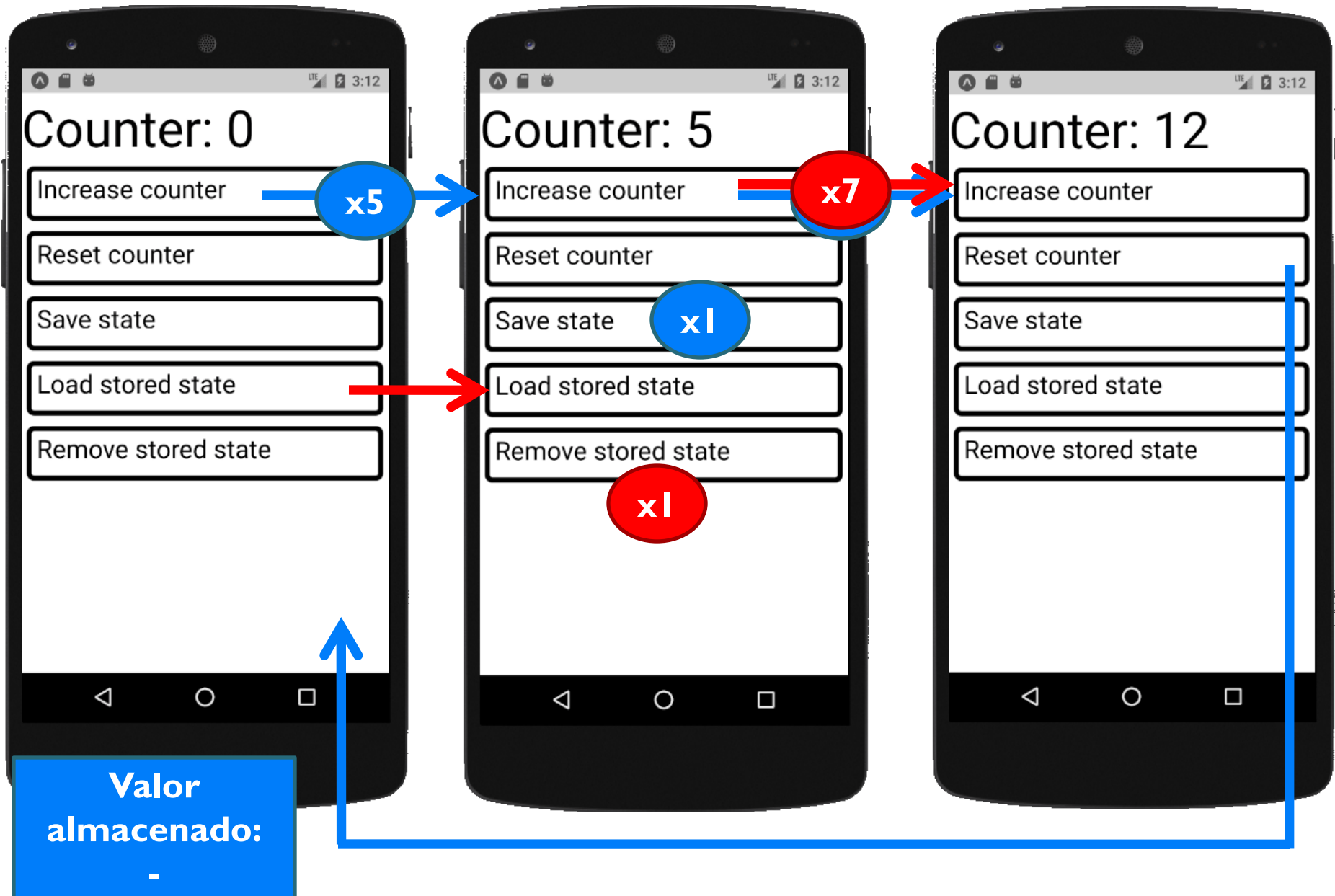
# AsyncStorage: Ejemplo II



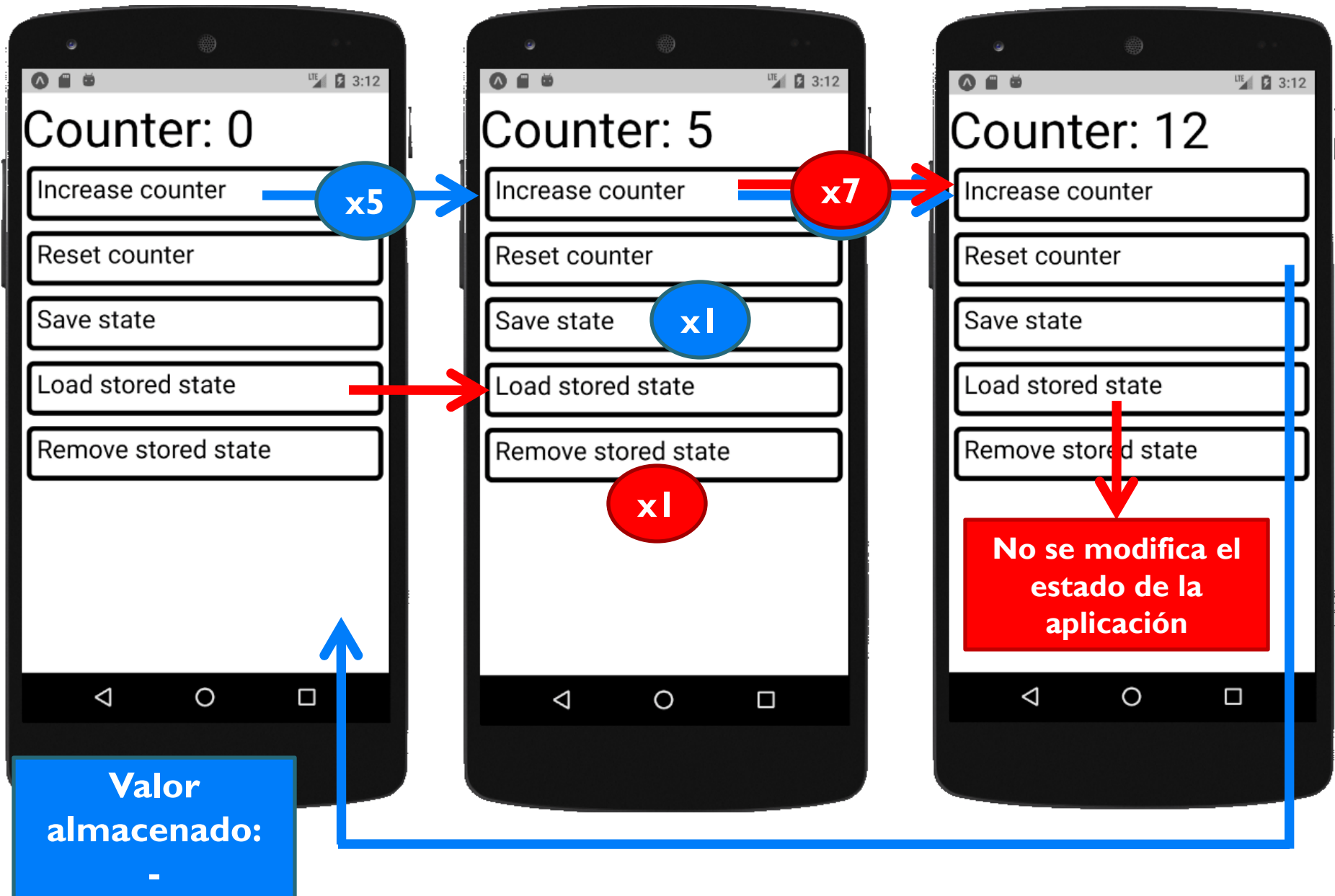
# AsyncStorage: Ejemplo II



# AsyncStorage: Ejemplo II



# AsyncStorage: Ejemplo II



# AsyncStorage: Ejemplo II

---

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

export default class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {counter: 0};
    this._loadState();
  }
  _increaseCounter() { [...]}
  _resetCounter() { [...]}
  async _saveState() { [...]}
  async _loadState() { [...]}
  async _removeState() { [...]}
  render() { [...]}
}
[...]
```

# AsyncStorage: Ejemplo II

```
render(){
  return (
    <View style={{ flex:1, alignItems:'stretch', justifyContent:'flex-start' }}>
      <Text style={{fontSize: 50}}>Counter: {this.state.counter}</Text>
      <TouchableOpacity onPress={this._increaseCounter.bind(this)}>
        <Text style={styles.button}>Increase counter</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={this._resetCounter.bind(this)}>
        <Text style={styles.button}>Reset counter</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={this._saveState.bind(this)}>
        <Text style={styles.button}>Save state</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={this._loadState.bind(this)}>
        <Text style={styles.button}>Load stored state</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={this._removeState.bind(this)}>
        <Text style={styles.button}>Remove stored stated</Text>
      </TouchableOpacity>
    </View>
  )
}
```

# AsyncStorage: Ejemplo II

---

```
_increaseCounter(){  
  this.setState({counter: (this.state.counter+1)});  
}
```

```
_resetCounter(){  
  this.setState({counter: 0});  
}
```

```
async _saveState(){  
  try {  
    var currentState = JSON.stringify(this.state);  
    await AsyncStorage.setItem('@HelloWorld:state',currentState);  
  } catch (error) {  
    // Error saving state  
  }  
}
```

# AsyncStorage: Ejemplo II

---

```
async _loadState(){
  try {
    var storedState = await AsyncStorage.getItem('@HelloWorld:state');
    if (storedState !== null){
      var state = JSON.parse(storedState);
      this.setState(state);
    }
  } catch (error) {
    // Error retrieving state
  }
}

async _removeState(){
  try {
    await AsyncStorage.removeItem('@HelloWorld:state');
  } catch (error) {
    // Error removing state
  }
}
```

# AsyncStorage: Ejemplo II

---

```
var styles = StyleSheet.create({
  button: {
    fontSize: 25,
    color: 'black',
    borderWidth: 5,
    borderRadius: 5,
    padding: 5,
    paddingLeft: 10,
    margin: 5,
    borderColor: 'black'
  }
});
```

# Vibration

---

- **API JavaScript** que permite activar la vibración de los dispositivos
- Tiene dos funciones:
  - **vibrate**: activa la vibración
  - **cancel**: detiene la vibración
- Las funciones no tienen ningún efecto en aquellos dispositivos que no soporten vibración (ej: emuladores)
- Android permite configurar el tiempo durante el cual vibrará el dispositivo mientras que en iOS el dispositivo vibrará durante una cantidad de tiempo fija
- Documentación:  
<https://facebook.github.io/react-native/docs/vibration.html>

# Vibration: Ejemplo

```
import React from 'react';
import { Vibration, TouchableHighlight, Text, View } from 'react-native';
export default class App extends React.Component {
  _vibrate() {
    Vibration.vibrate(3000);
    // Android: the device will vibrate for 3s
    // iOS: duration is not configurable, so the device will vibrate for a fixed time (about 500ms)
  }
  _cancelVibration() {
    Vibration.cancel(); //Stop vibration (Android and iOS)
  }
  render() {
    return (
      <View style={{ flex:1, alignItems:'center', justifyContent:'center' }}>
        <TouchableHighlight onPress={this._vibrate}>
          <Text>Tap me to vibrate</Text>
        </TouchableHighlight>
        <TouchableHighlight onPress={this._cancelVibration}>
          <Text>Tap me to cancel vibration</Text>
        </TouchableHighlight>
      </View>
    )
  }
}
```

# CameraRoll

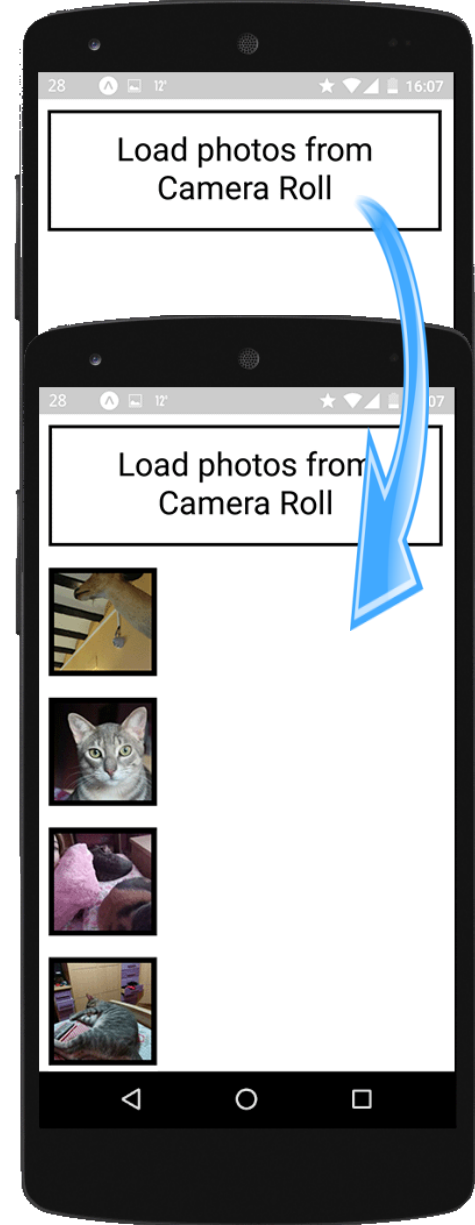
---

- **API JavaScript** que proporciona acceso a la **galería de fotos de la cámara**
- Tiene dos funciones:
  - **getPhotos**: permite obtener fotos y vídeos existentes en la galería
  - **saveToCameraRoll**: permite guardar nuevas fotos y vídeos en la galería
- Las funciones de la API devuelven **promesas**
- Documentación:  
<https://facebook.github.io/react-native/docs/cameraroll.html>

# CameraRoll: Ejemplo

```
import React from 'react';
import { CameraRoll, FlatList, Image, TouchableHighlight, StyleSheet, Text,
View } from 'react-native';

export default class App extends React.Component {
  constructor(props){ super(props); this.state = {images: []} }
  _onPressButton() { [...] }
  _onGetPhotos(photos){ [...] }
  _renderItem({item}){ [...] }
  render() {
    return (
      <View style={{flex:1,alignItems:'flex-start',justifyContent:'flex-start' }}>
        <TouchableHighlight onPress={this._onPressButton.bind(this)}>
          <Text style={styles.text}>Load photos from Camera Roll</Text>
        </TouchableHighlight>
        <FlatList data={this.state.images}
          renderItem={this._renderItem.bind(this)} />
      </View>
    )
  }
}
```

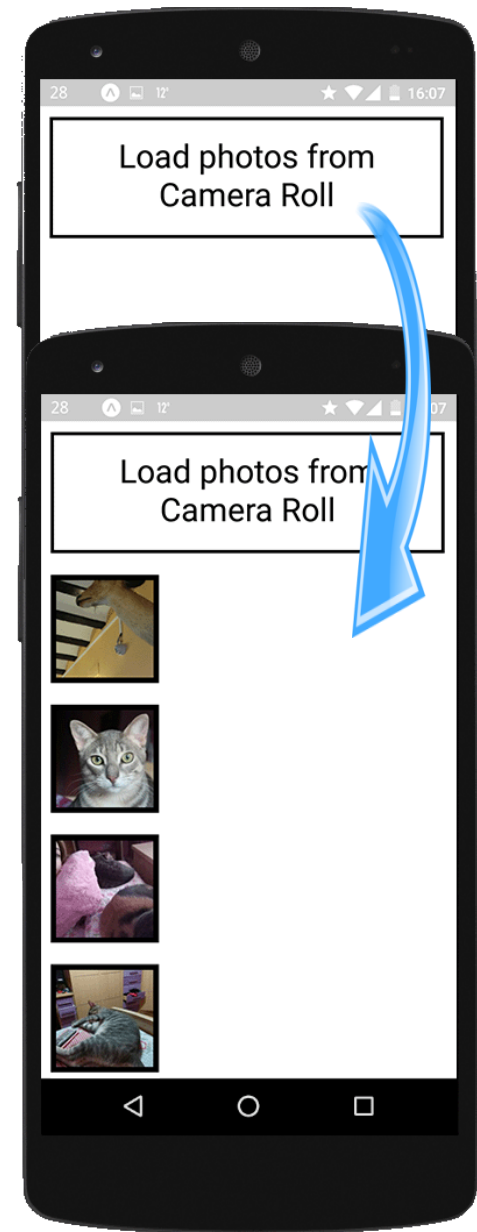


# CameraRoll: Ejemplo

```
_onPressButton(){
  CameraRoll.getPhotos({
    first: 20,
    assetType: 'Photos'
  }).then(r => this._onGetPhotos(r.edges));
}

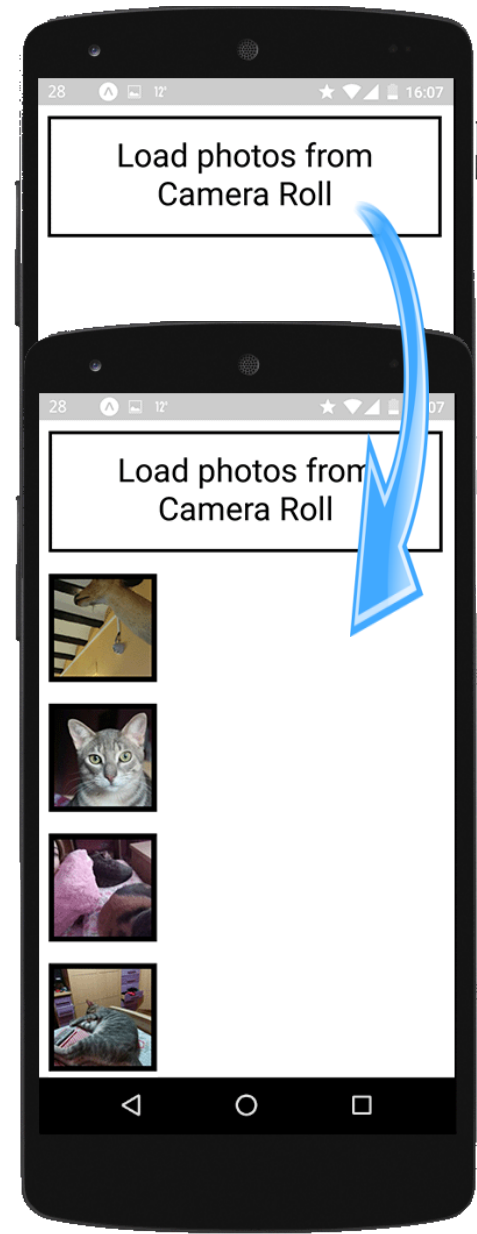
_onGetPhotos(photos){
  var images = photos.map((asset) => {
    let image = asset.node.image;
    image.key = image.uri;
    return image;
  });
  this.setState({images: images});
}

_renderItem({item}){
  return (
    <View>
      <Image style={styles.image} source={{ uri: item.uri }}/>
    </View>
  )
}
```



# CameraRoll: Ejemplo

```
const styles = StyleSheet.create({
  text: {
    padding: 15,
    margin: 10,
    backgroundColor: 'white',
    color: 'black',
    borderWidth: 3,
    borderColor: 'black',
    fontSize: 30,
    textAlign: 'center'
  },
  image: {
    width: 100,
    height: 100,
    margin: 10,
    borderWidth: 5,
    borderColor: 'black'
  }
})
```



# React Native



## Depuración

# Depuración

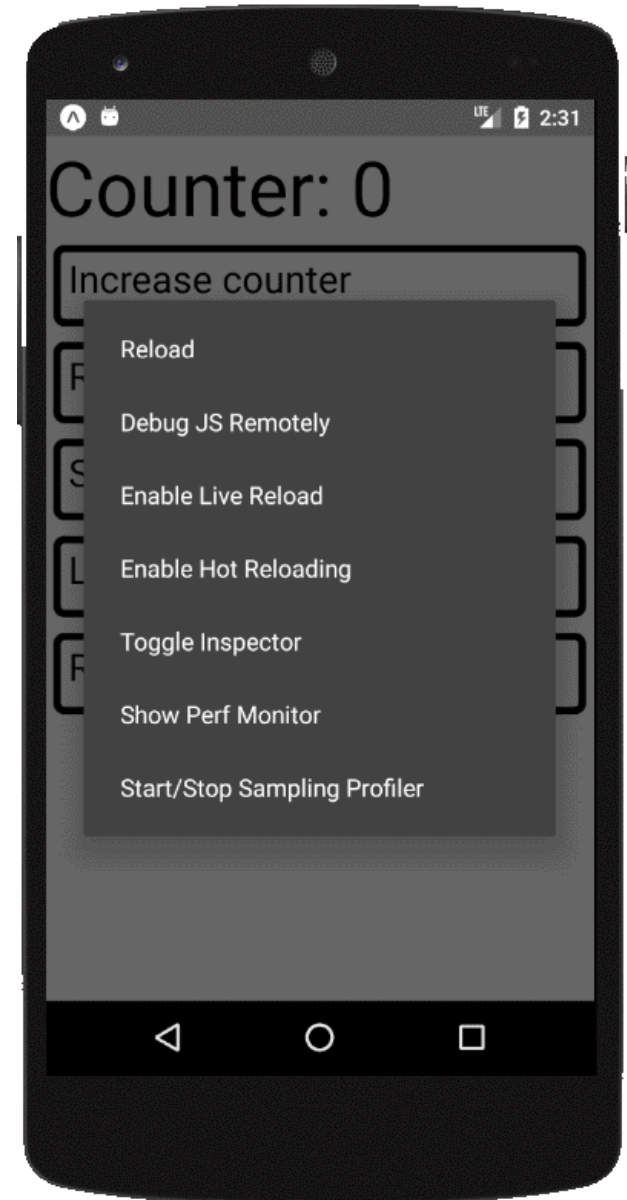
---

- **Mostrar menú de desarrollo**
  - Emulador de Android: ctrl+M (Linux y Windows), comando+M (Mac)
  - Emulador de iOS: comando+D
  - Dispositivos físicos: gesto de sacudida
- **Recarga de aplicaciones React Native**
  - Cualquier dispositivo: abrir menú de desarrollo y seleccionar opción 'Reload'
  - Emulador de Android: pulsar dos veces la tecla R
  - Emulador de iOS: comando+R
- **Recarga automática de aplicaciones React Native**
  - Abrir menú de desarrollo y seleccionar opción 'Enable Hot Reloading'

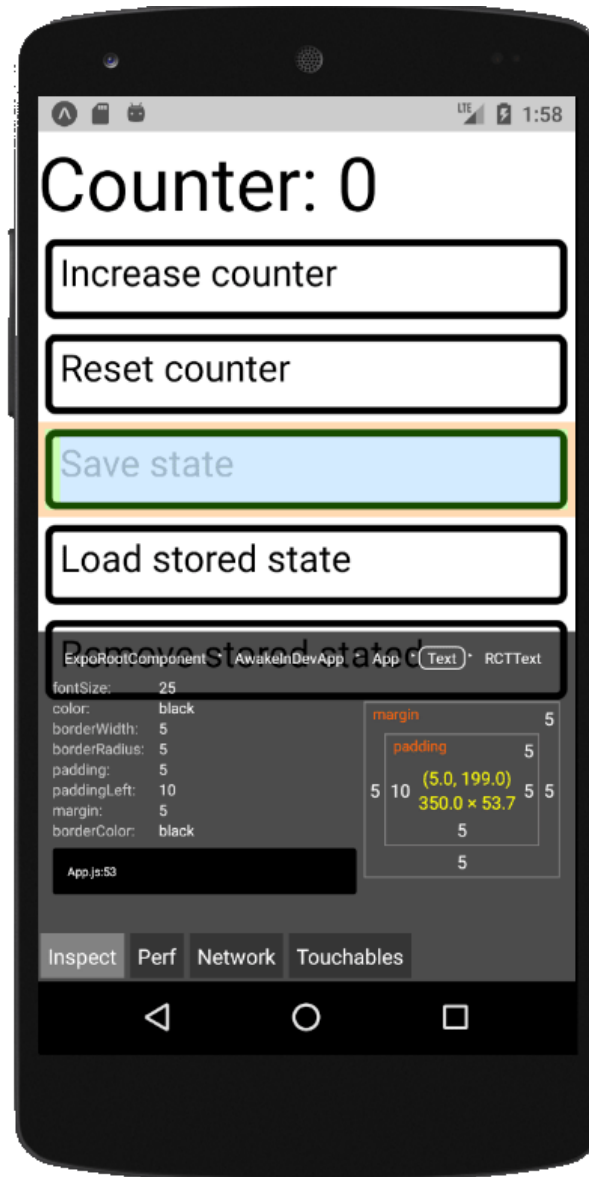
# Depuración: Menú de desarrollo

---

- Reload
- Debug JS Remotely
- Enable Live Reload
- Enable Hot Reloading
- Toggle Inspector
- Show Perf Monitor



# Depuración: Inspector

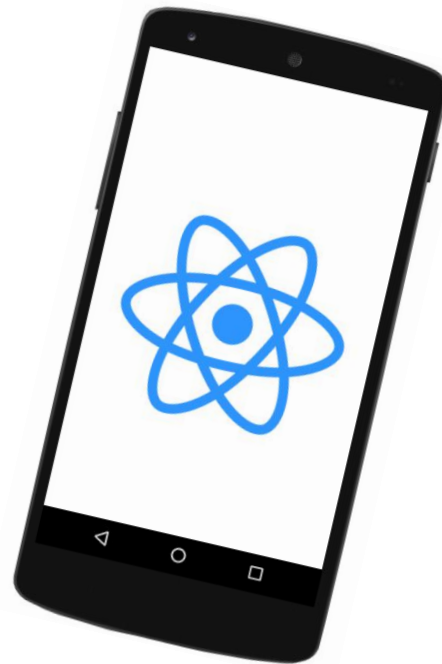


# Depuración

---

- **Visionado de logs**
  - **Create React Native App:** podemos ver los logs en el terminal donde hayamos lanzado el packager
  - **React Native CLI:**
    - react-native log-android (Android)
    - react-native log-ios (iOS)
- Herramientas para depurar código JavaScript
  - Chrome Developer Tools
  - React Developer Tools
- Más información:  
<https://facebook.github.io/react-native/docs/debugging.html>

# React Native



## Documentos y recursos adicionales

# React Native

---

- Documentación oficial

[\*http://facebook.github.io/react-native/docs\*](http://facebook.github.io/react-native/docs)

- Implementación de módulos nativos
- Integración con aplicaciones nativas existentes

- Herramientas y entornos de desarrollo

[\*https://facebook.github.io/react-native/docs/more-resources.html\*](https://facebook.github.io/react-native/docs/more-resources.html)

[\*https://snack.expo.io\*](https://snack.expo.io)

- Ejemplos de aplicaciones React Native

[\*https://github.com/ReactNativeNews/React-Native-Apps\*](https://github.com/ReactNativeNews/React-Native-Apps)

- Libros y cursos

[\*http://www.reactnative.com/books\*](http://www.reactnative.com/books)

- Conferencias de React

2017: [\*https://www.youtube.com/playlist?list=PLb0IAmt7-GS3fZ46IGFirdqKTlxlws7e0\*](https://www.youtube.com/playlist?list=PLb0IAmt7-GS3fZ46IGFirdqKTlxlws7e0)

2016: [\*https://www.youtube.com/playlist?list=PLb0IAmt7-GS0M8Q95RIc2IOM6nc77q1IY\*](https://www.youtube.com/playlist?list=PLb0IAmt7-GS0M8Q95RIc2IOM6nc77q1IY)

- Stack Overflow

[\*http://stackoverflow.com/tags/react-native\*](http://stackoverflow.com/tags/react-native)

# ¿Preguntas?

Aldo Gordillo

[agordillo@dit.upm.es](mailto:agordillo@dit.upm.es)