### Instalación y configuración

Para el desarrollo de aplicaciones móviles requerimos tener instalado Android Studio o Xcode (IOS). Toda la información la encontramos en la web <https://facebook.github.io/react-native/docs/getting-started>

Además, se requiere de una versión superior a nodeJS 10 para emplear un cliente Expo que nos permite ver en tiempo real en nuestro dispositivo, mediante una app instalada (expo) la app generada o superior a nodeJS 8.3 si empleamos el cliente React Native Cli.

Nos solicitan instalar (siguiendo las instrucciones de la web) las librerías watchman y openjdk. Una vez realizamos todos los procesos de preparación de entorno pasamos a crear nuestra aplicación, comenzando por instalar el cliente de aplicaciones nativas de React:

npm install -g react-native-cli

Existen diferentes caminos para combinar React Native con typescript, seguir estos pasos, en el momento que se escribe este documento es la manera más fácil: <https://facebook.github.io/react-native/blog/2018/05/07/using-typescript-with-react-native>

Seguimos los siguientes pasos para crear nuestro proyecto empleando Typescript.

Crear el proyecto:

react-native init MyAwesomeProject

Entrar en el proyecto:

cd MyAwesomeProject

Instalar typescript:

npm install typescript

Instalar la librería que transforma a typescript:

npm install react-native-typescript-transformer

Crear el archivo tsconfig:

tsc --init --pretty --jsx react

Ojo, descomentar la siguiente línea:

"allowSyntheticDefaultImports": true, /\* Allow default imports from modules with no default export. This does not affect code emit, just typechecking. \*/

Crear el archivo de configuración en la carpeta raíz del proyecto (rn-cli.conf) de transformer e insertar el siguiente código:

module.exports = {

getTransformModulePath() {

return require.resolve('react-native-typescript-transformer');

},

getSourceExts() {

return ['ts', 'tsx'];

},

};

Añadir los tipos de react y react native para typescript:

npm install @types/react @types/react-native

npm install @types/react-test-renderer

Ahora renombramos los archivos App.js por App.tsx, pero index.js debe mantenerse con extensión js. A partir, de aquí todos los archivos serán ts o tsx si incorporan JSX.

Por simplicidad, vamos a eliminar la carpeta \_\_test\_\_ que contiene los archivos de pruebas unitarias. No obstante, en la web se explica cómo instalar la compatibilidad.

Por último, actualizamos el código de App.tsx en las siguientes líneas:

Eliminamos:

import React, { Component } from 'react';

Añadimos:

import React from 'react'

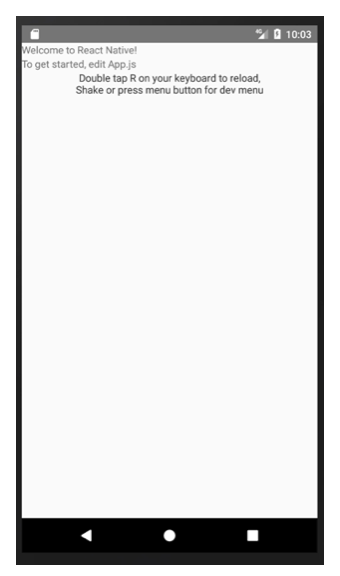
import { Component } from 'react';

(Nota, sin realizar este último cambio también funciona con la versión actual)

En este documento lo vamos a probar con la plataforma Android, por tanto, nos aseguramos de haber creado un AVD y tenerlo arrancado. Posteriormente ejecutamos:

react-native run-android

Tras esperar un largo tiempo con paciencia obtendremos la siguiente salida en el emulador:



En caso de querer emularlo en un dispositivo físico debemos conectarlo por USB y habilitar el modo depurador.

Ahora vamos a realizar unas modificaciones para generar un proyecto real que use componentes nativos de un dispositivo Android. Concretamente, será una aplicación que permitirá compartir o imágenes de tus álbumes de fotos o de una foto que se realice desde la app.

### Desarrollo de la aplicación

Partimos del proyecto creado en el apartado anterior. Tendremos el componente principal App.tsx y uno personalizado llamado CameraReact.tsx que nos permitirá inyectar todas nuestras funcionalidades.

Para acceder a los recursos de nuestro dispositivo Android vamos a indicar en el archivo /android/app/src/main/AndroidManifest.xml los siguientes permisos:

<uses-permission android:name=“android.permission.READ\_EXTERNAL\_STORAGE”/>

<uses-permission android:name="android.permission.VIBRATE"/>

<uses-permission android:name="android.permission.CAMERA" />

En orden, su uso será:

* Acceso al almacenamiento para leer las fotos de la galería.
* Vibrar para indicar que se está cargando la galería.
* Acceder a la cámara

Para realizar estas funcionalidades requerimos de la instalación de algunas librerías externas:

* React-native-fs: para el acceso al almacenamiento. Lo empleamos para leer una foto seleccionada y realizar su conversión a base64 y, así, poder enviarla como mensaje. Los instalamos con el siguiente comando:

npm install react-native-fs

react-native link react-native-fs //if error on loading the component

Hay que recordar que cada vez que se realiza una instalación de un nuevo paquete, es necesario para la ejecución de la consola que se abre al ejecutar:

react-native run-android

Para ejecutarla de nuevo y que cargue las nuevas dependencias.

* React-native-share: para compartir mensajes y contenido multimedia mediante todos los medios que cuente el sistema del dispositivo. Para instalar ejecutaremos:

npm install react-native-share

react-native link react-native-share //if error on loading the component

* React-native-camera: para el acceso a las cámaras del dispositivo. Permite realizar fotos o vídeos. Para instalar ejecutaremos:

npm install react-native-camera

react-native link react-native-camera //if error on loading the component

Para ejecutar la librería ha sido necesario incluir la siguiente línea en el archivo /android/app/build.gradle:

android {

...

defaultConfig {

...

missingDimensionStrategy 'react-native-camera', 'general' <-- insert this line

}

}

#### Código fuente

App.tsx:

import React,{ Component } from 'react'

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

import CameraReact from './CameraReact';

const instructions = Platform.select({

ios: 'Press Cmd+R to reload,\n' + 'Cmd+D or shake for dev menu',

android:

'Double tap R on your keyboard to reload,\n' +

'Shake or press menu button for dev menu',

});

type Props = {};

export default class App extends Component<Props> {

render() {

return (

<View>

<CameraReact frontCamera={false}></CameraReact>

</View>

);

}

}

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#F5FCFF',

},

welcome: {

fontSize: 20,

textAlign: 'center',

margin: 10,

},

instructions: {

textAlign: 'center',

color: '#333333',

marginBottom: 5,

},

});

CameraReact.tsx:

import React from "react";

import {

Text,

Button,

StyleSheet,

View,

ScrollView,

CameraRoll,

Image,

Vibration,

PermissionsAndroid,

TouchableOpacity,

GetPhotosParamType

} from "react-native";

import \* as RNFS from "react-native-fs";

import Share from "react-native-share";

import { RNCamera } from "react-native-camera";

//defining props accepted from parent

export interface Props {

frontCamera: boolean;

}

//defining variable state <- any change here makes a new render

interface State {

npics: number;

photos: any;

camera: any;

lastCursor: string;

}

export default class CameraReact extends React.Component<Props, State> {

//References of a html or component element, in this case to Camera Component

camera = React.createRef<RNCamera>();

cameraNode: RNCamera;

constructor(props: Props) {

super(props); //mandatory

this.state = { npics: 0, photos: [], camera: styles.hide,lastCursor:null }; //default value

this.clickOnImage = this.clickOnImage.bind(this); //bind this to this methods

this.takePicture = this.takePicture.bind(this);

}

//This is executed after render, so we can get the reference to the camera component (in HTML render)

componentDidMount() {

if (this.camera.current) {

this.cameraNode = this.camera.current;

}

}

//General method to request Permissions with custom message

async requestPermission(perm: any, tit, msg) {

try {

const granted = await PermissionsAndroid.request(perm, {

title: tit,

message: msg,

buttonNeutral: "Ask Me Later",

buttonNegative: "Cancel",

buttonPositive: "OK"

});

if (granted === PermissionsAndroid.RESULTS.GRANTED) {

//only android, it has to be updated to IOS

return true;

} else {

console.warn("Permission denied");

return false;

}

} catch (err) {

console.error(err);

return false;

}

}

/\*\*

\* Arrow definition, interesting.

\* This method access to the external storage to get pics from cameral roll.

\* It uses state.lastCursor to keep the last image loaded, in order to allow recall this

\* method a get another 10 pics more

\*/

private \_loadImages = async () => {

try {

Vibration.vibrate(1000);

let auth = await this.requestPermission(

PermissionsAndroid.PERMISSIONS.READ\_EXTERNAL\_STORAGE,

"Load Data",

"Let's see, this app needs to access to the external storage in order to get your pics, can it?"

);

if (auth) {

let props:GetPhotosParamType={

first: 10,

assetType: "Photos"

}

if(this.state.lastCursor){

props.after=this.state.lastCursor;

}

let pics = await CameraRoll.getPhotos(props);

this.setState({ photos: pics.edges.concat(this.state.photos) }); //concat new pics before last ones

this.setState({lastCursor:pics.page\_info.end\_cursor}); //next reload starts from the last

return true;

} else {

return false;

}

} catch (err) {

console.error(err);

return false;

}

};

//This method use styles to hide or show the camera. It is called from a button

private \_showCamera = () => {

this.state.camera==styles.container?

this.setState({ camera: styles.hide }):

this.setState({ camera: styles.container })

};

/\*\*

\* this method takes a pic from the camera and convert it automatically to base64.

\* Then it hides the camera and call share method (clickOnImage)

\*/

async takePicture() {

try {

if (this.cameraNode) {

const hasCameraPermissions = await this.requestPermission(

PermissionsAndroid.PERMISSIONS.CAMERA,

"Take Pic",

"Let's take some pics, can i?"

);

if (!hasCameraPermissions) return;

const options = { quality: 0.5, base64: true, fixOrientation: true };

let data = await this.cameraNode.takePictureAsync(options);

this.setState({ camera: styles.hide });

await this.clickOnImage(data.base64, true); //d.base has directyle de base64 code

}

} catch (err) {

console.error(err);

return false;

}

}

/\*\*

\*

\* @param x uri of pic to share if base 64 is false or null. Base64 code if base 64 is true

\* @param base64 indicates if x is an uri or base64 code directly

\*/

async clickOnImage(x: string, base64?: boolean) {

try {

//if it's uri, it gets the image from storage and convert it to base64

if (!base64) {

x = await RNFS.readFile(x, "base64");

}

let shareOptions = {

title: "My React Sample",

dialogTitle: "My React Sample",

subject: "My React Sample",

message: "This is a test",

url: `data:image/png;base64,` + x

};

//Native component to share info

Share.open(shareOptions);

return true;

} catch (err) {

console.error(err);

return false;

}

}

render() {

let whichCamera=RNCamera.Constants.Type.back;

if(this.props.frontCamera){

whichCamera=RNCamera.Constants.Type.front;

}

return (

<View>

<View style={this.state.camera}>

<RNCamera

ref={this.camera}

style={styles.preview}

type={whichCamera}

/\*flashMode={RNCamera.Constants.FlashMode.on}\*/

androidCameraPermissionOptions={{

title: "Permission to use camera",

message: "We need your permission to use your camera",

buttonPositive: "Ok",

buttonNegative: "Cancel"

}}

androidRecordAudioPermissionOptions={{

title: "Permission to use audio recording",

message: "We need your permission to use your audio",

buttonPositive: "Ok",

buttonNegative: "Cancel"

}}

/>

<View>

<TouchableOpacity

onPress={this.takePicture.bind(this)}

style={styles.capture}

>

<Text style={{ fontSize: 14 }}> SNAP </Text>

</TouchableOpacity>

</View>

</View>

<Button

color="#841500"

title="Take a pic"

onPress={() => this.\_showCamera()}

/>

<Button

color="#841584"

title="Load Images"

onPress={() => this.\_loadImages()}

/>

<ScrollView contentContainerStyle={styles.root}>

{this.state.photos.map((p, i) => {

return (

<TouchableOpacity

key={i}

activeOpacity={0.5}

onPress={() => this.clickOnImage(p.node.image.uri)}

>

<Image

key={i}

style={{

width: 300,

height: 100,

margin: 10,

borderRadius: 5,

borderColor: "#AAA",

borderWidth: 1

}}

source={{ uri: p.node.image.uri }}

/>

</TouchableOpacity>

);

})}

</ScrollView>

</View>

);

}

}

// custom styles

const styles = StyleSheet.create({

root: {

justifyContent: "center",

alignItems: "center"

},

images: {

width: 300,

height: 100,

margin: 10

},

container: {

backgroundColor: "black",

display: "flex"

},

preview: {

justifyContent: "flex-end",

alignItems: "center",

height: 400

},

hide: {

height: 0,

display: "none"

},

capture: {

backgroundColor: "#ff0",

borderRadius: 5,

padding: 15,

paddingHorizontal: 20,

alignSelf: "center",

margin: 20

}

});