|  |  |
| --- | --- |
| Tutorial | 85 |

1. El último elemento de la sección de contactos parece que se trata de un contacto, pero en realidad no lo es. Es otra vista envuelto conTouchableOpacity labrado a fluir sin problemas con las otras vistas de contacto real.
2. El botón “Guardar” no es en realidad parte de este punto de vista. Se representa en la barra de navegación (que hemos revisado anteriormente), pero nosotros asegúrese de recibir eventos de ella dentro del cuerpo de la vista principal detalle.
3. Lo mismo aquí, la “vuelta” funcionalidad de este punto de vista no es realmente en él. Se encuentra en la barra de navegación.
4. Lo mismo de nuevo. El título se deriva de nuestra asignador de ruta, que hemos cubierto anteriormente. Es agradable ver cómo encaja todo!

Vamos a empezar la disección de la vista de detalles, esta vez vamos a empezar directamente con la hacer métodos:

* hacer: **función**() {
* **dejar** ít= **esta**.state.item
* **dejar** contactsViews=\_.map (item.contacts, (v, k)=>{
* **regreso**(
* <Avatar clave={K}

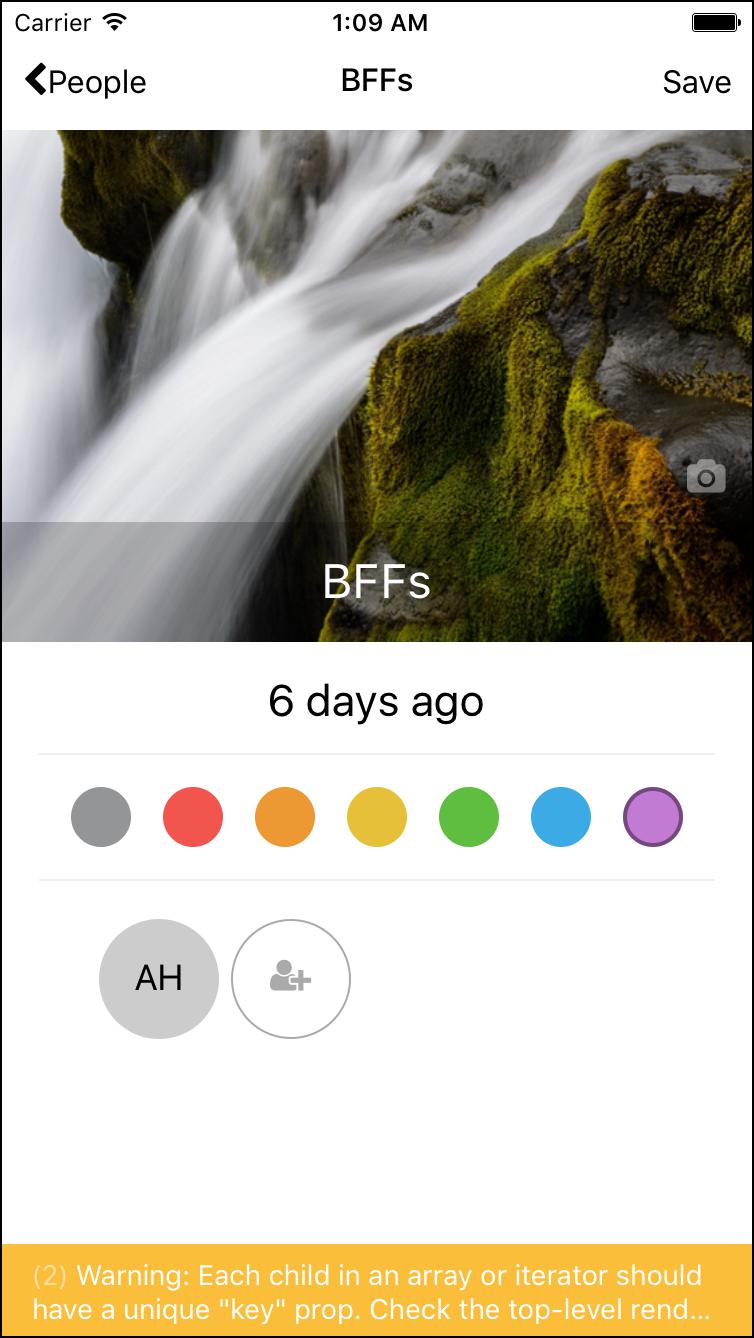
|  |  |
| --- | --- |
| 6 | imagen={} V.thumbnailPath |
| 7 | estilo={} Styles.avatar |
| 8 | estilo de texto={} Styles.avatarText |
| 9 | firstText={} V.firstName |
| 10 | secondText={} V.lastName /> |

1. )
2. })

Estamos tirando de la ítdirectamente de nuestro estado, que se modificó y se mantuvo allí y en un principio recibimos de nuestros puntales cuando alguien navega en la vista Detalle. Siguiente estamos construyendo la sección de contactos; esta sección es simplemente mapeoitem.contacts en una lista de nuestra costumbre Avatarcomponentes. Es importante proporcionar un únicollave Al igual que con cualquier colección de puntos de vista ya que esto ayuda con Reaccionar de [reconciliación](https://facebook.github.io/react/docs/multiple-components.html)[51](#page1). Si se ha perdido esto, obtendrá una advertencia amarilla amable desde Reaccionar nativo directamente en su aplicación:

1. <https://facebook.github.io/react/docs/multiple-components.html>

|  |  |
| --- | --- |
| Tutorial | 86 |



**La caja de Yello**

Pruebe a ejecutar el proyecto de ejemplo y la eliminación de las propiedades clave. Entonces, cuando llegue la caja amarilla, haga clic en él para obtener más información.

*PD* también estamos utilizando la nueva ES6 [dejar](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/let)[52](#page2)declaración de variables para una mejor semántica de alcance.

* **dejar** tagsViews=\_.map (etiquetas, (v, k)=>{
* **dejar** selectedStyle=(k==item.tag)?styles.tagSelected:{}
* **regreso**(
* <clave TouchableOpacity=estilo {k}={[Styles.tag, selectedStyle, {backgroun\ 5 dColor: v.color}]} onPress={()=> **esta**.onTagSelected (k)} />
* )
* })

8

9**dejar** marcador de posición= **nulo**

1. <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/let>

|  |  |
| --- | --- |
| Tutorial | 87 |

1. **Si**(!item.image) {
2. marcador de posición = (
3. <Nombre icónico="Foto-O" tamaño={84} estilo={} Styles.placeHolder/>
4. )
5. }

Casi lo mismo que con la sección de avatares, la sección de etiquetas hace una recopilación de los selectores de etiquetas y esta es la forma en que las hacen, teniendo en cuenta que sólo la etiqueta seleccionada en ese momento (item.tag) Debe ser representada como seleccionado. Simplemente pasa por el estilo seleccionado a nuestraTouchableOpacitycomponente que sirve como el cuerpo de la etiqueta y el respondedor gesto táctil. Esta es también una gran oportunidad de mostrar Reaccionar composición de estilos del nativo; tenemos un estilo general de etiquetasstyles.tag, Un posible estilo seleccionado selectedStyleY un color de fondo específico decidido por el valor de la etiqueta v.color. Como siempre, cualquier interacción con un componente generado es simplemente aplazado al componente de “inteligente” sostiene.

A continuación decidimos si no hay imagen proporcionada con nuestra ít, De ser cierto, construimos un marcador de posición para cubrir la falta de imagen de la portada. Esto sería un gran icono, cubriendo el espacio de la imagen de la portada. En este caso, estamos usando-reaccionar-vector-iconos nativos que se puede obtener [aquí](https://www.npmjs.com/package/react-native-vector-icons)[53](#page3). La idea es hacer las maletas cada icono de vector que es libremente disponibles, tales como Fontawesome y jónico marco de Ionicons, así como iconos de la Fundación y mucho más. Esto crea una zona de juegos increíble y guarda los recursos en caso de que desee invertir en un conjunto de iconos o contratar a un diseñador. Para utilizar estos iconos que sólo tiene que proporcionar unanombre y tamañoapuntalar. El estilo habitual mantiene así:

* <Nombre icónico="Foto-O" tamaño={84} estilo={} Styles.placeHolder/>

Para instalar cualquier módulo 3 ª parte, como con cualquier proyecto basado en Node.js que utilizaría NPM:

* $ NPM instalar reaccionar nativo-vector-iconos --save

Sin embargo, ya que este módulo específico implica algún trabajo manual inusual, como la inclusión de los archivos de recursos de fuentes reales en cada uno de sus iOS y plataformas Android, podría ser útil para que usted esté familiarizado con rnpm, Que es una herramienta complementaria para NPM, Sólo para Reaccionar nativo. Hacer esto:

* $ NPM instalar rnpm -g

Y luego, después de instalar React módulo nativo, puede ejecutar:

1. <https://www.npmjs.com/package/react-native-vector-icons>

|  |  |
| --- | --- |
| Tutorial | 88 |

* $ Enlace rnpm

Dentro de su directorio del proyecto. rnpmbuscará módulos con un requerimiento nativa como un binario para vincular en contra de su aplicación, o un recurso de incluir dentro de cualquiera de sus proyectos de iOS o Android y hacer que el trabajo para usted. Debe tener en cuenta, sin embargo, que algunas veces se echa de menos e incluye un ejemplo callejero proyecta que alguien ha puesto junto con el proyecto principal, o los archivos que realmente no necesita, así que asegúrese de echar un vistazo a su proyecto nativa (IOS o androide) debe su construcción repente funcionar mal. Es por esta razón siempre prefiero hacer la conexión manualmente y la lectura de laREADME antes de la mano.

El siguiente, que necesitamos para componer todo juntos:

* **dejar** ImageComponent=item.image?Imagen:Ver

2**regreso** (

3 <ScrollView automaticallyAdjustContentInsets={**falso**} contentContainerStyl\ 4 mi={} Styles.container>

5 <fuente ImageComponent={(Item.image) H.documentsImage} estilo={styles.im\ 6 años}>

* {Marcador de posición}
* <Ver estilo={} Styles.toolBar>
* <estilo TouchableOpacity={} Styles.pickImageIcon onPress={**esta**.onPic\

10 kImage}>

11 <Nombre icónico='cámara' estilo={} Styles.pickImageIconText tamaño={18} />

1. </ TouchableOpacity>
2. </ Ver>

14

1. <Entrada de texto
2. estilo={} Styles.input
3. onChangeText={**esta**.onTitleChanged}
4. valor={} Item.title
5. />
6. </ ImageComponent>

21

1. <Estilo de texto={} Styles.date>
2. {H.capitalize (H.fromNowInWords (item.date))}
3. </ Texto>

25

1. <Ver estilo={} Styles.tagHolder>
2. {} TagsViews
3. </ Ver>

29

1. <Ver estilo={} Styles.avatarHolder>
2. {} ContactsViews
3. <estilo TouchableOpacity={} Styles.avatarAdd onPress={**esta**.onPeoplePres\

|  |  |
| --- | --- |
| Tutorial | 89 |

1. SED}>
2. <icono de estilo={} Styles.avatarAddText nombre="User-plus" tamaño={18}/>
3. </ TouchableOpacity>
4. </ Ver>
5. </ ScrollView>
6. )
7. }

En primer lugar, se decide si el componente principal de la cubierta es un vivo Imagen cuando tenemos una imagen para mostrar o un fracaso Vercuando esa propiedad de imagen es nulo; es lo que llamamosImageComponent.

A continuación, nos envuelva todo con ScrollView. El nativo ReaccionarScrollView es bastante flexible, y usted debe tomar un momento para mirar el [Ver documentos de desplazamiento](https://facebook.github.io/react-native/docs/scrollview.html)[54](#page5)antes de continuar. Aquí básicamente compensar nuestra barra de navegación, y especificar el estilo de contenedor que es generalmente donde se especificaFlex: 1 (Tenga en cuenta que esto no es lo habitual estilo apuntalar).

Componemos la ImageComponentcontenido. En este caso, será la imagen de la portada, el título editable y el botón de la cámara. También hacemos un poco de estilo Flex para este componente, echar un vistazo a esta hoja de estilo, sacamos destyles.js:

* imagen:{
* flexDirection:'columna',
* justifyContent:'Flex-final',
* alignSelf:'tramo',
* altura: 320, 6 },

Por lo que estamos diciendo contenido debe fluir como una columna, y también todo lo que desee adherirse a la parte inferior con flex-end. Estamos estirando la imagen de la portada, y dándole una altura fija.

A continuación se muestran el la sección de etiquetas de tiempo y:

* <Estilo de texto={} Styles.date>
* {H.capitalize (H.fromNowInWords (item.date))} 3 </ Texto>

4

5 <Ver estilo={} Styles.tagHolder>

* {} TagsViews 7 </ Ver>

Dónde MARIDOes sólo una forma abreviada de nuestros ayudantes, y el objetivo es hacer que la fecha en un texto de estilo “hace tiempo”. A continuación se presenta el diseño de las etiquetas, y aquí, así que estamos haciendo algunos maquetación Flex:

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

|  |  |
| --- | --- |
| Tutorial | 90 |

* tagHolder:{
* marginTop: 8,
* marginBottom: 8,
* relleno: 8,
* flexDirection:'fila',
* borderTopWidth: 1,
* borderBottomWidth: 1,
* color del borde: '#eee', 9 },

Como se puede ver que estamos dando instrucciones al motor de diseño Flex a fluir el contenido como una fila. Por último, nuestra sección de contactos se representa como la siguiente:

* <Ver estilo={} Styles.avatarHolder>
* {} ContactsViews
* <estilo TouchableOpacity={} Styles.avatarAdd onPress={**esta**.onPeoplePressed}>

4<icono de estilo={} Styles.avatarAddText nombre="User-plus" tamaño={18}/>

* </ TouchableOpacity> 6 </ Ver>

Estamos volcar el contactViews variables construimos anteriormente que contiene una lista de los contactos Avatarcomponentes y, a continuación, haciendo nuestro botón especial “añadir contacto” con el mismo estilo de los avatares que había antes de él. Aquí, también, es de suponer, estamos haciendo uso de Flex, dejando que el flujo de contenido como una fila:

* avatarHolder:{
* flexDirection:'fila',
* FlexWrap:'envolver',

4anchura: 300,

* relleno: 8, 6 },

Eso es todo por hacer que la vista de detalle. Como era de esperar este punto de vista, siendo un detalle ver que es más pesado en la interfaz de usuario. El siguiente, vamos a echar un vistazo a la mecánica detrás de la vista, de cómo los flujos de datos y las interacciones; que en realidad es todo, excepto lahacer función.

Así es como comienza nuestro componente:

|  |  |
| --- | --- |
| Tutorial | 91 |

* **var** Detalle=R ({
* mixins: [Subscribable.Mixin],

3

4componentDidMount: **función**() {

5**esta**.addListenerOn (**esta**.getStore (). eventos,'cambio', **esta**.onStoreChanged)

6**esta**.addListenerOn (**esta**.props.navEvents,'salvar', **esta**.onSave)

* },

8

9onStoreChanged: **función**(evento){

1. **Si**(event.action=="actualizar") {
2. **Si**(event.subject.id== **esta**.state.item.id) {
3. **esta**.setState ({item:event.subject})
4. }
5. }
6. },

dieciséis

1. getStore: **función**() {
2. **devolver este**.props.store
3. },

Mezclamos la suscribiblesmixin, de modo que disfrutemos de una eliminación automática de los oyentes de manera segura cuando el componente está desmontaje. Ver más[aquí](https://github.com/facebook/react-native/blob/master/Libraries/Components/Subscribable.js)[55](#page7) sobre suscribible.

A continuación se registra abonados en nuestra tienda (convenientemente sacados de nuestros puntales), y nuestro navegador para el evento “salvar”. Una vez más, estas suscripciones convenientemente serán eliminados por nosotros cuando desmontar porque vamos a través de la suscribibleaddListenerOn función.

Cuando los cambios de la tienda, simplemente hacer una comprobación de validez de verificar que el elemento que cambia es la que estamos sosteniendo. Si es así, simplemente establecer el estado y reaccionar se encarga del resto.

Vamos a ver cómo se manejan los eventos. En la lista, tenemos que manejar guardar, eliminar, cambiar el título, recoger la imagen, y la etiqueta seleccionada.

* OnSave: **función**() {
* **dejar** ít= **esta**.state.item
* **esta**.getStore (). actualización (elemento)
* **esta**.props.navigator.pop ()

5},

Este evento viene de nuestra navEventsdespachador de eventos que vimos durante nuestra discusión del código de arranque y de navegación anterior. Aquí simplemente obtenemos nuestra tienda y le diremos que actualizar el elemento. Más encima, una vez salvados queremos volver a la vista principal, o que jamás vista que nos llama - por lo que sacar la pila de navegación, tan simple como eso!

1. <https://github.com/facebook/react-native/blob/master/Libraries/Components/Subscribable.js>

|  |  |
| --- | --- |
| Tutorial | 92 |

* onTitleChanged: **función**(texto){

2**dejar** ít= **esta**.state.item

3item.title = texto

4**esta**.setState ({item:ít})

5},

Se trata de un controlador simple para cuando el título cambia y se ve obligada a nuestra Texto componente. Nosotros sólo modifican el estado, y cuando el uso golpea “Guardar” todo lo que vamos a comprometernos a nuestra tienda.

El siguiente, vamos a ver cómo estamos recogiendo una imagen. Para ello utilizamos un módulo tercera parte llamadareaccionará-imagen-selector nativa. Para instalar y vincularlo que haremos:

* $ NPM instalar reaccionar-imagen-nativa-selector --save && enlace rnpm

Ahora para el controlador:

* onPickImage: function () {
* UIImagePickerManager.showImagePicker (pickerOptions, (respuesta) => {

3console.log ( `DETALLE: imagen captada, la respuesta`, respuesta)

4if (! response.didCancel) {

5this.state.item.image = {URI: response.uri}

* this.setState ({
* material: this.state.item
* })
* }

1. })
2. },

Dado que este es un módulo nativo, no es realmente un componente de interfaz de usuario se incluyen, pero algo que se parece a un servicio del sistema. UIImagePickerManager es ese módulo y nos dicen que para mostrar la interfaz utilizando nuestro selector pickerOptions (echar un vistazo [aquí](https://github.com/marcshilling/react-native-image-picker)[56](#page8) para estas opciones) y obtenemos una respuesta adecuada sólo después de haber escogido el uso de una imagen (o cancelada). Si el usuario no canceló la enchufamosresponse.uri en nuestra propiedad imagen dentro ít, Que más adelante vamos a inyectar directamente en un Reaccionar nativo Imagencomponente. A continuación, forzar una actualización consetstate.

1. <https://github.com/marcshilling/react-native-image-picker>

|  |  |
| --- | --- |
| Tutorial | 93 |

* onTagSelected:**función**(etiqueta){

2**dejar** ít= **esta**.state.item

3item.tag = etiqueta

* **esta**.setState ({item:ít})
* },
* onPeoplePressed:**función**() {
* **esta**.getStore (). actualización (**esta**.state.item)
* **esta**.props.navigator.push ({
* carné de identidad: 'gente',

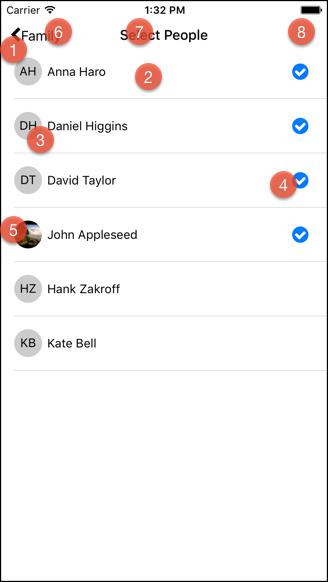
1. título: "Seleccionar a las personas",
2. accesorios: {
3. Identificación del artículo: **esta**.state.item.id,
4. },
5. })
6. },

Tanto de los manipuladores restantes se ocupan de la selección contactos o las etiquetas. Sin embargo, cuando se selecciona un contacto (onPeoplePressed) Que navegar a una pantalla completamente diferente - la pantalla Contactos, que enumerará los contactos en el teléfono del usuario. Si usted escudriña, se puede decir que laempujar función empuja en realidad una rutaen el navegador, que se cansa de nuevo a nuestro código de asignación de ruta y navegación. Correcta o no, desde el punto de vista del producto, se guarda los cambios realizados por el usuario hasta el momento antes de navegar a la nueva pantalla.

**La gente (Contactos) de la pantalla**

El último “componente inteligente” que vamos a diseccionar es la pantalla de contactos, donde un usuario es capaz de seleccionar los contactos que deberían pertenecer a un grupo determinado. Vamos a echar un vistazo ahora.

|  |  |
| --- | --- |
| Tutorial | 94 |



**contactos de la pantalla**

1. Al igual que antes con nuestra opinión Maestro, esta es una Vista de la lista
2. Y otra vez, cada Vista de la lista ejerce una célula prototipo, y este es nuestro celular
3. Dentro de cada célula, tenemos nuestro Avatar componente nuevo, jugando un papel diferente, pero sigue siendo el mismo componente reutilizable
4. Generamos una marca de verificación con nuestra Icono componente para elementos que están seleccionados, toda la lista se actualiza y reconstruida de manera que los contactos seleccionados están siempre en la parte superior.
5. Los avatares pueden ser titulares de una imagen, o si falta, mostrar una primera carta de cada parte del nombre del contacto (nombre y apellido).
6. La barra de navegación puede ir de nuevo al grupo venimos, esto se hace automáticamente para nosotros por el mapeo de navegación
7. Lo mismo para el título
8. Y hay un botón aquí, no tenemos eran más ir

|  |  |
| --- | --- |
| Tutorial | 95 |

Vamos a empezar revisando el código de la pantalla de contactos, esto será de nuevo un patrón maestro-detalle, así que vamos a comenzar con la parte principal. Vamos a hacer esto al revés, así, a partir de lahacer función.

* hacer: **función**() {
* **regreso** (
* <estilo ListView={{acolchado superior: 50, flexionar: 1}}
* fuente de datos={**esta**.state.contacts}
* renderRow={**esta**.renderRow}
* />
* );

8}

Nada nuevo aquí, nuestra fuente de datos es ahora el más propiamente llamado contactos (Si desea una revisión de Vista de la listay fuentes de datos, se refieren de nuevo a la vista Patrón de disección). Continuemos en nuestrorenderRow función.

* renderRow:**función**(fila){
* **regreso** (
* <clave de la célula={**esta**.state.item.id} artículo={**esta**.state.item} contactos={Fila} ONP\ 4 reso={()=> **esta**.didSelectRow (fila)}/>
* )

6},

Buenas noticias aquí, esto también tiene el mismo aspecto. Un patrón de detalle maestro sería probablemente siempre el mismo aspecto, como este. Usted está pensando lo que estoy pensando? Sí. Se podría hacer unaMaterComponent que incorpora este patrón, dándole sólo dos cosas:

* *// Imaginario MasterComponent*
* **clase** Gente **se extiende** MasterComponent {
* didMountListView () {

4**regreso** {

5datos: contactos,

6fila: (ít)=> <Celda .../>

* }
* }
* *// Ahora acaba de manejar eventos regulares componentes inteligentes.*

10 }

Esto tiene sentido, ya que iOS ha incrustado un patrón de este tipo con [UITableViewController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewController_Class/)[57](#page11) como así, ya que esta infraestructura no hace realmente existo para Reaccionar nativo sin embargo, voy a dejar esto como preparación :-).

Vamos a ver lo que pasa cuando montamos este punto de vista, en concreto, necesitamos a buscar contactos.

1. <https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewController_Class/>

|  |  |
| --- | --- |
| Tutorial | 96 |

* componentDidMount: **función**() {
* **esta**.addListenerOn (**esta**.getStore (). eventos,'cambio', **esta**.onStoreChanged)

3

4Contacts.getAll ((err, contactos) => {

* console.log ("Contactos", Contactos)

6\_.each (contactos, (contacto)=>{

7Nombre de contacto = emo (contact.givenName || Nombre de contacto || "")

8contact.lastName = emo (contact.familyName || contact.lastName || "")

* })

1. **Si**(errar&&err.type==='Permiso denegado') {
2. console.log ("La gente: no hay permisos contactos")
3. }
4. **más**{
5. contacts.sort (**esta**.sortBySelectionAndName)
6. **esta**.setState ({contactsArray:contactos, contactos: **esta**.state.contacts\
7. .cloneWithRows (contactos)})
8. }
9. })

19

1. },

En primer lugar, añadimos un oyente en la tienda. Recuerde que la tienda se han aprobado para este componente de nuestra pila de navegación. A continuación, utilizamos elcontactosmódulo, que es un módulo NPM 3 ª parte que utilizamos para obtener una bodega de instalación de los contactos de cada plataforma. Es posible imaginar que no sólo iOS y Android no comparten la misma API para recuperar los contactos, pero el modelo de datos y las cosas de nombres pueden ser diferentes también. Aquí es donde-reaccionar-contactos nativosllena el vacío. Para instalarlo:

* $ NPM-instalar contactos reaccionar nativos --save && enlace rnpm

Ahora usted debe conseguir a una API que se parece [esta](https://github.com/rt2zz/react-native-contacts)[58](#page12) en el código común de JavaScript. Si usted se pregunta cómo se realiza esta magia, cubrimos cómo hacer módulos nativos, componentes de interfaz nativos y más en este libro (y en el momento de la escritura, que podría ser el único lugar para leer sobre él con excepción de los documentos oficiales , que por desgracia proporcionan una cobertura parcial).

Cambiando de tema, una obtener todaObtiene todos los contactos del dispositivo, y reparte los diccionarios Javascript ingeniosas con campos llenos sobre cada contacto. Excluimos emoji porque nos gustaría mantenerlo negocio (puede dejar como está y mantener el emoji), utilizando elemoji-tirabiblioteca de NPM, y eso es una librería Javascript completamente claro que funciona fuera de la caja con Reaccionar nativo. Después de la fijación de los contactos, clasificamos ellos y ponerlos a nuestro estado.

Lo siguiente que manejamos onStoreChangedmediante la actualización de nuestros contactos en caso de que el propio grupo ha cambiado, pero esto es sólo una salvaguardia ya que este escenario no existe en realidad. También nos ocupamos de la selección de filas de este modo:

1. <https://github.com/rt2zz/react-native-contacts>

|  |  |
| --- | --- |
| Tutorial | 97 |

* didSelectRow:**función**(fila){

2**var** ít= **esta**.state.item

3**var** rowid=row.recordID

* **Si**(Item.contacts [rowid]) {
* **borrar** item.contacts [rowid]
* }**más** {
* item.contacts [row.recordID] = fila
* }
* **esta**.getStore (). actualización (elemento)

10},

Esto significa básicamente que sea colocado el DI de ese contacto en nuestro grupo o eliminarlo por completo. De cualquier manera empujamos hacia atrás nuestros cambios a nuestra tienda.

Vamos a pasar a la celda de lista. Se da la circunstancia de que nuestro celular es una[funcional](https://facebook.github.io/react/blog/2015/10/07/react-v0.14.html#stateless-functional-components) [componente](https://facebook.github.io/react/blog/2015/10/07/react-v0.14.html#stateless-functional-components)[59](#page13). Me encanta componentes funcionales (o “puros”) porque hacen una trampa del éxito; te obligan a la manera correcta de hacer las cosas, sin efectos secundarios, lo que lleva a un peso más ligero y más componentes componibles.

* module.exports = (accesorios) =>{

2**dejar** icono= **nulo**

3**dejar** contacto=props.contact

* **Si**(Props.item.contacts [contact.recordID]) {
* icono = <icono de estilo={} Styles.check nombre='Check-círculo' tamaño={22} />
* }

7

8**dejar** nombre de pila=Nombre de contacto

9**dejar** apellido=contact.lastName

1. **dejar** avatar= <Avatar

|  |  |
| --- | --- |
| 11 | firstText={nombre de pila} |
| 12 | secondText={apellido} |
| 13 | imagen={} Contact.thumbnailPath |
| 14 | estilo de texto={} Styles.avatarText |
| 15 | estilo={} Styles.avatar |
| dieciséis | /> |
| 17 |  |

1. **regreso** (
2. <ListItem onPress={} Props.onPress>
3. <Ver estilo={} Styles.holder>
4. <Ver estilo={} Styles.avatarHolder>

22 {avatar}

1. </ Ver>
2. <Ver estilo={} Styles.content>

25 <Estilo de texto={} Styles.text>{nombre Apellido}</ Texto>

1. </ Ver>
2. <https://facebook.github.io/react/blog/2015/10/07/react-v0.14.html#stateless-functional-components>

|  |  |
| --- | --- |
| Tutorial | 98 |

1. {icono}
2. </ Ver>
3. </ ListItem>
4. )
5. }

En este punto de vista, estamos componiendo un elemento de la lista. Estamos empezando con la decisión - qué o qué no dibujar un signo de verificación? El propio signo de verificación se hace con nuestro preferido-reaccionar-vector-iconos nativos módulo, y el Iconocomponente. A continuación vamos a componer laAvatar con todo lo que necesita de nuestro contacto, y por último el ListItem sí, y recuerda, al ser un “punto de vista mudo” simplemente estamos disparando manipuladores que se pasaron a través de nuestros accesorios.

La completa nuestra revisión de la totalidad de las pantallas de aplicaciones. A continuación, usted puede haber notado que estamos usando componentes tales comoIcono y Avatartodo en muchas situaciones diferentes, que es grande, porque estamos reutilizando componentes. Uno de estos componentes,Avatares uno que construimos para nosotros mismos. Vamos a ver cómo asegurarse un componente reutilizable siguiente.

**La fabricación de componentes reutilizables**

Durante nuestra opinión, nos encontramos con Avatarvarias veces. Es ahora veces para ver cómo funciona, y lo que es el razonamiento detrás de hacer un componente reutilizable.

En primer lugar vamos a describir. El avatar:

* Rendir una imagen cuando existe, y aplicar estilos generales a sí mismo, y estilos específicos de imagen, si existe.
* Render una “imagen” sintético que está componiendo la primera letra y la última letra del nombre completo del contacto (que es bastante común en los teléfonos móviles), y luego aplicar estilos de texto específicos a que, así como los estilos de imagen de usuario general.

De esta definición se llega a tres conjuntos diferentes de estilos:

1. componente estilo
2. avatar estilo de texto
3. estilo de imagen de avatar

Tomaremos como éstos estilo (¡Sorpresa!), estilo de texto y imageStyle. vamos a tener que cablear cada uno de los dos últimos en la parte superior deestilo, Pero eso es fácil, porque ya vimos Reaccionar estilos nativos son fácilmente componibles.

Ahora que estamos llevado a cero en la lógica, vamos a echar un vistazo al código:

|  |  |
| --- | --- |
| Tutorial | 99 |

* **clase** Avatar **se extiende** React.Component {
* constructor (apoyos) {
* **súper**(accesorios)
* \_.mapa(Objeto.getOwnPropertyNames (Objeto.getPrototypeOf (**esta**)),
* (X)=>{

|  |  |
| --- | --- |
| 6 | **Si**(X.match (/ ^ \_\_ [az] /)) { |
| 7 | console.log (x) |
| 8 | **esta**[X]= **esta**[X] .bind (**esta**) |
| 9 | } |

1. })
2. }
3. \_\_signature (primera="",segundo="") {
4. **regreso** `**ps**(primero[0]||"?") .toUpperCase ()**ps**segundo&&second.length> 0? \
5. segundo[0] .toUpperCase () : ""**}**`
6. }
7. hacer(){
8. **var** avatar= **nulo**
9. **Si**(**esta**.props.image) {
10. avatar = <Fuente de imagen={{uri: **esta**.props.image}}

20 estilo={[Styles.image, **esta**.props.style, **esta**.props.imag\

1. eStyle]} />
2. }**más**{
3. avatar = (<Ver estilo={[Styles.view, **esta**.props.style]}>

|  |  |
| --- | --- |
| 24 | <Estilo de texto={**esta**.props.textStyle}> |
| 25 | {**esta**.\_\_firma(**esta**.props.firstText, **esta**.props.second\ |

1. Texto)}

|  |  |
| --- | --- |
| 27 | </ Texto> |
| 28 | </ Ver>) |

1. }
2. **regreso** avatar
3. }
4. }

En primer lugar, hay un super extraño bloque de código en el constructor no está allí? Este es un código genérico que puede utilizar para enlazar automáticamente los métodos de claseesta. Con Javascript encontrará algunos términos son una especie de romperse, como el tratamiento de la esta. Mientras que la mayor parte de lo que se está fijo con ES6 y probablemente ES7, todavía hay un cierto desajuste de impedancia cuando se desea utilizar el nuevo estilo de React componentes (clase) y no el viejo estilo (React.createComponent); ver más[aquí](http://egorsmirnov.me/2015/08/16/react-and-es6-part3.html)[60](#page15). Así que aquí, estamos contando el componente de obligar cualquier método que comienza con\_\_, Por convención (la convención). Esto es sólo para mostrar cómo se puede ahorrar hasta en ese código repetitivo; usted puede adoptar ese enfoque si lo desea.

A continuación, componemos la vista con alternar entre nuestro Imagen o Texto basado

1. <http://egorsmirnov.me/2015/08/16/react-and-es6-part3.html>

|  |  |
| --- | --- |
| Tutorial | 100 |

ver. Tomar nota de la composición de estilo, en el caso deImagen:

* estilo={[Styles.image, **esta**.props.style, **esta**.props.imageStyle]}

Y el Texto visión basada en:

* avatar = (<Ver estilo={[Styles.view, **esta**.props.style]}>

2 <Estilo de texto={**esta**.props.textStyle}>

También estamos manteniendo los estilos en línea en el mismo archivo para que nuestro componente es verdaderamente autónomo y puede ser autónomo. También tiene sentido utilizar el menor número de bibliotecas externas como sea posible (a diferencia de lo que hicimos aquí, mediante la inclusiónlodash - pero eso es sólo una conveniencia y puede ser fácilmente reemplazado con sencillo Javascript).

Como tarea, intente convertir el Avatar componente en un componente funcional puro.

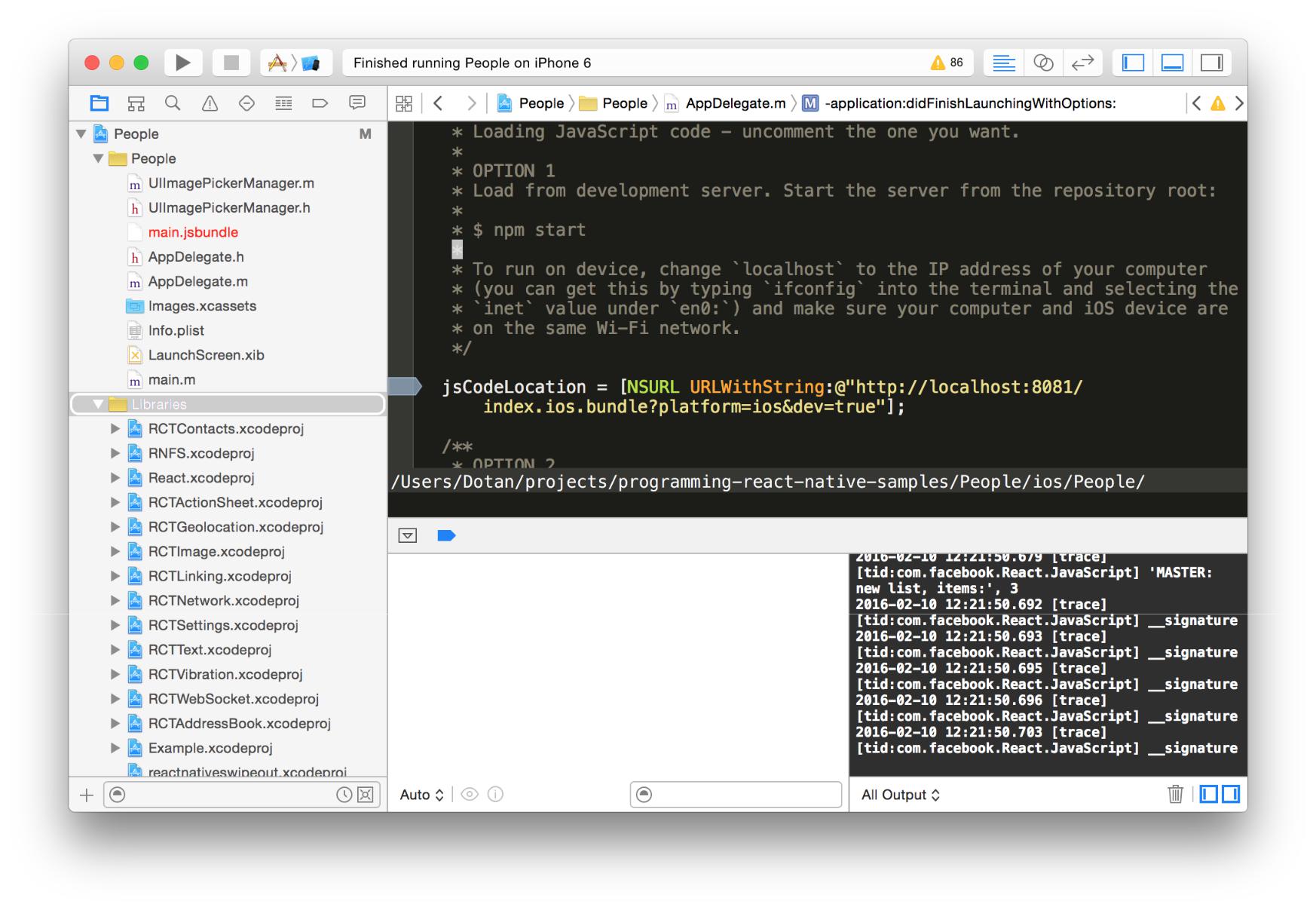
**Utilización de componentes de la comunidad**

Hasta ahora hemos usado componente comunitario con bastante facilidad NPM para instalar estos y rnpmde unirlas. Mientrasrnpmes útil (y muy nuevo en el momento), es importante ver lo que hay que hacer para componentes “link” a su iOS y Android proyectos. Esperamos que en el futuro,rnpm será tan a prueba de balas que no será necesario que el conocimiento en absoluto.

**Vincular los proyectos de iOS**

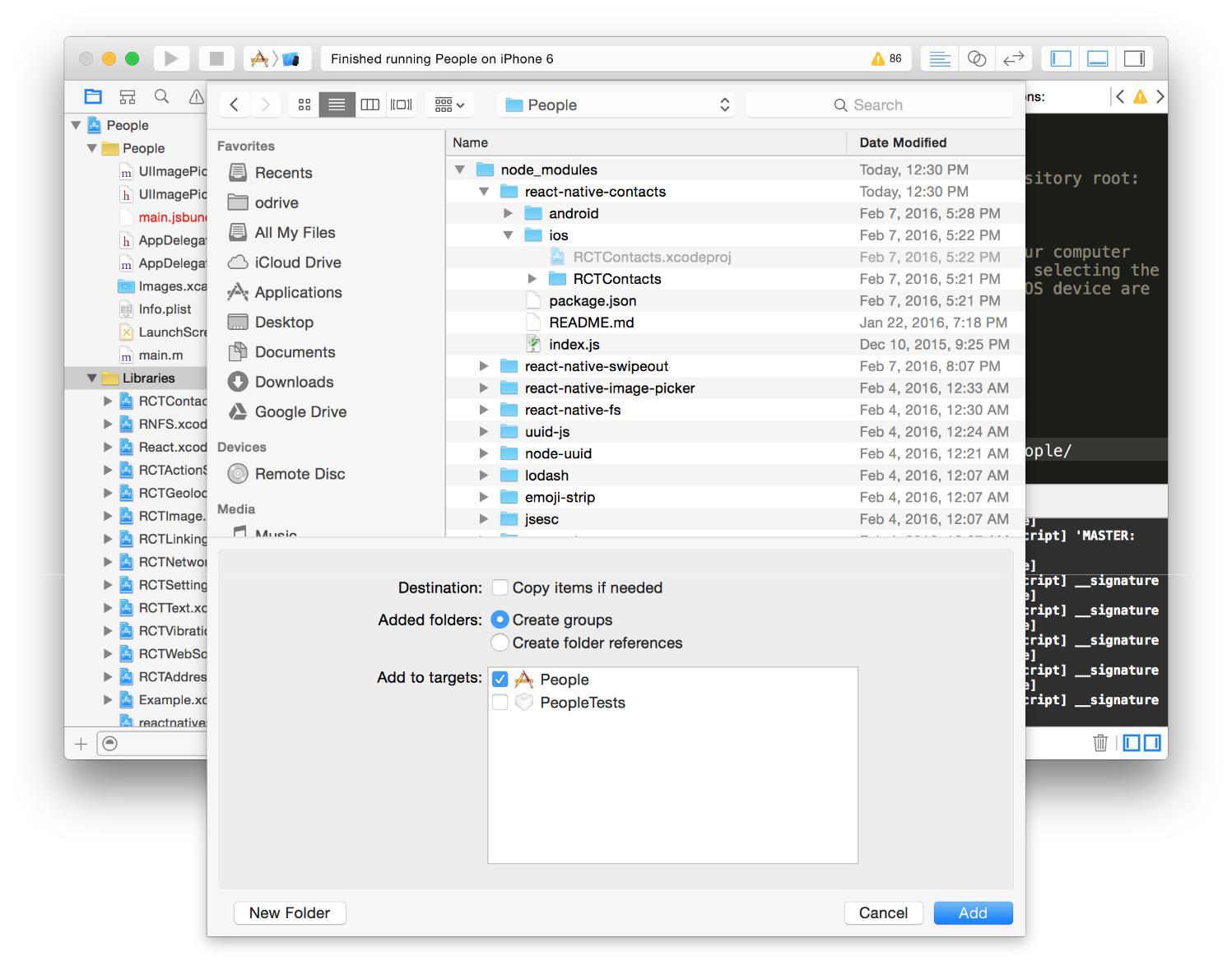
Algunos componentes tienen una parte nativa de ellos, y que deben estar vinculados en contra de su proyecto nativa. Para iOS estos son los pasos que debe seguir después de instalar ese componente conNPM:

|  |  |
| --- | --- |
| Tutorial | 101 |



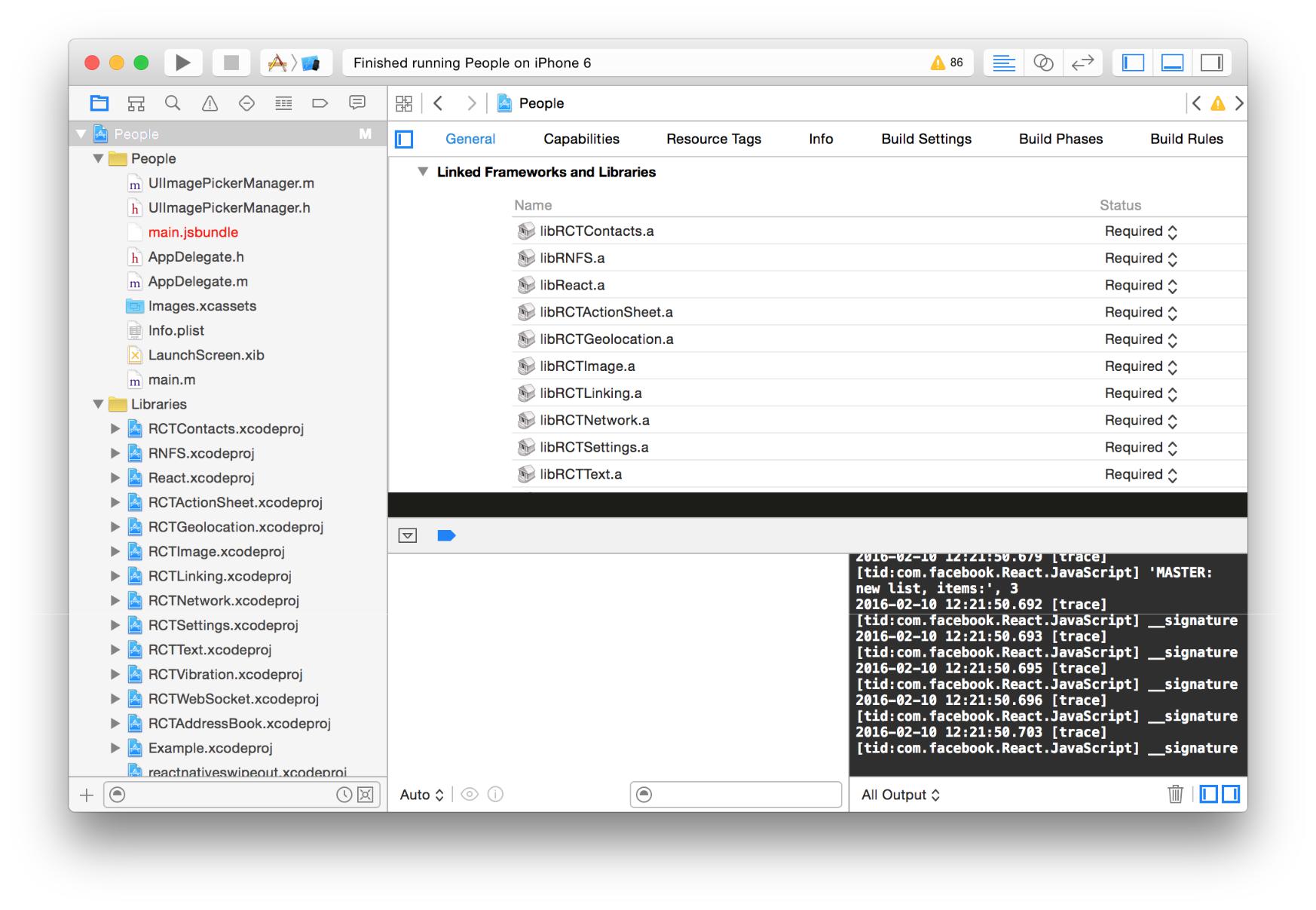
Busque su grupo de bibliotecas dentro de su esquema de proyecto, y el pop en el menú contextual con el botón derecho del ratón. A continuación, seleccione “Agregar nuevo archivo”.

|  |  |
| --- | --- |
| Tutorial | 102 |



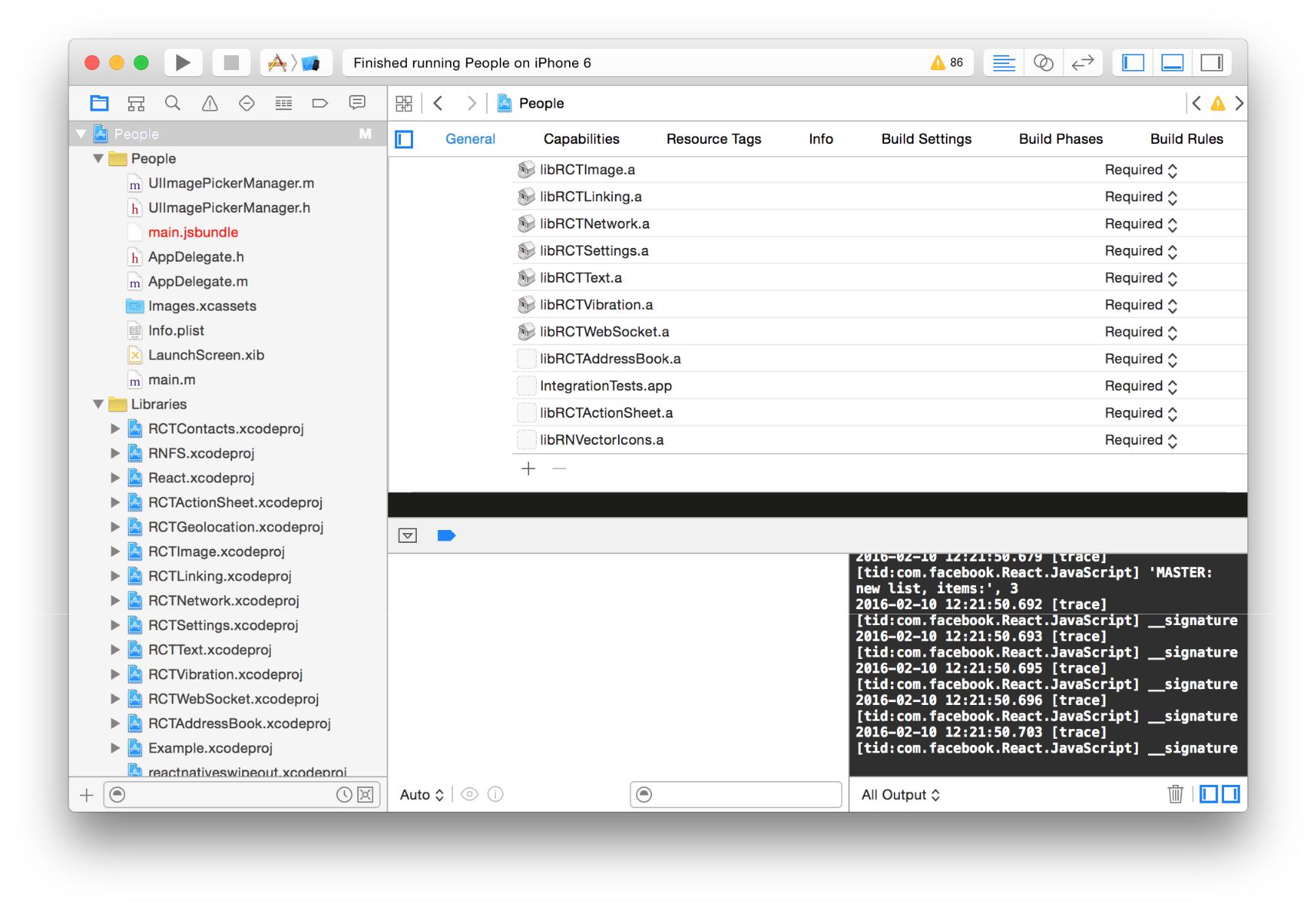
En su árbol fuente del proyecto, encontrar su principal node\_modulescarpeta, y profundizar en la biblioteca que desea vincular. En nuestro ejemplo vamos a vinculará los niveles-reaccionar-contactos nativos biblioteca. Usted necesita encontrar el proyecto Xcode iOS (extremos con .xcodeproj). Una vez encontrado, seleccionarla.

|  |  |
| --- | --- |
| Tutorial | 103 |



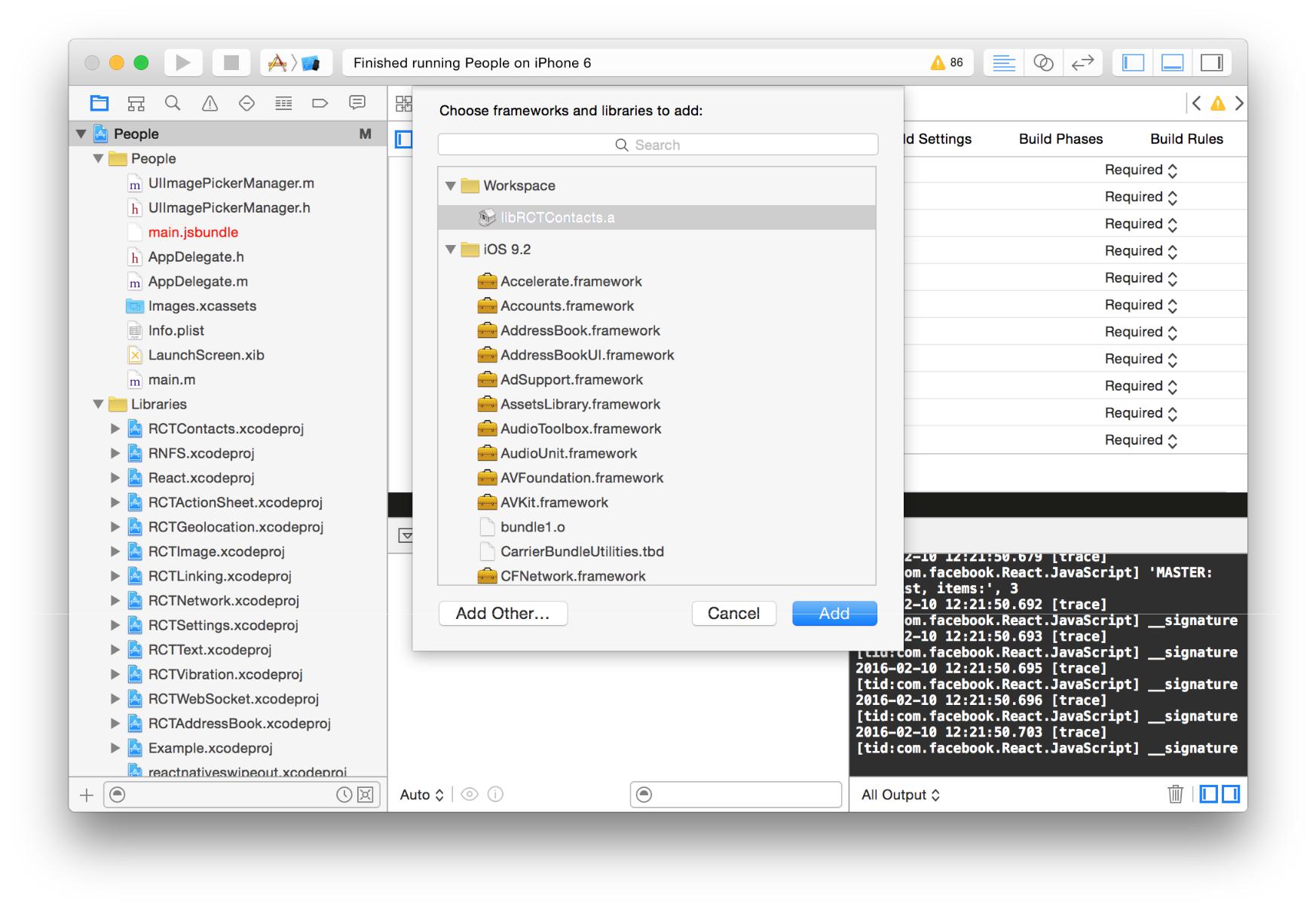
A continuación, vaya a la configuración de proyectos de construcción, en la pestaña “General”. Desplazarse hacia abajo-salas y encontrar los “Marcos Vinculados y bibliotecas” sección.

|  |  |
| --- | --- |
| Tutorial | 104 |



En la parte inferior de esta sección encontrará un botón “+”. Haga clic en eso.

|  |  |
| --- | --- |
| Tutorial | 105 |



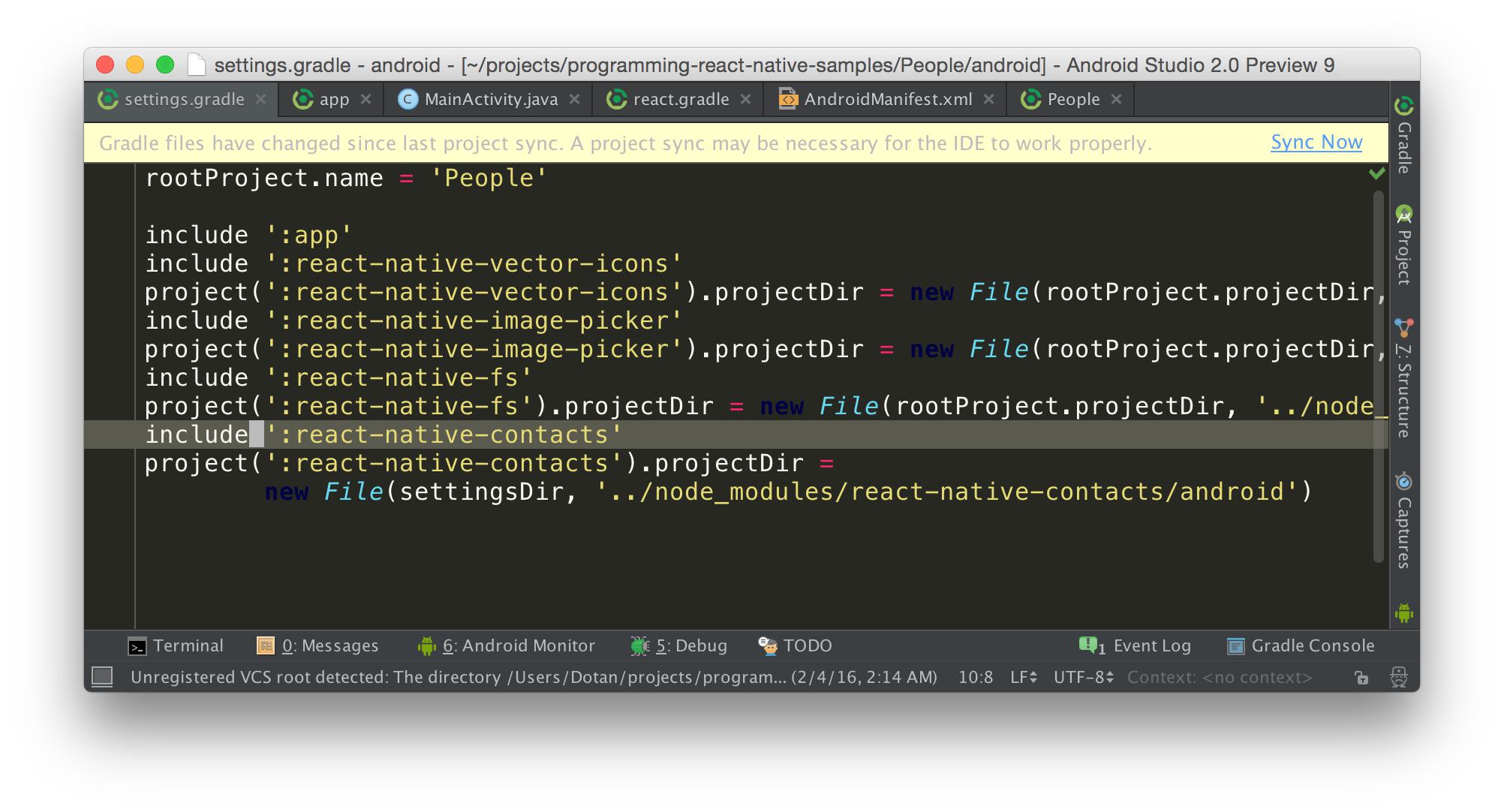
Tendrá que encontrar la .unbiblioteca para enlazar, por lo general aparecerá en primer lugar, y las bibliotecas que no están vinculados enumerados en el mismo, por lo que debe ser muy obvio. Aquí estamos viendo lalibRCTContacts.a binario que tenemos que añadir.

Eso es básicamente la misma. Construir para asegurarse de que todo pasa con éxito.

**La vinculación de Proyectos Android**

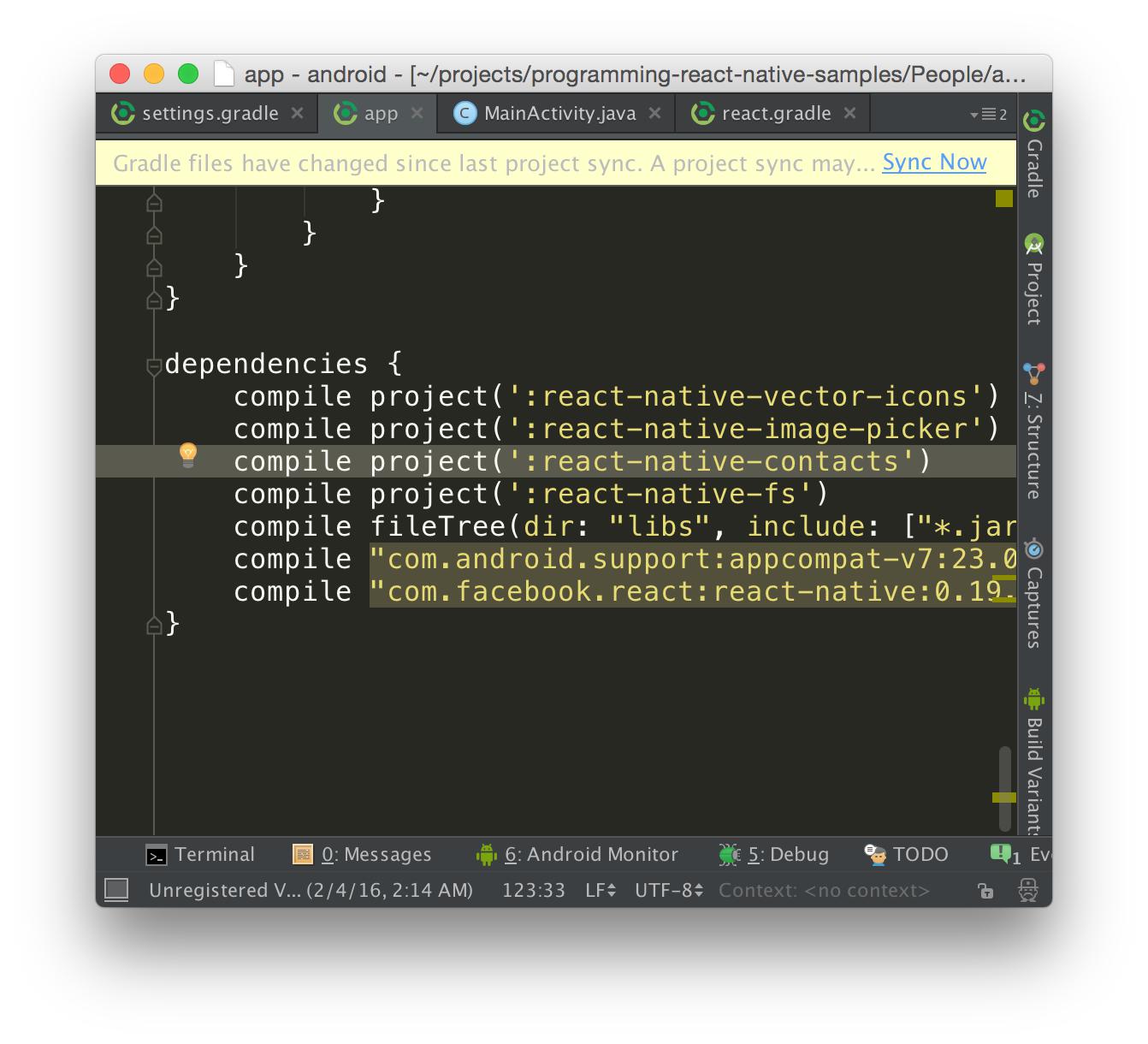
Por suerte, Android adoptó Gradle un par de años. Esto hace la vida muy simple para nosotros.

|  |  |
| --- | --- |
| Tutorial | 106 |



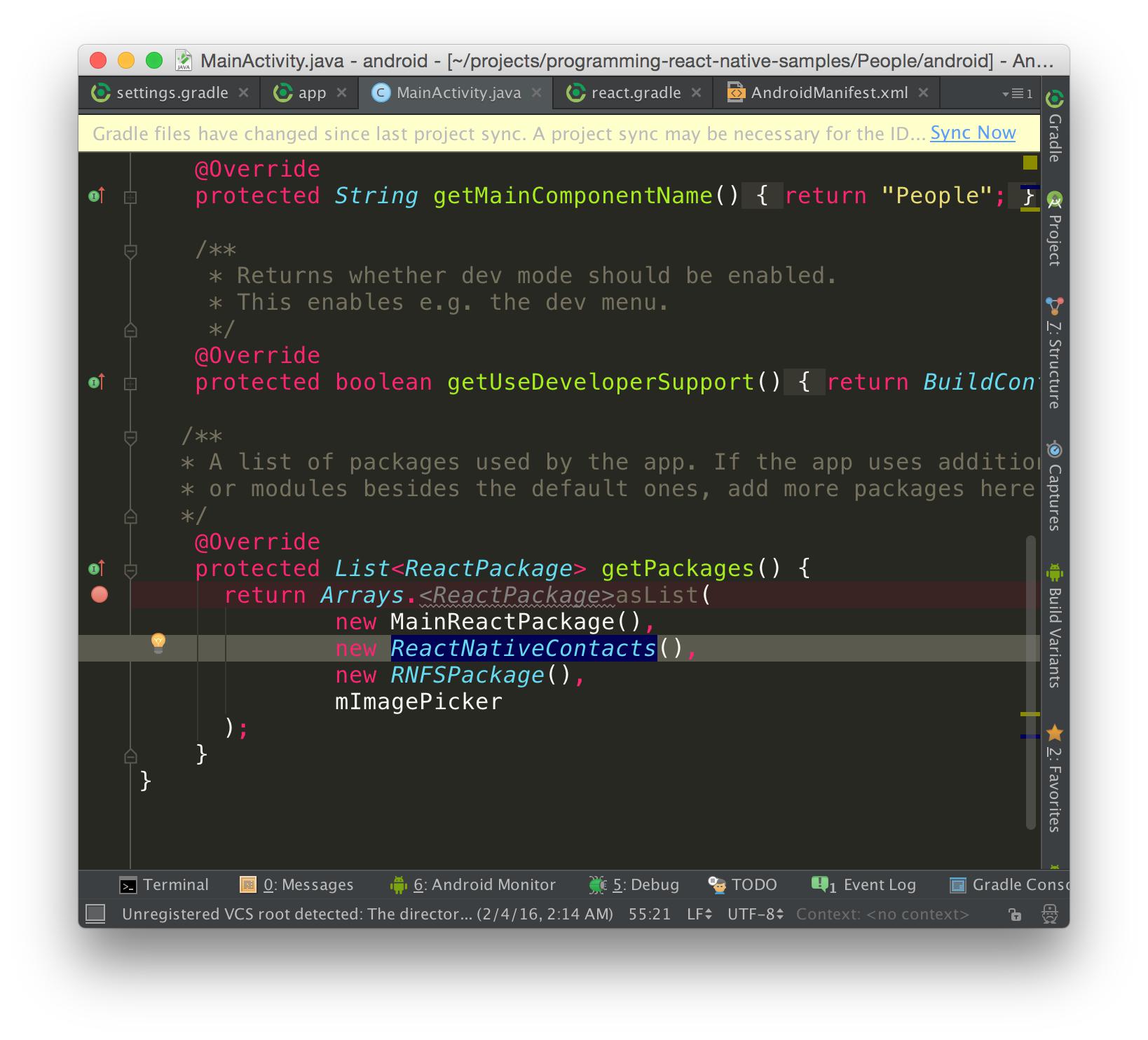
localizar el settings.gradlepresentar en la raíz de su proyecto, y añadir las líneas anteriores a la misma. Básicamente estamos diciendo que hay un nuevo proyecto de software, y su ubicación es en algún lugar dentronode\_modules; usted tendrá que localizar manualmente. Aquí, de nuevo estamos vinculando la-reaccionar-contactos nativos módulo.

|  |  |
| --- | --- |
| Tutorial | 107 |



A continuación encontrar su app / build.gradlepresentar y declarar que el proyecto como una fuente de compilación. Esta es la forma en Gradle toma dependencias y sabe para incluir la fuente de un subproyecto en el objetivo principal compilado. Ahora, el código nativo contactos puede ser visible dentro de nuestra propia.

|  |  |
| --- | --- |
| Tutorial | 108 |



Registrar el paquete (se puede leer más en los paquetes y módulos nativos en el capítulo dedicar módulos nativos) dentro de la actividad principal (y único) como el de arriba. Tendrá que incluir todas las importaciones de Java que faltan (Intellij / An-androide Estudio solicitará felices de hacer esto para usted).

Eso es todo, construir y asegurarse de que todo pasa.

**componentes Javascript**

No hay nada para enlazar aquí como casi todos los proyectos que está en NPMestará disponible para usted. Es una buena idea leer todo el proyecto deREADMEpara tratar de averiguar si hay alguna parte nativa que necesita saber. En el momento de escribir esto, no hay una forma estándar de declarar:

* Si el proyecto tiene partes nativas
* Si es compatible tanto para iOS y Android, o sólo uno de ellos

|  |  |
| --- | --- |
| Tutorial | 109 |

* Si hay cualquier paso adicional que hacer (por ejemplo, incluyendo los recursos, como en el caso de -reaccionar-vector-iconos nativos)

Y de nuevo, espero rnpm está destinado a cambiar y hacer que estas preocupaciones transparente para el usuario, sin embargo, sin un diseño estandarizado proyecto - es prudente que yo y usted, o menos, acerca de cualquier proyecto dado.

Otra preocupación en la evaluación de los componentes de JavaScript, es tener en cuenta el riesgo de que utiliza algunas funciones Node.js que no está disponible con el motor nativo Reaccionar (trate de usar el popular node-uuid para generar UUID en lugar de uuid-jsla utilizamos). En este caso simplemente encontrar la variante Javascript pura de la biblioteca.

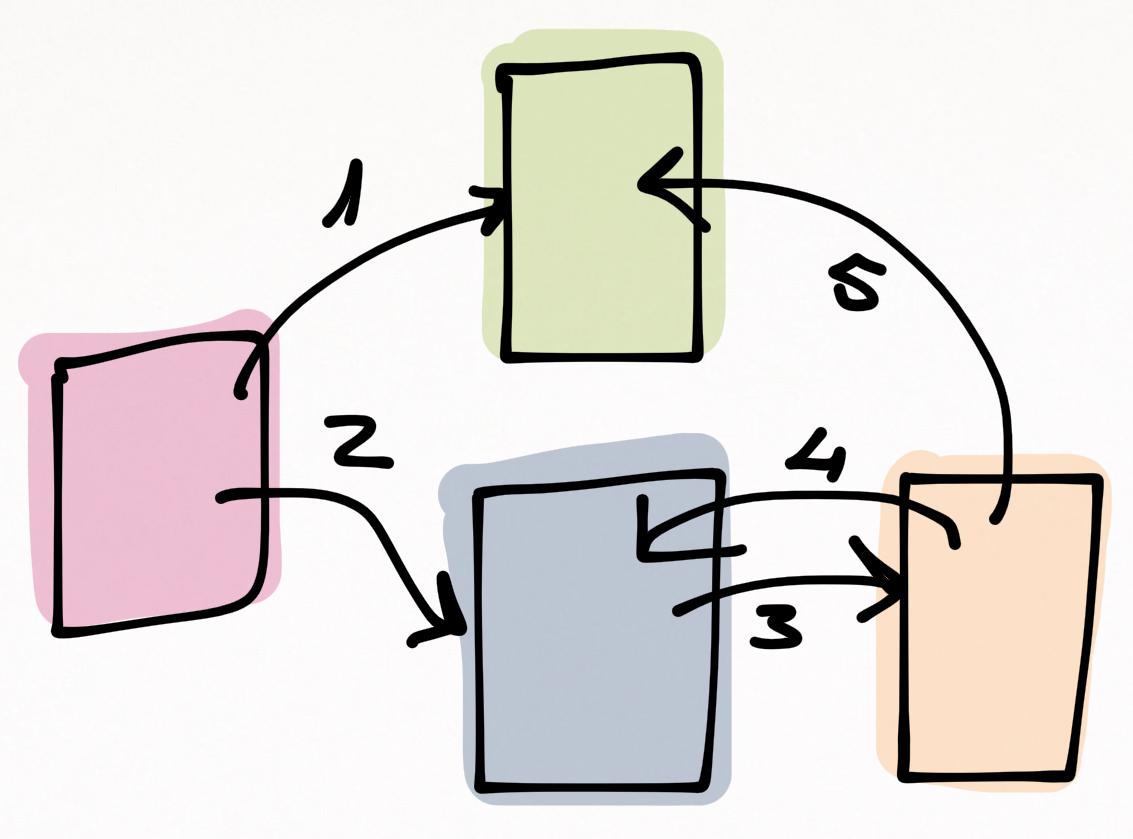
Todos en todo lo que he informado sobre cómo encontrar esos casos extremos que con seguridad ser una pérdida de tiempo, sin embargo, usted debe saber que la mayoría de las bibliotecas funcionan bien sin ningún tratamiento especial. Y por supuesto, está la convención de una biblioteca que tiene el prefijoreaccionar-native-. Usted puede encontrar muchos de esos y más en mi[impresionante reaccionan de forma nativa](https://github.com/jondot/awesome-react-native)[61](#page25) repositorio en Github.

**Resumen**

Este ha sido el capítulo más intenso, lleno de acción todavía. En ella, se ha aprendido sobre un patrón de potencia en aplicaciones móviles (maestro-detalle), y que ha visto cómo construir sus propios componentes inteligentes, componentes mudos. También hemos aprendido acerca de bootstrapping su aplicación en el archivo de entrada y cómo las preocupaciones centrales como la navegación y almacena entran en juego. También hemos visto cómo hacer que los componentes reutilizables y componentes de uso de la comunidad; esta riqueza es impulsado directamente por Reaccionar ser un gran marco para la composición, y el ecosistema que permitió crear. También nos dieron a conocer las herramientas, los entornos de desarrollo,NPM y rnpm. Definitivamente cubierto mucho, y puede que se sorprenda de que los siguientes capítulos no están bajando la barra; vamos a inmersión profunda en temas que son notoriamente difíciles para el desarrollo de aplicaciones móviles.

1. <https://github.com/jondot/awesome-react-native>

**Navegación y enrutamiento**



**¿Por qué la navegación**

Usted puede preguntarse: ¿por qué necesita la navegación? ¿Qué es una pila de navegación? Bueno, la navegación en aplicaciones móviles cubre estas preocupaciones:

* conocimiento de la situación: saber dónde estás, y, posiblemente, de dónde viene
* Funcional: ser capaz de navegar. Volver atrás, deshacer, saltar, o “enlace profundo” en algunas funciones de la aplicación
* Infraestructural: consolidar ceremonias que necesitan ser hechas antes de emitir una nueva pantalla.
* Capacidad de mantenimiento: para darse cuenta de la preocupación anterior, a menudo, que necesita para construir una máquina de estados. Una pila de navegación es una máquina de estados. Plataformas pueden dejar de codificar esta máquina de estados de forma declarativa o visualmente, y promover la capacidad de mantenimiento, manteniendo toda esa lógica en un solo lugar.

**¿Por qué la navegación es asustadiza**

En mi opinión, uno de los temas más espantoso en el desarrollo de aplicaciones móviles es el enrutamiento y la navegación.

infraestructura de navegación por lo general tiene una pila de navegación, que tiene pantallas, que contienen componentes y datos. A menudo, este estado no se represen-tantes en cualquier lugar; es un estado transitorio que el usuario ha creado simplemente navegar alrededor, por lo que hace que sea difícil de razonar acerca.

110

|  |  |
| --- | --- |
| Navegación y enrutamiento | 111 |

flujos de navegación también son difíciles de reproducir, en caso de que toparse con errores en esa zona, ya menudo estos errores llevar a pérdidas de memoria y el tipo (hicimos sólo decir una pila navegador tiene referencia a las pantallas, que sostienen referencia a componentes, que tiene referencia a de datos, y así sucesivamente).

ofertas de navegación UX con cómo cada sistema operativo trajo consigo su propia verdad en cuanto a cómo hacer correctamente la navegación, así como la forma de facilitarlo: implícita comportamiento del botón de regreso de Android y iOS con su explícita puramente barra de navegación de texto.

Y, por último, cada plataforma tiene su propia herramienta y cómo aliviar los dolores por todo dicho. Android tiene el enfoque tradicional, en el que construir las pantallas y las transiciones de código con el omnipotenteIntenciónY últimamente iOS ofrecen un enfoque más intuitivo, con guiones gráficos y casi conectar físicamente pantallas.

**La navegación en React nativo**

La mala noticia es que la navegación no recibe menos miedo a mí, incluso con Reaccionar nativo. Supongo que para mí el dicho “cuanto más se sabe, más se preocupe” se aplica dolorosamente.

Sin embargo, la buena noticia, o tal vez la gran noticia es - que con la Reaccionar, Javascript, y reaccionar combinado todo lo nativo se vuelve sorprendentemente más predecibles.

* Reaccionar deportes de una vía de unión, lo que nos ayuda a razonar acerca de nuestro código. Así que para encontrar fugas de memoria viajamos un árbol de dependencias en una sola dirección (tu caso es distinto).
* Javascript trae consigo las Devtools Chrome, que por ahora son uno de los conjuntos de herramientas más potentes para los desarrolladores. Si usted va a construir algo compleja, tiene suerte de estar viviendo en el año 2015 (por lo menos, que es cuando se escribió este libro, considérese afortunado si usted es del futuro!).
* Reaccionar nativo nos da un montón de vías de escape. Este es el caso en el que queremos usarNavigatorIOS, El IOS pila de navegación nativa, con el fin de reducir al mínimo el elemento de sorpresa al no emulando una pila de navegación, o simplemente poner - se pegue al sistema operativo anfitrión mejores prácticas.

Además, si lo que se está construyendo es muy dolorosamente complejo (años de esfuerzo y complejidad), supongo que en esta etapa es probable que no necesita realmente este libro y debe construir su aplicación en cada plataforma por separado utilizando herramientas y enfoques dedicado, Solo para estar seguros.

Vamos a echar un vistazo a las dos soluciones de navegación Reaccionar nativos.

|  |  |
| --- | --- |
| Navegación y enrutamiento | 112 |

**Navigator vs NavigatorIOS**

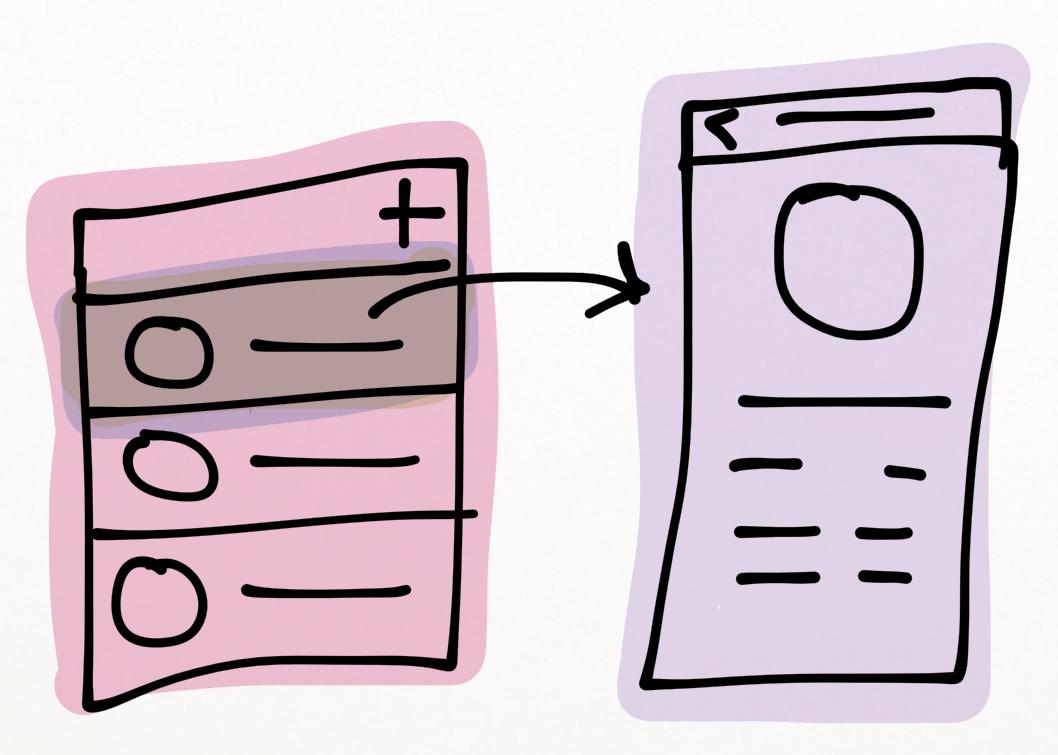
Reaccionar nativo comenzó en IOS, y esta es la razón por la mejor en su clase, o tal vez más utilizado es la pila de navegación NavigatorIOS. Si usted está construyendo una única aplicación para iOS y quiere ir a lo seguro, lo utilizan. Sin embargo, si un año o más pasó de la escritura del libro, podría ser posible que el genéricoNavegador empequeñecido NavigatorIOS en las características y la estabilidad, por lo que se permita a evaluar de nuevo buscando en Google alrededor.

Navegador Reaccionar es el intento del nativo para abstraer el concepto de navegación. A el momento de la escritura, es bueno-pero-no-gran apuesta para una conducción suave si se quiere mantener una base de código de navegación común para iOS y Android.

Usaremos NavigatorIOS para iOS y Navegadorpara Android. Incluso si la realidad cambia yNavegadorse convierte en el todo de extremo toda la solución de navegación para ambas plataformas, me gustaría mantenerla intacta. Se siente que la navegación en iOS y Android siempre va a querer ser diferente, de alguna manera, ya sea UX o hardware-sabio, así que tiene sentido para el futuro a prueba de esto y hacer espacio para el código de evolucionar de manera diferente.

**Navegador**

Vamos a través de una breve Navegadorejemplo. Para ello, vamos a suponer que el patrón de detalle maestro común, donde tenemos una vista maestra que contiene una lista de elementos y, a continuación, al tocar un elemento que queremos para navegar a un elemento secundario (o detalle).



**cableado Navigator**

Esto es lo que probablemente quería leer antes me llena la cabeza con botón de retroceso locura.

El Navigator es un componente Reaccionar que se ocupa de dos cuestiones principales:

|  |  |
| --- | --- |
| Navegación y enrutamiento | 113 |

1. La devolución de un bien configurado Navegadorcomponente, con el icono, icono de la espalda, los colores, la ruta inicial y más. Además, se debe cablear unarenderScene manipulador, que hablamos a continuación.
2. Una aplicación manejador renderScene, que es una función que se encarga de asignar una ruta (que sólo un objeto de JavaScript) a una pantalla de nuestra elección. El objetivo es similar en concepto al backend enrutamiento con marcos como Express o rieles, que se ocupa de la separación de enrutamiento de destino.

Aquí es cómo poner esto en marcha, por un sencillo de dos de pantalla ( “primera” a “segundo”)

flujo de navegación:

* **clase** Navegación **se extiende** React.Component {

2render () {

* **regreso** (
* <Navegador
* estilo={} Styles.container
* initialRoute={{carné de identidad: 'primero'}}
* renderScene={**esta**.navigatorRenderScene}/>
* );

9}

10

1. navigatorRenderScene (ruta, navegador) {
2. \_navegador = navegador;
3. **cambiar** (Route.id) {
4. **caso** 'primero':
5. **regreso** (<primer navegador={} Navegante título="primero"/>);
6. **caso** 'segundo':
7. **regreso** (<En segundo navegador={} Navegante título="segundo"/>);
8. }
9. }
10. }

En el hacer función, hemos creado una Navegador, En la que las propiedades más importantes son para especificar el initialRoute y renderScene. Tenga en cuenta que una ruta es simplemente cualquier objeto JavaScript que se llevará de pantalla en pantalla. A continuación, hacemos un convenio para tener nuestra principal preocupación enrutamiento existir detrás de unacarné de identidadpropiedad; si tenemos cualquier otra cosa que pasar, vamos a utilizar una carga útil adecuada, dentro del mismo objeto. Sin embargo, vamos a estar de acuerdo en quecarné de identidad siempre será reservado para el enrutamiento.

La buena noticia es que pasamos por la parte torpe de declarar una Navegadorcomponentes y el cableado de las pantallas. Puede llegar a ser menos torpe, tal vez, algún día. Puede tener algún tipo de DSLish sienten a la misma, tales como:

|  |  |
| --- | --- |
| Navegación y enrutamiento | 114 |

* *// Esto en realidad no existe, pero se puede hacer*

2 **var** Navegación = rutas ({

3'primero': (Ruta, navegador)=>{

4<primer título="Foobar">

* }

6 })

Pero este tipo de experiencia no existe todavía. Es toda una fruta que cuelga bajo, por lo que pensar en ello como un proyecto de fin de semana agradable de hacer :)

**ToolbarAndroid y Navigator.NavigationBar**

Es importante notar que Navegadorviene con una pieza pre-cocido de interfaz de usuario que funciona como su barra de navegación superior de Android, así como la barra de navegación iOS. Si va a profundizar en[UIExplorer](https://github.com/facebook/react-native/tree/master/Examples/UIExplorer/Navigator)[62](#page30) dentro de facebook / reaccionar nativo Github repo, verá este tipo de cosas:

* **var** NavigationBarRouteMapper={

2

3Botón izquierdo: **función**(Ruta, navegador, índice, navState) {

4**Si** (índice=== 0) {

5**return null**;

* }

7

8**var** previousRoute=navState.routeStack [índice- 1];

* **regreso** (

1. ...
2. );
3. },

13

1. Botón derecho: **función**(Ruta, navegador, índice, navState) {
2. **regreso** (
3. ...
4. );
5. },

19

1. Título: **función**(Ruta, navegador, índice, navState) {
2. **regreso** (
3. ...
4. );
5. },

25

1. };
2. <https://github.com/facebook/react-native/tree/master/Examples/UIExplorer/Navigator>

|  |  |
| --- | --- |
| Navegación y enrutamiento | 115 |

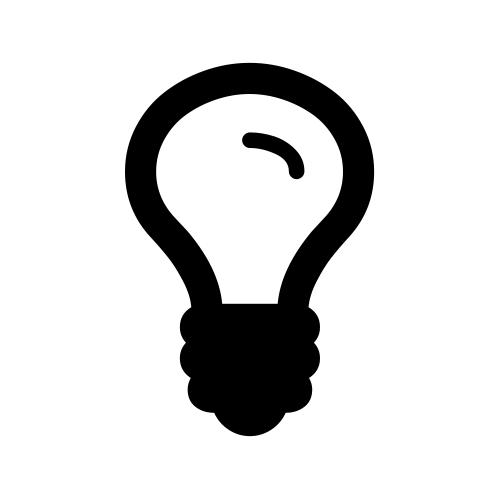
Ese es un concepto de un NavigationBarRouteMapper o en otras palabras, usted está diciendo a su Navegador cómo combinar la información Barra de navegación'S interfaz de usuario en cada ruta.

A continuación, se pega una real Barra de navegación en tu Navegador de esta manera:

* <Navegador
* ...
* barra de navegación={
* <Navigator.NavigationBar
* routeMapper={} NavigationBarRouteMapper
* />
* }

Así que lo que tenemos aquí, es una Navegador lo que hemos aprendido cómo configurar y diga que el modo de hacer a sí misma como una respuesta a una ruta determinada, y una Barra de navegación que más o menos hace lo mismo.

Inmediatamente, usted comienza a pensar - ¿por qué separar los dos? TuBarra de navegación y Navegador prestación se extendió a cabo a través de dos creadores de mapas di-ferentes, y también, por qué no hacer una ruta a un ciudadano de primera clase y haciéndose render? algo comoShowCartRoute.render (), y entonces ShowCartRoute.renderBar ()? Bueno, esto es loExponentjs / ex naveganteresuelve con exactitud, y más o menos se siente. yo recomiendo[comprobando](https://github.com/exponentjs/ex-navigator) [fuera](https://github.com/exponentjs/ex-navigator)[63](#page31). Además, es posible imaginar que esto no va a ser el único la gente de sabor les gustaría hacer el enrutamiento con, por lo que seguir viendo a cabo por las cosas nuevas.



En caso de utilizar una ToolbarAndroid o una Navegador'S propia Barra de navegación? La respuesta es de nuevo - compensación. losBarra de navegación está fuertemente ligada a las rutas, ToolbarAndroid está ligado a la vista. ToolbarAndroid Es decir, así, Android, y que probablemente se podría implementar una cosa así mismo de forma genérica.

Así que la línea inferior, si se va a implementar algo sencillo, y luego ir con de iOS NavigatorIOS para iOS (que viene en la siguiente sección), y ToolbarAndroidpara Android. De lo contrario, utilice unaNavegador para ambos, y, o bien Barra de navegación, O en su propio pedazo de barra que se construye manualmente. Y, por supuesto, hacer echar un vistazo rápido en mi[impresionante reaccionan de forma nativa](https://github.com/jondot/awesome-react-native)[64](#page31) lista para cualquier bar más elegante que desea.

**ToolbarAndroid**

Asi que Navegador y Barra de navegación a un lado, vamos a dar ToolbarAndroidun decente encubrimiento de edad en caso de que escogería esta opción. En ese caso, las pantallas serán vistas normales que se ocupan de masaje a unaToolbarAndroid subvista a sus necesidades.

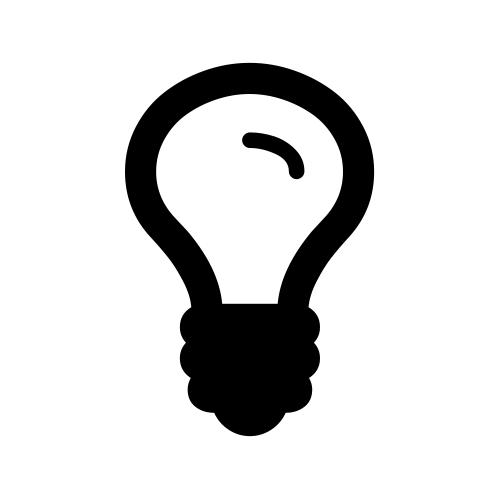
1. <https://github.com/exponentjs/ex-navigator>
2. <https://github.com/jondot/awesome-react-native>

|  |  |
| --- | --- |
| Navegación y enrutamiento | 116 |

UN ToolbarAndroides nuestra barra de navegación superior, y con Reaccionar nativo, es bastante flexible. Tan flexible, de hecho, que es necesario para codificar cada punto de decisión, como cuando para mostrar el icono de la espalda, ¿qué tipo de título para mostrar y así sucesivamente, con base en las pantallas de nuestros actuales y pasados.

UN ToolbarAndroid es también el verdadero androide Barra de herramientas Reproductor (véase [aquí](https://developer.android.com/intl/zh-tw/reference/android/support/v7/widget/Toolbar.html)[sesenta y cinco](#page32)) Que, de acuerdo a nuestro tema aquí (mostrando la forma específica de la plataforma en primer lugar) es un componente específico en Android.

La flexibilidad puede ser bueno y malo. Bueno ya que en cada pantalla, podemos especificar exactamente cómo queremos que las cosas que se tenía en la barra de navegación. Malo, porque tenemos que estar a la defensiva aquí, ya que puede convertirse en una sobrecarga de mantenimiento o una máquina de estados de la peor clase - la que se extiende a través de toda nuestra base de código.



Aquí está nuestra Segundo pantalla:

* **clase** Segundo **se extiende** React.Component {

2render () {

* **regreso** (
* <Ver estilo={} Styles.container>
* <estilo ToolbarAndroid={} Styles.toolbar

|  |  |
| --- | --- |
| 6 | título={**esta**.props.title} |
| 7 | Navicon={exigir('Imagen! Ic\_arrow\_back\_white\_24dp')} |
| 8 | onIconClicked={**esta**.props.navigator.pop} |
| 9 | titleColor={'FFFFFF'}/> |

1. <Texto>
2. Segunda pantalla
3. </ Texto>
4. </ Ver>
5. );
6. }
7. };

Si está tratando de esto, asegúrese de que ic\_arrow\_back\_white\_24dpes un icono que ha caído en la carpeta de recursos - en este caso Android. Por el bien del experimento se puede utilizar una imagen de alta resolución única para todas las variantes de tamaño.

Observamos aquí, que especificamos nuestra Navicon explícitamente. Queremos que los usuarios sean capaces de aprovechar un icono de vuelta a la derecha en la barra de navegación.

El siguiente, nuestra primero pantalla:

1. <https://developer.android.com/intl/zh-tw/reference/android/support/v7/widget/Toolbar.html>

|  |  |
| --- | --- |
| Navegación y enrutamiento | 117 |

* **clase** primero **se extiende** React.Component {
* navSecond () {
* **esta**.props.navigator.push ({
* carné de identidad: 'segundo'
* })
* }
* render () {
* **regreso** (
* <Ver estilo={} Styles.container>

1. <estilo ToolbarAndroid={} Styles.toolbar

|  |  |
| --- | --- |
| 11 | título={**esta**.props.title} |
| 12 | titleColor={'FFFFFF'}/> |

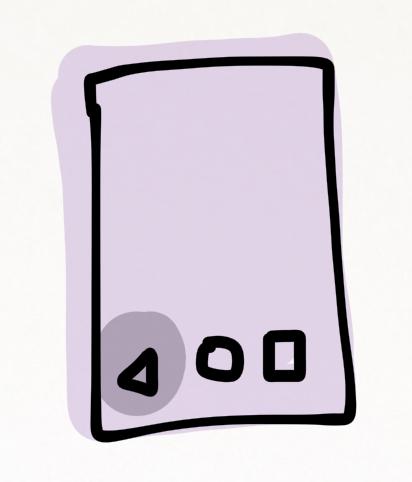
1. <TouchableHighlight onPress={**esta**.navSecond.bind (**esta**)}>
2. <Texto>Vaya a segunda pantalla</ Texto>
3. </ TouchableHighlight>
4. </ Ver>
5. );
6. }
7. }

En este caso, no se especifica un botón de retroceso, ya que reconocemos como la pantalla de raíz. La pieza más importante (y divertido) del código aquí es cómo navegar a laSegundopantalla. Nosotros soloempujar un nuevo objeto que pasa a contener una carné de identidad estuvimos de acuerdo en que es de nuestra propiedad de enrutamiento que basamos nuestra enrutamiento en. Nota que no se especifica el tipo de pantalla, objeto o etiqueta aquí - esta es la esencia de la separación de enrutamiento de destino o aplicación, que es una buena cosa.

En Android, vamos a tener que cuidar de navegación, y también el botón de retroceso.

|  |  |
| --- | --- |
| Navegación y enrutamiento | 118 |

**Trasera del botón de Android**



Aprovechando el botón de retroceso es única para los dispositivos Android, ya que es en ocasiones un botón de “hardware” en una zona de contacto dedicada del vidrio como en los teléfonos Samsung o HTC teléfonos, o un botón de software que hace que en la parte inferior de la pantalla (común en la familia de dispositivos Nexus).

El siguiente texto modelo se necesita más o menos textualmente en nuestra aplicación para apoyar la respuesta al botón de retroceso. Tenga en cuenta que la\_navegador-variable es de alcance global, y se llena en la primera navegación. En primer lugar leer este fragmento de código para entender lo que está pasando, y luego me gustaría recomendar el seguimiento de la\_navegador variable a lo largo también.

* **var** {
* ...,
* ...,
* ...,
* BackAndroid

6 } = Reaccionar;

7

8 **var** \_navegador; *// rellenamos esto en el primero de navegación.* 9

1. BackAndroid.addEventListener ('HardwareBackPress', () => {
2. **Si** (\_navigator.getCurrentRoutes (). Longitud=== 1) {
3. **falso retorno**;
4. }
5. \_navigator.pop ();

|  |  |
| --- | --- |
| Navegación y enrutamiento | 119 |

1. **return true**;
2. });

BackAndroid es una biblioteca simple que se une a los eventos nativos del huésped dispositivo.

*Profundo alerta de buceo!*: Aquí es como funciona (nótese el caso especial exitApp):

* RCTDeviceEventEmitter.addListener (DEVICE\_BACK\_EVENT, **función**() {

2**var** invokeDefault= **cierto**;

3\_backPressSubscriptions.forEach ((suscripción) => {

4**Si** (Suscripción ()) {

5invokeDefault = **falso**;

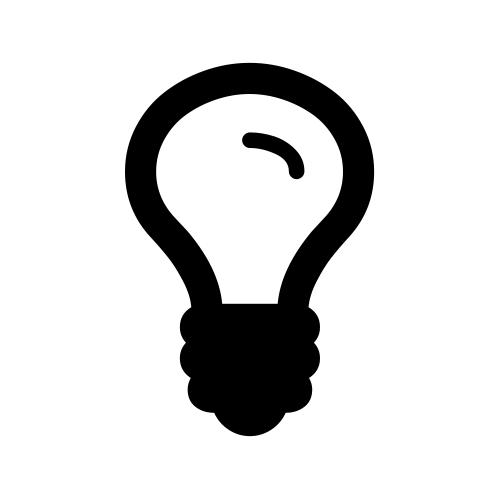
* }
* });
* **Si** (InvokeDefault) {
* BackAndroid.exitApp ();

10}

11 });

En iOS usted no necesita preocuparse si estás por error (o si perezosamente desea utilizar el mismo código) utilizando la BackAndroid Módulo - todas las funciones que ofrece son no-ops.

Al igual que con cualquier detector de eventos, cuando se agrega un oyente, debe solicitar de inmediato a sí mismo cómo va a quitarlo - hace el suscribir el método devuelve un mango especial que necesita para proporcionar, cuando cancelando ?, o tiene que hacer la contabilidad usted mismo y llegar a la misma referencia al controlador que ya ha proporcionado?



En el BackAndroidcaso, cuando suscribimos hay que mantener una referencia a la función de controlador que le damos. Sin embargo, lo que realmente vemos deshabilitar el botón de retroceso en la vida real?

**NavigatorIOS**

Si usted no está haciendo una versión para Android de su aplicación, o si usted no decide implementar Navegador y Barra de navegaciónentonces este NavigatorIOS es mucho más simple, y se basa en la pila nativa de navegación iOS. La pila de navegación IOS es una poderosa bestia, sólo es alentador que se oculta bajo una sencilla Reaccionar dicho componente nativo, sin embargo, estar seguro de que si queríamos hacer las cosas más complicadas (personalizadas SEGUÉS y tal) que podría haberse convertido en más difícil.

Por ahora, vamos a implementar la misma Navegador ejemplo, con NavigatorIOS.

|  |  |
| --- | --- |
| Navegación y enrutamiento | 120 |

* **clase** Navegación **se extiende** React.Component {

2render () {

* **regreso** (
* <NavigatorIOS
* estilo={} Styles.container
* initialRoute={{
* título: 'primero',
* componente: primero
* }} />

1. );
2. }
3. }

Nuestra Navegación componente es simple y explícita, y la ruta inicial especifica el componente textual (aquí primero) Que queremos ejecutar. Como nota al margen, también al igual que llamar a este componenteRouter o Entrenador de animales o cualquier cosa que representa un concepto de un componente de la cubierta de enrutamiento.

A continuación, echamos un vistazo a nuestros dos pantallas, primero y Segundo.

* **clase** primero **se extiende** React.Component {
* navSecond () {
* **esta**.props.navigator.push ({
* título: 'segundo',
* componente: Segundo
* })
* }
* render () {
* **regreso** (

1. <Ver estilo={} Styles.content>
2. <TouchableHighlight onPress={**esta**.navSecond.bind (**esta**)}>
3. <Texto>Vaya a segunda pantalla</ Texto>
4. </ TouchableHighlight>
5. </ Ver>
6. );
7. }

Algo similar en estructura a NavegadorSin embargo, de nuevo vemos una mención explícita de la Segundo pantalla.

Tenga en cuenta que cuando se utiliza una NavigatorIOS y un niño normal Vercomo en este caso, tendremos que manejar la altura de una típica barra de navegación iOS. En otras palabras, tenemos que añadir unaacolchado superior propiedad a nuestra Ver tal que el contenido será compensada por debajo de la navegación bar. En los componentes más avanzados, tales como elScrollView, Lo que se quiere buscar la automaticallyAdjustContentInsets propiedad que permite que el componente de manejar por nosotros automáticamente.



|  |  |
| --- | --- |
| Navegación y enrutamiento | 121 |

los Segundo pantalla parece al menos tan simple:

* **clase** Segundo **se extiende** React.Component {

2render () {

* **regreso** (
* <Ver estilo={} Styles.container>
* <Texto>
* Segunda pantalla
* </ Texto>
* </ Ver>
* );

10}

11 };

Como se mencionó durante el Navegador visión de conjunto, NavigatorIOS es más simple - no tenemos ninguna barra de navegación para ajustar aquí, y en este caso el punto de vista es completamente vainilla - reutilizables y limpia como está.

**datos pasando**

A menudo, cuando se hace la navegación, además de la ruta, queremos transmitir un conjunto de datos adicionales, para la nueva pantalla-a-ser para erradicar sucesivamente.

En el genérico Navegador componente, los datos se pasa como parte del objeto Javascript llano (en este ejemplo, el datos propiedad), por lo que podemos hacer algo-algo como esto:

* **cambiar** (Route.id) {

2**caso** 'primero':

3 **regreso** (<primer navegador={} datos del navegador={} Route.data título="primero"/ \ 4 >);

Con Barra de navegación que en su mayoría serán lo mismo, si se cuida a pasar sus datos dentro de su routeMapper objeto de ruta, y recogiendo lo distinguen en su routeMapper devoluciones de llamada.

Con NavigatorIOS tenemos que utilizar el especial passProps propiedad:

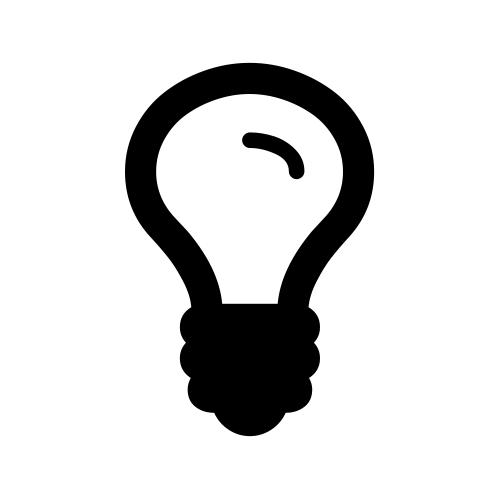
|  |  |
| --- | --- |
| Navegación y enrutamiento | 122 |

* <NavigatorIOS
* initialRoute={{
* componente: Primero,
* título: 'primero',
* passProps: {datos: 'algunos datos' },
* }}

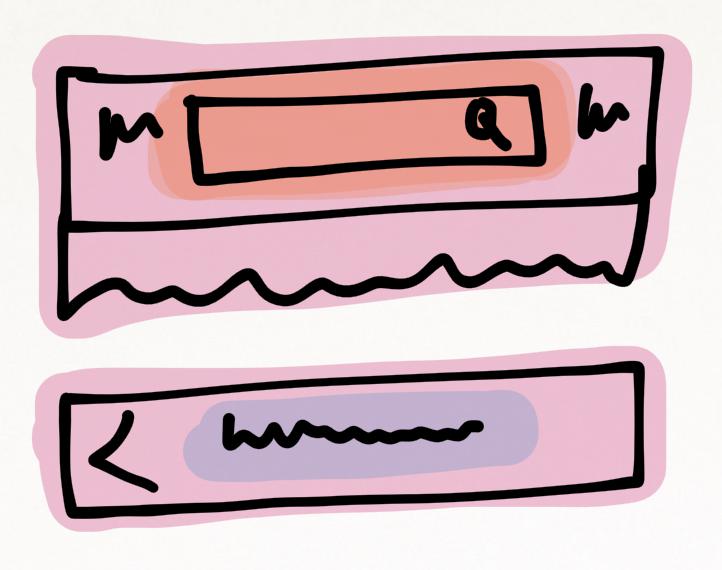
7/>

Y el componente receptor obtendrá tanto el datos propiedad y un especial ruta y navegador los apoyos que puede utilizar para tomar decisiones y para navegar promover.

Hay una pregunta fascinante acerca de cuándo una nueva pantalla nace debido a un nuevo sistema de navegación. Son los datos que la pantalla justo ya ha recibido la ruta un puntero a los datos que necesita para traer ?, o los datos reales que necesitaba para traer textualmente ?. No hay respuesta es incorrecta: es una solución de compromiso entre ser implícita y explícita, o defensiva más que ingenuo, respectivamente.



**Patrones**



**Buscar en la barra de navegación**

* NavigatorIOS - No se puede hacer esto.
* Navegador, con Navigator.NavigationBar - asegúrese de que el asignador de la barra de navegación hará que su variante de un título como este:

|  |  |
| --- | --- |
| Navegación y enrutamiento | 123 |

* Título: **función**(Ruta, navegador, índice, navState) {
* **regreso** (
* <AwesomeSearchbar .../>
* )
  + Navegador y ToolbarAndroid - asegurarse de que la pantalla se está trazando a, que se supone que contiene la barra de búsqueda inline dentro de la barra de navegación, ahora deben emitir un solo niño que es su barra de búsqueda, prefiere no proporcionar un título en este caso (el estilo y los demás son cortadas por razones de brevedad):
* <ToolbarAndroid>
* <AwesomeSearchbar .../> 3 </ ToolbarAndroid>

Una barra de búsqueda es interactiva - es totalmente OK para realizar la AwesomeSearchbar componente interactivo mediante el suministro de devoluciones de llamada a través de accesorios. Puede tirar de estas devoluciones de llamada de dos maneras:

1. NavigatorBar render - de su ruta, o haciendo que la toma de render parámetros.
2. ToolbarAndroid - ya que se hace dentro de una vista de contenedores, simplemente el devoluciones de llamada del componente inteligente neareset harán.

**De contenido personalizado en el título**

Al igual que en el modelo anterior, la idea es sans similares las devoluciones de llamada.

* NavigatorIOS - el contenido es una cadena única
* NavigatorBar y ToolbarAndroid - Al igual que con el patrón anterior, simplemente entregar el componente que desea hacer, esta vez no necesitan devoluciones de llamada o la interactividad.

**Enrutada contenido barra de navegación**

Vimos esta manipulación, mientras NavigatorBar asignador - el contenido puede cambiar en función de la ruta no se viaja en.

* NavigatorIOS - cada vez que se empujar un sistema de navegación, se puede definir la forma en la llegando a ser barra de navegación se verá así, por lo que este es un asunto sencillo de utilizar un nuevo título:

|  |  |
| --- | --- |
| Navegación y enrutamiento | 124 |

* **esta**.props.navigator.push ({
* título: "Un nuevo título",
* ...
* })
  + Navegador y Barra de navegación - utilizar el asignador de ruta, como hemos visto antes.
  + ToolbarAndroid - esto es trivial ya que estás haciéndolo como parte de la vista, por lo que pueden par de la prestación a la vista en sí. Es decir, no quejarse con la ruta, pero con el contenido real componente de vista.

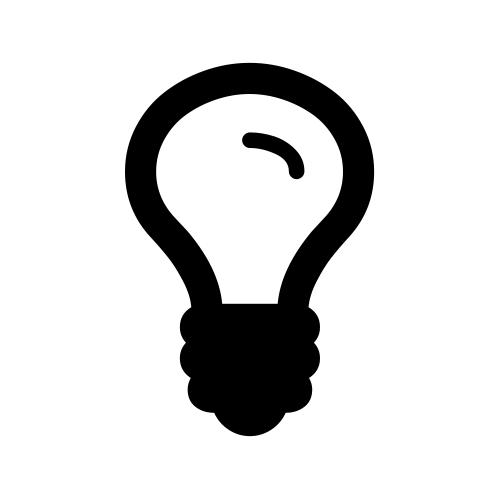
**Reactiva contenido barra de navegación**

Llega un momento en que los cambios navbar, pero no como una reacción a los cambios de ruta, pero a una cierta clase de un evento externo, evento programado, o cualquier cosa de naturaleza reactiva.

La forma de resolverlo en este momento, que está ganando consenso, es inyectar un emisor de sucesos a sus flujos de aplicaciones, y asegúrese de que los componentes de ese barra de navegación saben utilizarlo.

* NavigatorIOS - de nuevo, no es posible
* Navigator.NavigationBar y ToolbarAndroid - asegurarse de que el contenido das cada una, será capaz de utilizar su emisor evento global:
* <ToolbarAndroid>
* <emisor AwesomeSearchbar={**esta**.emitter} .../> 3 </ ToolbarAndroid>

Tampoco se puede utilizar un emisor explícita sino un despachador Flux, o no utilizar un emisor explícito y lo inyecta a través de algo similar a reaccionar de [contexto](https://facebook.github.io/react/docs/context.html)[66](#page40), etc. Dado que este es un tema material avanzado / pensamiento (En la categoría de “bombilla”) Lo dejo a usted para la exploración.



**entrada conseguir**

Si quieres ir a una pantalla para recolectar información (es decir, los modales), puede utilizar el real Modal componente (ver [aquí](https://facebook.github.io/react-native/docs/modal.html)[67](#page40)), O expulsar a una vista con una devolución de llamada dentro de la renderScenebloque lógico. En casos extremos se puede utilizar emisor de sucesos, o si está usando Flujo de una acción fundente trivialmente lo soluciona.

1. <https://facebook.github.io/react/docs/context.html>
2. <https://facebook.github.io/react-native/docs/modal.html>

|  |  |
| --- | --- |
| Navegación y enrutamiento | 125 |

**la difusión de los apoyos**

Si nos rascamos la cabeza por un momento, recordamos que en el encaminamiento Navegadorse hace con un objeto Javascript llanura. En realidad podemos ser obstinado y definir este objeto como tal:

* {
* carné de identidad: 'Ruta-id'

3accesorios: {

* algunos: 'accesorios'
* }

6 }

Y de esta manera que se puede hacer algo como esto, dentro de nuestra renderScene:

* navigatorRenderScene (ruta, navegador) {

2\_navegador = navegador;

3**regreso** (<primer navegador={navegador}

4 {...} route.props/>);

* }

Usando el nuevo operador de difusión ..., Que en línea con facilidad la totalidad accesoriosla bolsa de propiedades en nuestro componente. Recuerde Reaccionar irá de izquierda a derecha en varios accesorios para que pueda disfrutar de un efecto en cascada (proporcionar valores predeterminados y luego anular con datos específicos)

**Resumen**

Con esto concluye nuestra discusión acerca del enrutamiento. En resumen, hemos aprendido lo siguiente:

* Enrutamiento es difícil, sin embargo, al menos en el móvil, tenemos menos cuerda para colgar con nosotros mismos mediante la adopción de las mejores prácticas de cada plataforma móvil.
* Reaccionar nativo ofrece la genérica y flexible Navegador para un uso caso general, y el mayor NavigatorIOSpara un trabajo específico IOS. Elija este último si no se necesita flexibilidad y quiere correr rápido por conseguir el IOS comportamiento de la pila de navegación por defecto.
* Enrutamiento se define de forma implícita con Navegador y rutas, o explic-tamente con NavigatorIOS y los componentes específicos que ruta.
* Recuerde que debe cubrir una serie de fallos de interfaz de usuario cuando se utiliza una barra de navegación. Parte del contenido puede desaparecer bajo la barra de navegación, ya que no se rellena.
* los datos que pasan a través tanto es fácil Navegador y NavigatorIOS.
* Barra de navegación es un concepto común a ambos, tiene sus propias ventajas y desventajas.

**Going Native: interfaz de usuario nativa**

**¿Por qué ir nativo?**

Reaccionar nativo le permite construir una costumbre, la interfaz de usuario totalmente nativa, y se mezcla en su aplicación existente. Esto se logra mediante la introducción de una serie de abstracciones inteligente y bloques de construcción que usaremos en este capítulo.

Pero en primer lugar, si usted está leyendo este libro de principio a fin, existe la posibilidad desea explorar las razones para el uso de controles nativos puros con Reaccionar nativo. Después de todo, la razón principal para el uso Reaccionar nativo es mejorar tratar con todos y cada plataforma móvil, su utillaje, y la experiencia del desarrollador.

De lo contrario, si usted está aterrizando en este capítulo con la intención de encontrar la manera de hacer este tipo de vudú, entonces es probable que haya tropezado con una de las motivaciones enumero a continuación.

**Actuación**

Hay dos tipos de personas que tratan a cabo optimizaciones de rendimiento - los que realmente tienen un problema de rendimiento y están gastando cada día tratando de aliviar el dolor mediante la identificación de los cerdos de rendimiento, pérdidas de memoria, y los malos árboles de interfaz de usuario, y - los que se enteraron de este tipo de problemas.

En primer lugar, no ser el último; Si usted piensa que tiene un problema de rendimiento, tratar de crear un ejemplo aislado que demuestra. Desgarra esa parte de su aplicación, aislarlo, y trate de reproducir. Muchas veces durante este proceso por sí solo a averiguar el causante raíz de su problema; e incluso si no lo hizo, ahora tiene un ejemplo que se puede compartir con los demás. Se ha dicho que[La optimización temprana es la raíz de todo mal](http://c2.com/cgi/wiki?PrematureOptimization)[68](#page42)Y si resuelve problemas invisibles a pensar que está reparando realmente un problema de rendimiento, es probable que esté creando un problema de rendimiento más grande en su lugar.

Por último, sí. Si usted tiene un problema de rendimiento, el uso y la mezcla en un componente de interfaz de usuario nativa será muy bueno para usted. Estoy dispuesto a hacer una apuesta que está teniendo este problema en Android.

Por lo tanto, como regla general - probablemente se le mezcla en controles personalizados nativa en Android, con la participación listas y animaciones, que son los dos problemas más notorios en esa plataforma.

1. <http://c2.com/cgi/wiki?PrematureOptimization>

126

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 127 |

**Haciendo uso de trabajo existentes**

Si ya edificados aplicaciones iOS nativas, y tratando de hacer una nueva aplicación que va a ser multiplataforma con Reaccionar nativo, entonces es probable que desee utilizar alguna de que la infraestructura iOS existentes y los componentes de interfaz de usuario que se ha construido ya. Por supuesto, lo mismo es cierto si usted ya tiene aplicaciones de Android y quiere ampliar a la plataforma iOS. Esto hace que sea una decisión muy trivial, utilizar esos componentes existentes y cosas barco más rápido.

**mejor Tooling**

No es un secreto que Xcode es bastante bueno. En algunos casos, es posible que se encuentre la construcción de sus interfaces de usuario mucho más rápido en el Interface Builder. Para Android, es menos cierto ya que Android Studio es relativamente joven - pero puede tener sus razones para apreciar mejor Android Studio. Mezcla de controles nativos es bastante inteligente en realidad ya Reaccionar nativo no necesita un ViewController (IOS) o una actividad / Fragmento (Android) - sólo se necesita una visión simple. Así que una vez que hemos diseñado, podrás implementar toda la lógica en Javascript - esto es cómo reaccionan los nativos quita las malas decisiones y te deja con una buena manera de hacer las cosas.

**Interfaz de usuario personalizada y la interfaz de usuario Complejo Trabajo**

Esta sería probablemente la mejor razón para usar controles nativos. Mientras Reaccionar nativos no tienen como objetivo proporcionar una solución que lo abarca todo para todos sus desafíos móviles, usted debe esforzarse para ser práctico. Siempre pensar en eso[clavija cuadrada en agujero redondo](http://history.nasa.gov/SP-350/ch-13-4.html)[69](#page43) - si usted está tratando de hacer algo que nadie había hecho antes con Reaccionar nativo durante demasiado tiempo, podría tener sentido para acabar con eso y construir en la plataforma nativa relevante como vista personalizada.

**Envolviendo Componentes existentes**

Es posible que desee seleccionar un control que ha estado usando en su trabajo nativa por un tiempo, pero eso no existe para Reaccionar nativo. Si desea, puede poner el tiempo y esfuerzo de “envolver” de modo que estará disponible para usted en la tierra Javascript. Si lo hace, probablemente sería muy amable de su parte[compartirlo con la comunidad](https://github.com/jondot/awesome-react-native)[70](#page43) :).

**Un general escotilla de escape**

Cada solución arquitectónica cerrada debe tener vías de escape. Algunos pueden utilizar los puntos de fantasía “extensión” plazo, pero el razonamiento básico es el mismo:

1. <http://history.nasa.gov/SP-350/ch-13-4.html>
2. <https://github.com/jondot/awesome-react-native>

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 128 |

usted necesita estar seguro de que siempre tiene una solución en caso de necesitarla. Tener una puerta de escape nativa es poderoso porque es obvio que puede hacer todo lo que es posible hacer allí. Esto contribuye a la sensación de seguridad y el compromiso que se llevan al entrar en un nuevo proyecto, y definitivamente no es siempre cierto para otros móviles SDK multiplataforma. Además, esto también es cómo Facebook lo hizo.

**Cuando no ir nativo?**

Bueno, básicamente, la respuesta a esa recae en lo que estás haciendo. Si estás haciendo una aplicación estándar buscando, alguna variación de un maestro-detalle o una aplicación orientada documento, la UX regular o estándar (por medio de la plataforma móvil relevante), entonces por todos los medios tratar de evitar que los controles personalizados nativos. Echar un vistazo a la[Escaparate reaccionar nativo](https://facebook.github.io/react-native/showcase.html)[71](#page44) para averiguar si alguien “había hecho antes”.

Recuerde que cuando nos fijamos en el escaparate, ciertas aplicaciones son casos extremos - Chats, VR, Cámaras y efectos, y así sucesivamente son aplicaciones que probablemente tienen rendimiento o excepcionales en las preocupaciones ya exigen, y hay una posibilidad de que ellos mismos utilizan controles nativos . Así que cuando se intenta comparar la aplicación que desea construir con esos y juzgar la forma “estándar” que es, saber cuáles evitar cuando se comparan.

**Los bloques de construcción de una vista personalizada**

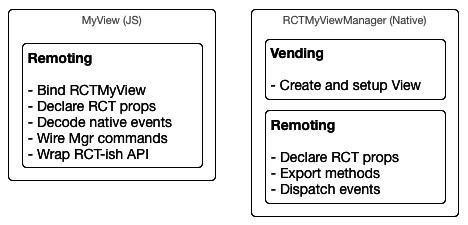
Si pensamos en ello, podemos definir lo que hace una vista personalizada de una manera sencilla:

* UI - La jerarquía de la interfaz de usuario nativa que se entregó a reaccionar nativo. Al final, esto puede ser cualquier cosa que se ajusta a unaUIView (IOS) o Ver(Androide); ya sea una vista personalizada de dibujo de su cuenta, o una vista compuesta con muchos subvistas.
* Puntales - La manera reaccionan siempre se esfuerza por entregar el estado. Incluso con la interfaz de usuario nativa, no hay manera apropiada para definir los apoyos Reaccionar-ish en su vista nativo, y dejar Reaccionar nativo encontrarlo.
* Eventos - Su punto de vista nativo puede emitir eventos. AlCambiarO cualquier otro evento que se pueda imaginar.
* Comandos - Menos Reaccionar-ful, un comando es una API que se sienta encima de su punto de vista, que ordena para hacer las cosas. Un ejemplo seríaScrollView's scrollTo. Es menos un “Reaccionar camino”, porque, obviamente, la posición de la barra de desplazamiento es también una especie de estado, y que los detalles específicos no se modela con una hélice. Habrá situaciones, en realidad, cuando se le necesita éstos.

1. <https://facebook.github.io/react-native/showcase.html>

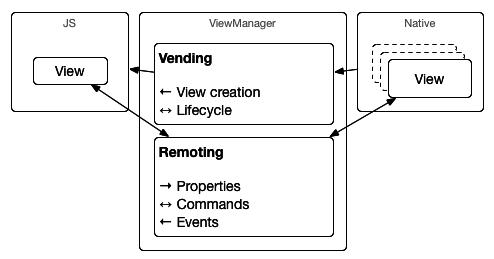
|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 129 |

Además, puede ayudar a mantener esta imagen mental en la mente (o en su escritorio), donde se muestra cada “mundo” - Javascript nativo y, y sus responsabilidades.



**el ViewManager**

Una vista personalizada nativa recibirá apoyos, emiten eventos, y aceptar com-mands. La forma Reaccionar colas nativos todos estos UP está creando el concepto de unaViewManager, Que es responsable de la creación de nuestra vista personalizada, el mantenimiento de instancias y el ciclo de vida de estos puntos de vista, y asegurarse de que los mensajes se pasan de ida y vuelta de nuestra Reaccionar nativo Javascript vista basada puramente y nuestros iOS nativas o ver Android.



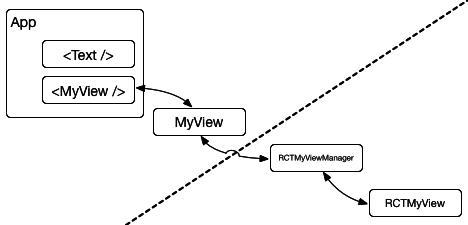
**Salvar fronteras de ViewManager**

Con este diagrama en mente, vamos a repasar lo que hace el ViewManager:

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 130 |

* Crear y configurar nuestro punto de vista nativo cuando es solicitado por Reaccionar nativo, mantener una instancia de ella de tal manera que no prestamos el alto precio de recrear ese punto de vista en cada render. Esto se conoce como “venta” de la vista.
* Implementar el puente establecer la propiedad para la vista administrado. A menudo, esto será sólo una simple delegación, pero a veces vamos a querer crear tipos personalizados a partir de los tipos simples que se están pasando a través de los límites remotos. A continuación, reflejar los cambios Reaccionar nativo de JavaScript vista del apoyo a través de estos de nuevo a la vista nativo que se está gestionando.
* En iOS, aplicar un puente de eventos de la vista conseguido Reaccionar despachador de nativos, y en Android este tipo de lógica también puede pertenecer a la propia vista conseguido si lo prefiere. A continuación, recibir eventos desde el punto de vista nativo y mover estos junto a la vista Reaccionar nativo Javascript través del despachador.
* En iOS, declarar comandos como un método exportado (más sobre esto más adelante), y en Android, definir los comandos disponibles e implementar un despachador de comandos que va a tender un puente sobre una orden dada a la vista administrado. Luego, aceptar el mando de React Javascript vista envoltorio de nativos y mover estos junto a la vista nativo.

Así que, básicamente, cada vez que su punto de vista nativo necesita comunicarse, estamos cruzando el abismo con la ayuda del administrador de vistas:



Ahora que ya sabe cómo este “vudú” funciona, va a ser divertido, más que extraña, al ver su funcionamiento.

**Nuestro Ejemplo: MessagesView**

El componente nativo ejemplo vamos a implementar es una vista mensajes. Una vista compuesta que tiene unaVista de la listaY una Etiqueta. Específicamente en IOS es una

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 131 |

UITableView y una UILabel y en Android es el rendimiento optimizado RecyclerView y un simple Vista de texto.

Nuestros mensajes ver voluntad:

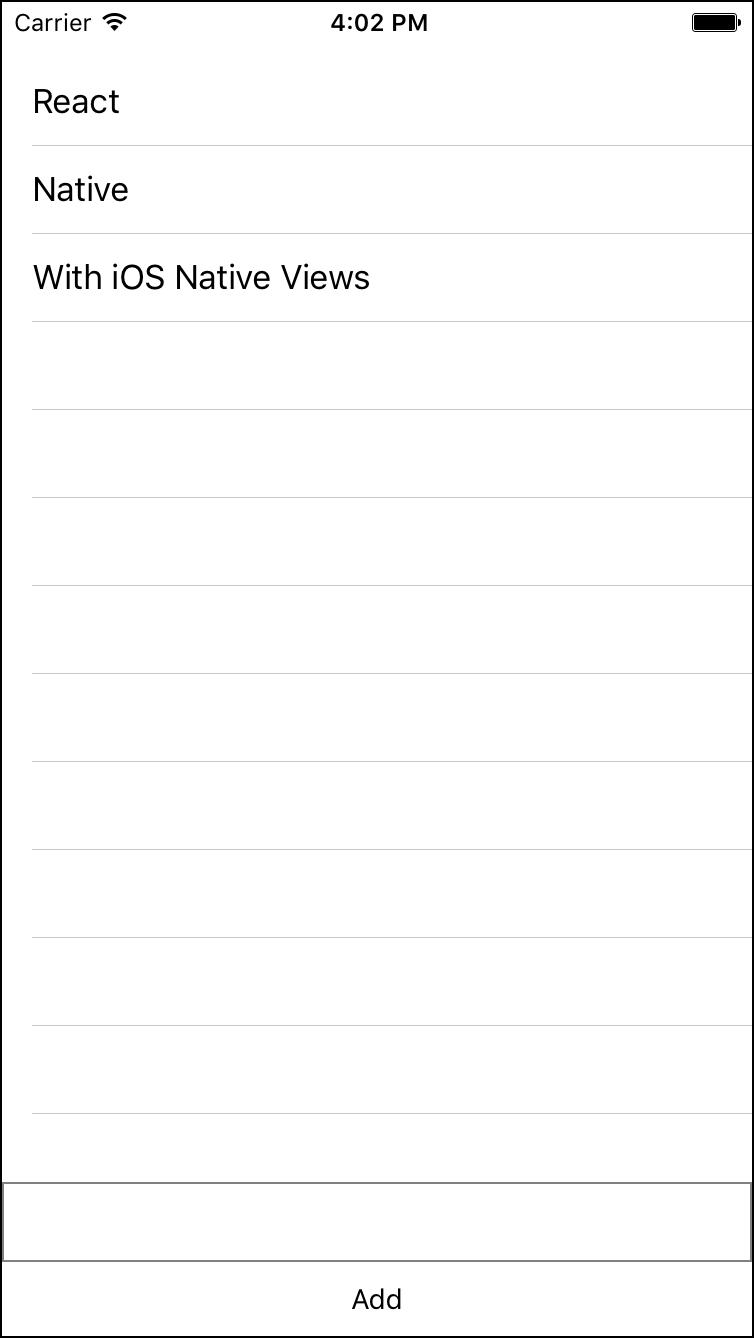
* mensajes mencionados de forma performant
* Desplazarse al último mensaje cuando se añade una nueva (Android)
* Enlazar la etiqueta a un último mensaje prop
* Aceptar un comando para agregar un mensaje (estamos modelando el estado de React que es una mala cosa, pero esto es sólo para demostración)

Este punto de vista mensajes puede ser una vista de conversación de chat, una vista de un elemento de lista de tareas, o lo que sea que se ajuste a la lista-e infinito--artículos patrón que a menudo presenta problemas de rendimiento para la prestación de la lista y una manipulación no trivial de lo que sólo funciona así cuando se rompe la abstracción y tener un alcance en la aplicación nativa.

A continuación, vamos a echar un vistazo a la nativecontrols muestrear y dividirla en sus partes entre sí.

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 132 |

**Desglose: iOS**



**Nuestros iOS híbridos + Reaccionar interfaz de usuario nativa**

A falta de un diseño adecuado, estos ejemplos están hechos para servir a un solo propósito: para que podamos averiguar cómo enjuague y repetición de envolver o crear nuestros propios componentes personalizados de nativos.

**MessagesView**

Hay varias maneras de construir vistas compuestas o personalizados en IOS. Programación, con Interface Builder, o una combinación. Vamos a utilizar una combinación: utilizamos Interface Builder para esbozar la interfaz de usuario, y modificarlo a través de código. Lo mejor es utilizar simplemente las muestras proporcionadas, en lugar construirlo desde cero si no tiene mucha experiencia con IOS hasta ahora, para más información sobre este tema bastante avanzada,[ver este gran guía](https://guides.codepath.com/ios/Custom-Views-Quickstart)[72](#page48).

1. <https://guides.codepath.com/ios/Custom-Views-Quickstart>

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 133 |

Algunas veces, ya tendrá una vista que simplemente querría terminar. En este caso, el procedimiento es exactamente lo mismo y nos centraremos en esa parte también.

En primer lugar, vamos a echar un vistazo a los trabajos internos de una vista personalizada tales. Vamos a tratar de señalar la extraña materia que sería poco familiar para cualquier persona que ha estado trabajando con IOS por un tiempo.

Como nota al margen - estaremos usando Swift para construir nuestro punto de vista, y Objective-C para el ViewManager.

* **importar** Fundación

2 **importar** UIKit

3

4 @MessagesViewDelegate protocolo objc: **clase**{

5func messagesView (messagesView: MessagesView, índice didSelectIndex: En t)

6 }

7

8 @objc **clase** MessagesView : UIView, UITableViewDelegate, UITableViewDataSource {

9@IBOutlet débil **var** tblView: UITableView!

1. @IBOutlet débil **var** lastMessageLbl: UILabel!
2. @IBOutlet **var** contentView: UIView!

12

1. **var** Ultimo mensaje:Cuerda {
2. obtener {
3. **regreso** lastMessageLbl.text!
4. }
5. conjunto (msg) {
6. lastMessageLbl.text=msg
7. }
8. }

21

1. **var** artículos:[Cuerda]=["Reaccionar","Nativo","Con iOS nativo Vistas"]
2. débiles **var** delegar: MessagesViewDelegate?
3. func addItem (punto: Cuerda) {
4. self.items.append (punto)
5. tblView.reloadData ()
6. }

30

1. func initViews () {
2. *// armar nuestra semilla / xib con la vista actual*
3. }

34

1. tableView func (tableView: UITableView, sección numberOfRowsInSection: En t) \
2. -> En t {

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 134 |

1. **regreso** self.items.count;
2. }

39

1. tableView func (tableView: UITableView, cellForRowAtIndexPath indexPath: NSI\
2. ndexPath) -> UITableViewCell {
3. **dejar** celda:UITableViewCell=self.tblView.dequeueReusableCellWithIdentifier\
4. ("celda")! como UITableViewCell
5. cell.textLabel?.texto = self.items [indexPath.row]
6. **regreso** celda
7. }

47

1. tableView func (tableView: UITableView, didSelectRowAtIndexPath indexPath: norte\
2. SIndexPath) {
3. delegar?.messagesView (auto como MessagesView, didSelectIndex: indexPath.ro\
4. w)
5. }
6. }

He quitado el código de la vista en el aparejo initViewsporque no es relevante (se puede echar un vistazo al código de ejemplo) y los constructores estándar por razones de brevedad. Por lo tanto, tomar un buen vistazo a este punto de vista. Ver nada Reaccionar nativo específico? La respuesta es que no hay ninguno. Para nosotros, esta es una vista en planta de iOS, y esto es una buena noticia ya que no hay abstracciones con fugas que ensucian nuestro punto de vista, y si teníamos vistas desde nuestra base de código existente - esto es una garantía de que nos gustaría probablemente no necesitaríamos modificarlos con el fin de cambiar la finalidad de ellos por Reaccionar nativo.

**RCTMessagesViewManager**

Aquí es donde la mayor parte de nuestra vida Reaccionar de código específico nativos. Y aun así, vamos a utilizar la infraestructura del Reaccionar nativo que utiliza misma por todo lo que usted ha utilizado hasta el momento (es decir,Texto, Ver y así sucesivamente), y veremos que se siente bastante al horno y no requiere que tanto la codificación.

Vamos a romper RCTMessagesViewManager en trozos y discutir cada pieza por separado.

* **@interfaz RCTMessagesViewManager**<MessagesViewDelegate>:RCTViewManager

2 **@fin**

Aquí obviamente queremos heredar de RCTViewManager que es el administrador vista base que reaccionan fuentes nativas, y se ajustan a MessagesViewDelegate que es, por convención, nuestra propia forma de decir que queremos que el punto de vista del hombre-ager a asumir la responsabilidad por las cosas que suceden en el interior MessagesView.

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 135 |

* **@implementación RCTMessagesViewManager**

2

3 RCT\_EXPORT\_MODULE ()

Utilizando RCT\_EXPORT\_MODULE estamos declarando nuestra aplicación de RCTMessagesView-Manager Reaccionar como un módulo nativo nativo. Esto es algo que va a hacer para cualquier componente que desea exponer a reaccionar nativo, ya sea un componente de interfaz de usuario o un módulo de servicio.

* - (UIView \*)ver

2 {

3MessagesView \*v = [MessagesView nuevo];

4v.delegate = yo;

5**regreso** v;

6 }

Esta es la parte “vending”. Aquí, además de crear una vista, estamos también su puesta en marcha con un delegado (que es el gestor de vista en sí).

Lo siguiente que vamos a través de los eventos, objetos y acciones bloques de construcción de un componente nativo.

* - (**vacío**) messagesView:(MessagesView \*)messagesView didSelectIndex:(**En t**)indexP\

2 ath

3 {

4NSDictionary \*evento = @ {

|  |  |
| --- | --- |
| 5 | @"objetivo": messagesView.reactTag, |
| 6 | @"índice": @ (IndexPath) |
| 7 | }; |

* [yo.bridge.eventDispatcher sendInputEventWithName:@ "TopChange" cuerpo: Incluso \ 9 t];

1. }

Comenzando con Eventos. Cuando declaramos nuestraRCTMessagesView como delegado MessagesView, Estábamos conformes con didSelectIndexcomo uno de los métodos delegados. En este caso, estamos llevando a cabo, y que al final queremos que nuestro punto de vista envoltorio Javascript para poder recibir un evento de nuestra nativaMessagesView componente.

Usamos topChange como un nombre de un evento que se refleja internamente a onChangedentro de la parte Javascript de nuestra vista. El siguiente, definimos la carga útil que se le pasa. Es muy importante para dar forma a esta carga útil adecuadamente, y no sólo lanzar un montón de valores clave / en la misma. Tiene unobjetivo clave que se debe dejar como está, con la coincidencia messagesView.reactTag valor, o anyOfYouViews.reactTag como ocurrirá con cualquier vista (recuerde que debe importar el archivos de cabecera correctos para obtener una bodega de que - aconsejan las muestras).

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 136 |

En este ejemplo, por razones de brevedad y la simplicidad del ejemplo nuestra carga útil es un int (indexPath). Para el caso general yo recomendaría dar forma a esta como un objeto estructurado, por ejemplo:

* "tipo": "Messagesview # didselectindex / 1"
* "carga útil" : {
* "llave": "valor"
* :
* :
* }

7 }

Dónde tipoes una directiva que apunta a qué método envió este evento. La manera que elegí para representar aquí es muy similar a cómo Rubí señala un método (#), y Erlang señala métodos con aridad (número de parámetros) en el nombre (/), para permitir sobrecargas, y para ayudar con el rastreo de nuevo a el método nativo justo en un momento de necesidad.

Si está utilizando Flux, este evento se debe formar al igual que sus eventos generados a partir de [acciones de flujo](https://github.com/acdlite/flux-standard-action)[73](#page52), y el tipo campo sería su SPECIFIC\_-REACT\_ACTION; que haría una arquitectura muy elegante. Por cierto, este Es así como los datos del ejemplo anterior se forma, con exclusión de la denominación de la acción disimulada como una manera de hacer una mejor experiencia de depuración.

* RCT\_CUSTOM\_VIEW\_PROPERTY (lastMessage, NSString, MessagesView)

2 {

3[ver setLastMessage: Json];

4 }

Pasando con Accesorios. Aquí es donde definimos nuestra prop: se llamaUltimo mensaje y así es como se os aparecerá en Javascript mundo, se escribe como una cadena (NSString) Y la vista de destino es una MessagesView que es exactamente lo que queremos.

Este es también el lugar donde ponemos en práctica el establecimiento de la hélice desde el mundo exterior a nuestro componente nativo. Tenga en cuenta que algo mágico está pasando aquí ya que tenemos dos variables mágicas:ver y JSONY, obviamente, estamos encargados de la fusión de los dos juntos. Así, en nuestro caso,JSON sería una NSString pero en otros casos puede ser un objeto más complejo que tendrá que [analizar gramaticalmente](https://github.com/facebook/react-native/blob/master/React/Views/RCTMapManager.m#L106) [y construir un objeto real estructurado](https://github.com/facebook/react-native/blob/master/React/Views/RCTMapManager.m#L106)[74](#page52) de exactamente aquí.

Si conoce el tipo de puntal es simple: String, Boolean, Integer, y así sucesivamente, se le permite al no proporcionar una parte de la implementación. Solo usaRCT\_EXPORT\_-VIEW\_PROPERTY con las mismas piezas de definición. Obviamente, se pueden definir todos sus propiedades y de esta manera llegan muy lejos.

1. <https://github.com/acdlite/flux-standard-action>
2. <https://github.com/facebook/react-native/blob/master/React/Views/RCTMapManager.m#L106>

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 137 |

* RCT\_EXPORT\_METHOD (añadir artículo: (No nulo NSNumber \*) reactTag

2 ít: (NSString \*)ít)

* {
* [yo.bridge.uiManager addUIBlock:
* ^(RCTUIManager \_\_unused \*UIManager, NSDictionary<NSNumber \*, MessagesView \ 6 \*> \*viewRegistry) {

7

8MessagesView \*ver = viewRegistry [reactTag];

9**Si** (!ver|| ![verisKindOfClass: [MessagesView **clase**]]) {

1. RCTLogError (@ "No encontraste RCTMessagesView con la etiqueta #% @", ReactTag);
2. **regreso**;
3. }

13

1. [ver añadir artículo: ít];
2. }];
3. }
4. **@fin**

terminando con comandos. Aquí es donde se define un comando que nos gustaría para exponer al mundo Javascript en una instancia de la vista que posee (utilizar unaárbitro para obtener una bodega de la misma). Mientras tanto complejo, si estrabismo, podremos para identificar una estructura muy simple:

* reactTag se pasa y se utiliza después para buscar a nuestro ejemplo de la Ver logrado
* código que se ejecuta es precedida por una pequeña ceremonia: El código debe ser suministrada a la UIManager lo que probablemente será decidido sobre qué cola de hilo para ejecutarlo.
* Cuando queremos ejecutar código en contra de nuestra vista (la orden), tenemos que encontrar en primer lugar, y que tenemos que hacer ese tipo de “manual”.

Así que si caminamos a través de esta pieza de código, vemos que estamos con el acceso a bridge.uiManager a instancia addUIBlock:

* [yo.bridge.uiManager addUIBlock:
* ^(RCTUIManager \_\_unused \*UIManager, NSDictionary<NSNumber \*, MessagesView \ 3 \*> \*viewRegistry) {

Mantener un fragmento de ese bloque de firma a sí mismo y sólo se aplican cada vez cuando sea necesario. A continuación, en el bloque estamos ir a buscar nuestro ejemplo de la especialviewRegistry variables, con nuestra reactTag (Que es sólo un número de internados):

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 138 |

* MessagesView \*ver = viewRegistry [reactTag];

Hacer el hula-hula requerido mediante la validación de la vista no sea nula y de la clase correcta, y que estamos.

* **Si** (!ver|| ![verisKindOfClass: [MessagesView **clase**]]) {
* RCTLogError (@ "No encontraste RCTMessagesView con la etiqueta #% @", ReactTag);
* **regreso**;

4}

Por último, llamar lo que quieras en el ver y utilizar los parámetros pasados ​​a través RCT\_EXPORT\_METHOD, en este caso ít.

* [ver añadir artículo: ít];

Esto completa nuestro análisis en profundidad de la ViewManager!. El siguiente, vamos a ver lo que se requiere para pegar una vista Javascript, o ver envoltorio en la misma.

**messagesview.ios.js**

Lo que queda por hacer ahora, es asegurarse de que creamos una vista Javascript, que sabe cómo enlazar a nuestra MessagesViewManagerY hacer nuestra asignación personalizada entre los eventos “en bruto” gestor de eventos y más bellas de Javascript.

Tenga en cuenta que puse el nombre de esta forma intencionada messagesview.ios.js, con el iosinfijo. Esto significa que este archivo sólo será visible en construye IOS, pero eso no quiere decir que usted debe construir una vista Javascript diferente para iOS y Android - por el contrario - que al final veremos que es prácticamente la misma, la cual lleva a casa por completo el hecho de que reaccionan nativo es tan grande a promover el intercambio de código entre plataformas.

La razón por la que empezar a cabo “pesimista” se debe a la intuición dicta que cuando está la construcción componentes nativos, no va a haber diferencias que son un poco irrelevante - UX, los apoyos personalizados, eventos o comandos. Que quiere dejar la escotilla abierta, y no optimizar prematuramente el código base forzándolo a ser la misma para ambas plataformas. Al final, cuando termino, puedo medir fácilmente la similitud entre los puntos de vista de iOS y Android Javascript, y entonces puedo tomar una decisión deliberada para fusionar el código base, así fácilmente identificar y separar las partes específicas de la plataforma quizás a otro Javascript módulo.

Ahora, vamos a repasar el bit solución a poco.

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 139 |

* **importar** Reaccionar,
* {RequireNativeComponent, Ver}

3de 'Reaccionar nativo'

4 **var** MessagesViewManager = exigir('' NativeModules) .MessagesViewManager

Aquí estamos tirando de la MessagesViewManager. Este es nuestro contacto en el nativoRCTMessagesViewManagerque acabamos de implementar. Reaccionar nativo se hizo cargo de los nombres. UsaremosrequireNativeComponent hacer la final y vinculante Ver pierde un set de accesorios requeridos directamente en nuestra vista personalizada.

* **clase** MessagesView **se extiende** React.Component {
* \_onChange (evento) {
* **esta**.props.onChange (evento)

4}

Recuerde que la nativa topChange¿parte? Aquí es donde estamos metiendo la mano en el caso de que hemos suministrado. Esta llamada interna\_onChangees responsable de hacer el mapeo en diversas quizás muchos tipos diferentes de eventos de cambio. Por lo tanto, la delegación general en lugar de directamente, como en este ejemplo, vamos a implementar una lógica de conmutación sobre eltipo de evento campo.

* addItem (elemento) {
* MessagesViewManager.addItem (
* React.findNodeHandle (**esta**),
* ít
* )

6}

Nuestra implementación de comandos. Recordemos que estamos aplicando métodos de Javascript tierra y espero que conseguir vigas hasta la tierra natal por medio de comandos. Esto significa que tenemos todo lo que hemos expuesto en nuestra nativaMessagesViewManagerY sólo tiene que acceder aquí. Tenga en cuenta que un especialfindNodeHandle (esto) hallazgo que se necesita en el orden y la viga hasta que reactTag que hemos estado hablando antes.

* render () {
* **regreso** <RCTMessagesView {...**esta**.props} onChange={**esta**.\_onChange.bind (thi\3 s)} />
* }

5 }

Esto debería ser trivial, estamos usando una RCTMessagesViewque vamos a crear en un par de líneas más, y el aparejo hasta accesorios y eventos. Hasta aquí todo bien.

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 140 |

* MessagesView.propTypes = {
* ... View.propTypes,
* Ultimo mensaje: React.PropTypes.string, 4 };

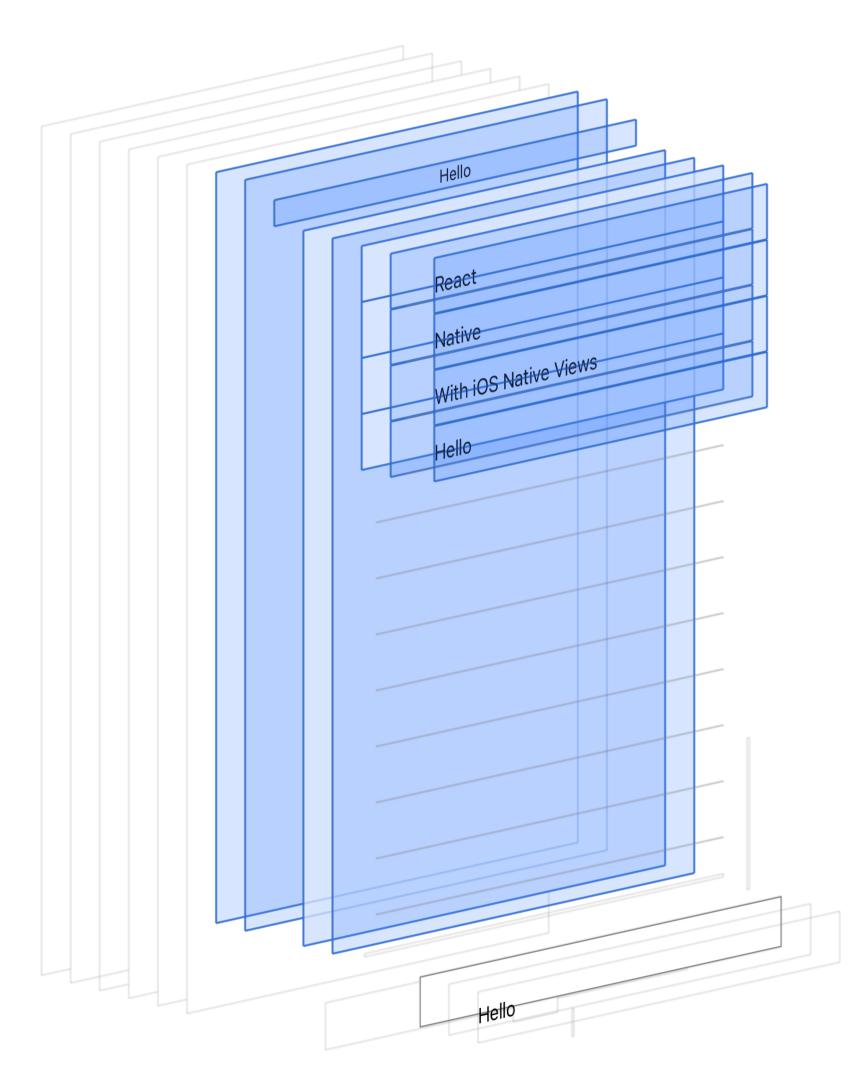
Al igual que con los normales Reaccionar código, estamos definiendo los apoyos requeridos y sus tipos. Sin embargo, tenga en cuenta que la suya es importante y no es opcional por lo que reaccionan nativo puede descubrir estos y atarlos.

* **var** RCTMessagesView=requireNativeComponent ('RCTMessagesView', MessagesView);

2 module.exports = MessagesView;

Por último, estamos llamando requireNativeComponent para asegurarse Reaccionar une nativos MessagesView y RCTMessagesView juntos. El objeto que se crea es el punto de vista real que podemos utilizar en nuestra hacermétodo. Tenga en cuenta que se siente un poco extraño para definir una clase, que utiliza una dependencia que sólo se define en el futuro, con la clase misma. Esta es una dependencia circular que en el software siempre es visto como malo, sin embargo aquí la semántica dictan que esto es en realidad lo que estamos haciendo - estos dos componentes son simbióticas.

Para refrescar la memoria, a continuación es el boceto cadena de llamadas de antes, y se puede casi ver cómo el componente nativo se basa en el componente Javascript y al revés. Esta imagen muestra lo maravilloso que la sinergia resulta ser:



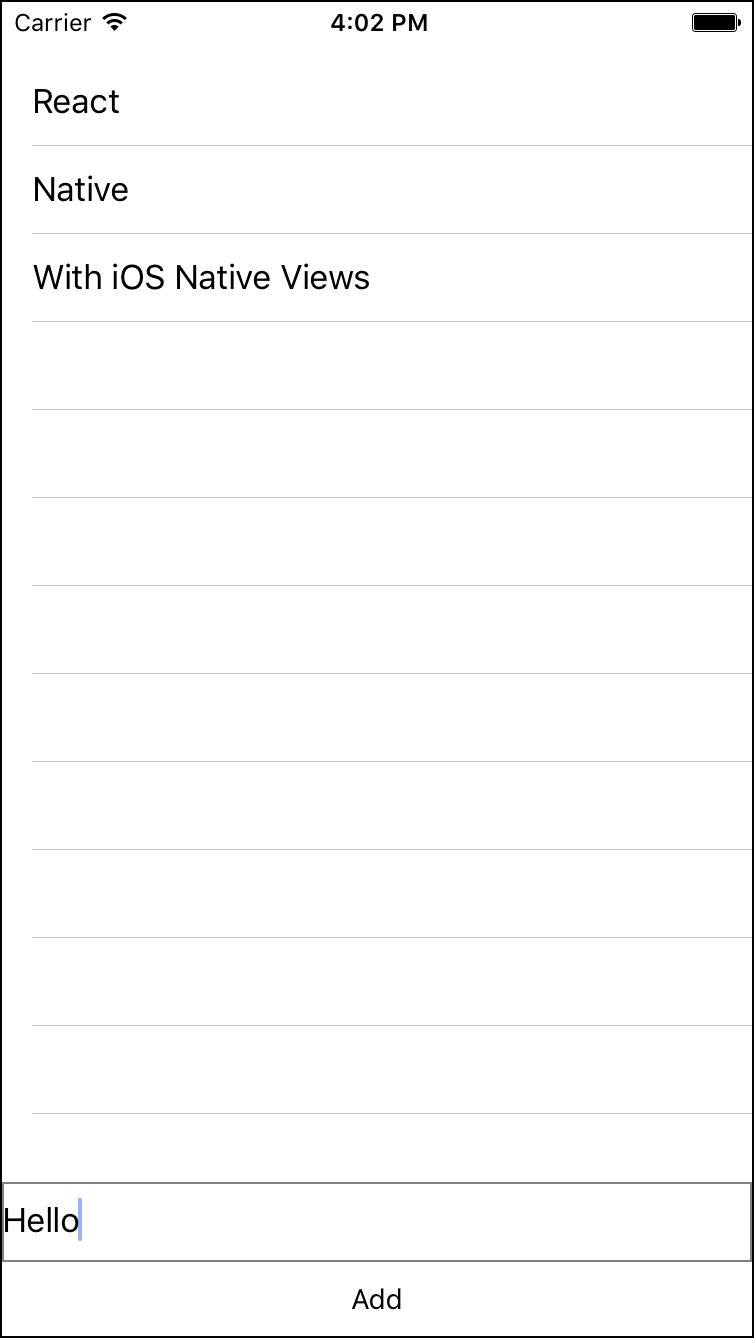
**Desglose de la interfaz de usuario de iOS: Nativo resaltado, Reaccionar nativo en blanco**

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 141 |

Con esto concluye toda nuestra travesía en el componente de interfaz de usuario nativa de iOS. Puede haber sentido “pesado” para aprender todas las cosas de bajo nivel, pero recuerda que reaccionan nativo es sólo a 0,19 (en el momento de la escritura), y en mi opinión, cuando se decida a adoptar una tecnología en dicha una de las primeras (relativamente) etapa, que debe ser capaz de razonar acerca de cómo funciona detrás de las escenas. Y lo más importante, usted debe tener ese componente nativo escotilla de escape cuando lo necesite.

Con todo, una vez que entras a través de éste tiempo y trabaja con el código de ejemplo, se sentirá seguro de que si se envuelve componentes de interfaz de usuario nativa, que suena y se siente avanzado, es realmente fácil y casi una cosa mecánica que hacer.

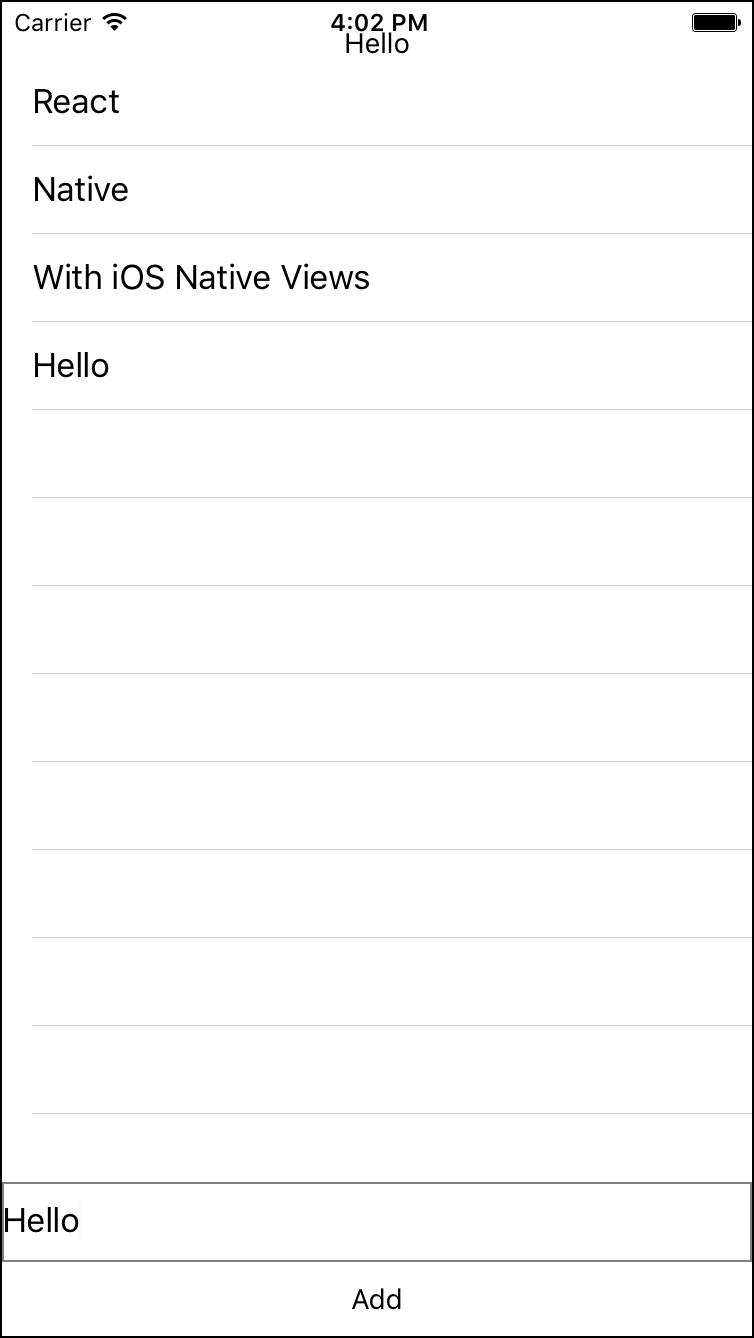
Con el tiempo, obtenemos lo siguiente:



**Reaccionar de entrada es nativo, diciendo “Hola”**

Y la adición de un artículo añade el artículo a nuestra lista de elementos, y la etiqueta:

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 142 |

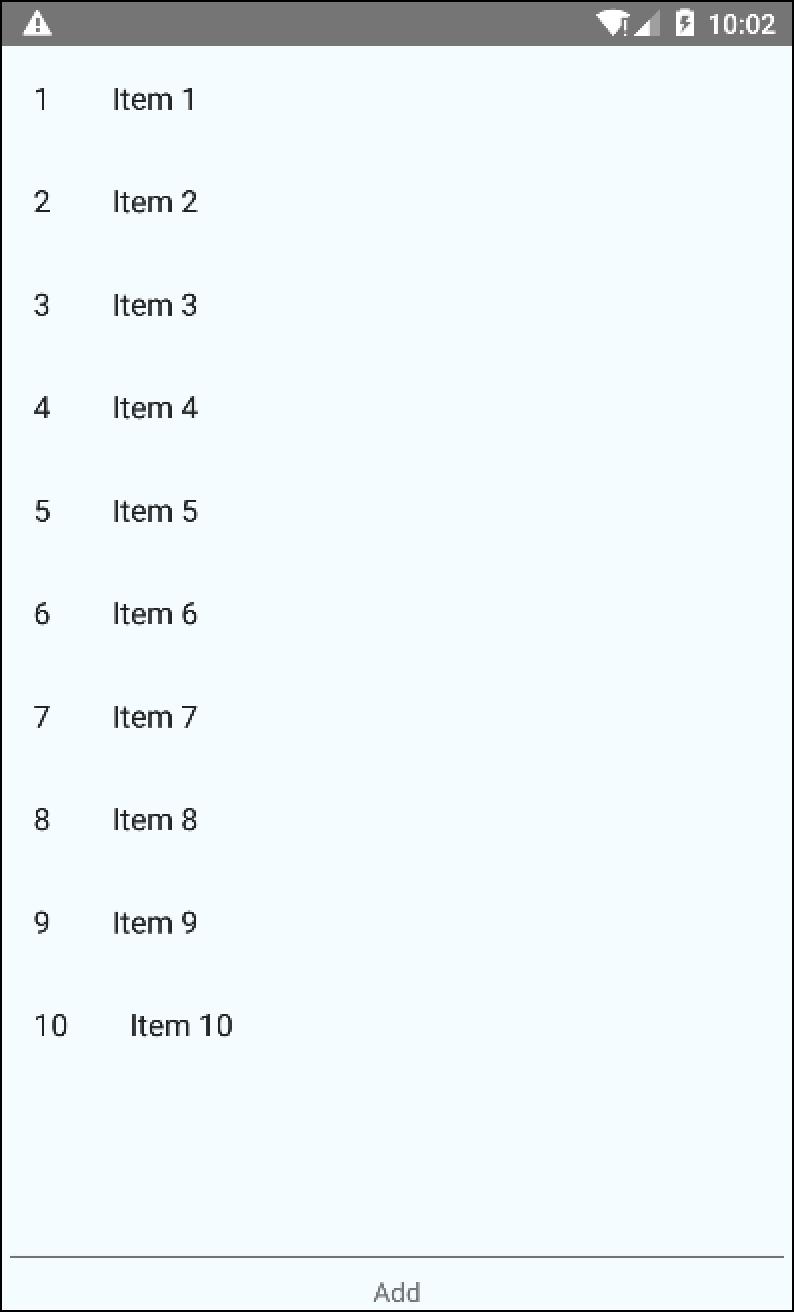


**Al hacer clic en “Añadir” añade un elemento**

¡Divertido!

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 143 |

**Desglose: Android**



**Nuestra Android híbrida + Reaccionar interfaz de usuario nativa**

Una vez más, observamos que estamos barato en el estilo de la interfaz de usuario y la fase de diseño. Recuerde: sólo se preocupan por encontrar la manera de hacer nuestros propios componentes personalizados de nativos aquí.

Vamos a echar un vistazo en nuestra MessagesView totalmente quemado, y como antes, tratar de identificar nada Reaccionar nativo específico que se ha filtrado en nuestra opinión.

**MessagesView**

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 144 |

* **clase pública MessagesView se extiende** LinearLayout **implementos** MessageViewDelegate\
* {
* **privado** Listview RecyclerView;

4**privado** Vista de Texto lastMessage;

5

6**público** MessagesView(contexto contexto) {

* **súper**(contexto);
* en eso();
* }

10

11**privado vacío** en eso() {

1. inflar(getContext(), R.diseño.messages\_view,**esta**);
2. **esta**.vista de la lista= (RecyclerView)findViewById(R.carné de identidad.vista de la lista);
3. **esta**.Ultimo mensaje= (Vista de texto)findViewById(R.carné de identidad.Ultimo mensaje);
4. LLM LinearLayoutManager = **nuevo** LinearLayoutManager(getContext());
5. LLM.setOrientation(LinearLayoutManager.VERTICAL);
6. vista de la lista.setLayoutManager(LLM);
7. vista de la lista.setAdapter(**nuevo** MessagesViewRecyclerViewAdapter(DummyContent.\
8. ARTÍCULOS, **esta**));
9. }

22

1. @Anular
2. **público vacío** onItemTap(DummyContent.DummyItemít) {
3. evento WritableMap = argumentos.createMap();
4. evento.putString("ít", ít.());
5. reactContext reactContext = (ReactContext)getContext();
6. reactContext.getJSModule(RCTEventEmitter.clase).receiveEvent(

|  |  |
| --- | --- |
| 29 | getId(), |
| 30 | "TopChange", |
| 31 | evento); |

1. }
2. **público vacío** añadir artículo(**final** contenido de la serie) {
3. DummyContent.ARTÍCULOS.añadir(**nuevo** DummyContent.DummyItem(contenido, contenido, do\
4. ontenido));
5. vista de la lista.getAdapter().notifyDataSetChanged();
6. **final En t** posición=DummyContent.ARTÍCULOS.tamaño() - 1;
7. RecyclerView.LayoutManager LayoutManager = vista de la lista.getLayoutManager(\
8. );
9. *//layoutManager.smoothScrollToPosition(listView, null, posición);*
10. LayoutManager.scrollToPosition(posición);
11. }

44

1. **público vacío** setLastMessage(msg cuerdas) {
2. Ultimo mensaje.setText(msg);

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 145 |

1. }
2. }

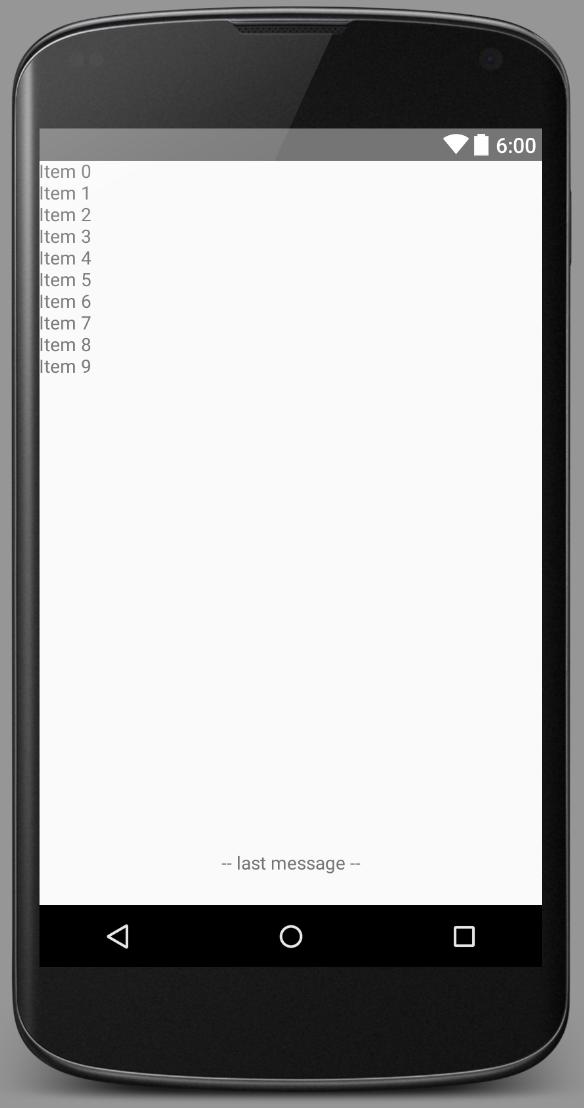
¿Ver cualquier cosa? Bueno, esta vez hay algo:onItemTapextiende la mano y tira de la Reaccionar emisor de sucesos nativo con el fin de empujar a cabo eventos. Ese detalle fue hecho intencionalmente, para mostrar que en Android puede hacer ambas cosas. Podemos salir de este código por conveniencia aquí, y también podemos sacarlo a nuestro gestor de vista (vamos a echar un vistazo a eso en un momento).

Me he tomado la libertad de usar las mismas expresiones idiomáticas como el código base IOS. Estamos utilizando unadelegar y nuestro MessagesViewmanos más que el delegado en nuestro adaptador. Algunos desarrolladores de Android se fruncir el ceño a esta selección de conceptos, pero que no viene del hecho de que yo soy un programador originalmente IOS - por el contrario, he empezado a hacer androide cuando fue primero y sólo unos pocos años más tarde hemos puesto en práctica la programación iOS nativa. Así que en realidad la idea aquí es que reaccionan nativo le permite adoptar incluso las mismas ideas como medios para un fin en diferentes plataformas, con tal de que funcione para usted y su equipo, ya que el quid de su productividad, idealmente ocurrir dentro Reaccionar nativo en sí.

Volver a nuestro código, de lo que estamos haciendo aquí es la creación de una [Ver compuesto](http://developer.android.com/guide/topics/ui/custom-components.html)[75](#page61), Que es vista compuesta de Android - una vista que contiene una o más subvistas. Por razones de conveniencia (o la pereza) construimos una vista raíz, que se infla un diseño que ha sido diseñado visualmente en Android Studio.

1. <http://developer.android.com/guide/topics/ui/custom-components.html>

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 146 |



**Android Studio UI Editor mostrando MessagesView**

Y este es el código de la vista de inicialización:

* **privado vacío** en eso() {
* inflar(getContext(), R.diseño.messages\_view,**esta**);
* **esta**.vista de la lista= (RecyclerView)findViewById(R.carné de identidad.vista de la lista);

4 **esta**.Ultimo mensaje = (Vista de texto)findViewById(R.carné de identidad.Ultimo mensaje); 5

6LLM LinearLayoutManager = **nuevo** LinearLayoutManager(getContext());

* LLM.setOrientation(LinearLayoutManager.VERTICAL);
* vista de la lista.setLayoutManager(LLM);
* vista de la lista.setAdapter(**nuevo** MessagesViewRecyclerViewAdapter(DummyContent.\ 10 ARTÍCULOS, **esta**));

11}

Siéntase libre para construir este nuevo mientras caminamos a través del código (si usted es un

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 147 |

desarrollador de Android en un principio), o simplemente utilizar las muestras proporcionadas. Puesto que estamos utilizando la fantasíaRecyclerView en nuestro diseño, tenemos que proporcionar un controlador de distribución, en este caso una LinearLayoutManager- y, en otras palabras, estamos haciendo una lista. el androideRecyclerView es capaz de optimizar una lista de las cosas en varios diseños (lista, rejilla, etc.) de manera que se pueda mover la basura los artículos que no se muestran con el fin de mejorar el rendimiento.

Lo que queda es como con la parte IOS, caminar a través de nuestros eventos, Puntales y

*comandos*. En ese orden, vamos a continuar con eventos:

* @Anular
* **público vacío** onItemTap(DummyContent.DummyItemít) {

3evento WritableMap = argumentos.createMap();

4evento.putString("ít", ít.());

5reactContext reactContext = (ReactContext)getContext();

* reactContext.getJSModule(RCTEventEmitter.clase).receiveEvent(

|  |  |
| --- | --- |
| 7 | getId(), |
| 8 | "TopChange", |
| 9 | evento); |

1. }

la aplicación de la MessagesViewDelegate interfaz, estamos chocando onItem-Tap cada vez que un usuario toca un elemento en nuestra lista. En ella, estamos construyendo nuestra carga útil que necesita ser transmitido de nuevo a la vista Javascript, y de nuevo aquí el mismo consejo es correcto - es aconsejable asegurarse de que la carga útil se ve estándar o con eficacia es una acción fundente si usted está en eso. Por el bien del ejemplo, la carga útil aquí no es más que una simple clave y valor para el elemento en sí.

El contexto en el que se te da dentro de la actividad o fragmento que incluso-tualmente tiene este punto de vista nativo, será un especial ReactContext, Por lo que cualquier momento que desee extender la mano y hacer algo Reaccionar nativo específico, asegúrese de caso con el contexto en lugar de la general de Android Contexto. Aquí estamos tirando de laRCTEventEmitter por lo que podemos dar instrucciones para que acepte un evento, y estamos suministrando una etiqueta de identificación con Reaccionar getId, Y de nuevo usando topChange con nuestros eventos

* esto debe ser muy similar pero un poco diferente al igual que con el ejemplo de iOS. Pasando a los apoyos, vamos a observar lo siguiente:
* **público vacío** setLastMessage(msg cuerdas) {
* Ultimo mensaje.setText(msg);

3}

Esto es algo que vamos a utilizar más tarde, cuando se introduce el gestor de vista, así que por ahora tener en cuenta que es sólo un regulador simple.

Y finalmente comandos, mira esto:

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 148 |

* **public void** añadir artículo(**final** Cuerda contenido) {
* DummyContent.ITEMS.add (**nuevo** DummyContent.DummyItem (contenido, contenido, c\ 3 ONTENIDO));
* listView.getAdapter () notifyDataSetChanged ().;
* **final int** posición=DummyContent.ITEMS.size ()- 1;
* RecyclerView.LayoutManager LayoutManager = listView.getLayoutManager (\

7 );

8*//layoutManager.smoothScrollToPosition(listView, null, posición);*

* layoutManager.scrollToPosition (posición);

10}

Este comando está expuesto a cabo como públicoy - usted lo adivinó - luego serán utilizados en nuestro gestor de vista. Tenga en cuenta que estamos llegando a nuestro a nuestro propio punto de vista, la infraestructura (adaptador) y nuestro gestor de diseño con el fin de realizar el desplazamiento. Comentada es unasmoothScrollvariante de desplazamiento - pienso en lo difícil sería crear las distintas Reaccionar nativo de una manera performant? Si quieres una pista, echar un vistazo[en esta discusión](https://github.com/FaridSafi/react-native-gifted-messenger/issues/3)[76](#page64) en el excelente dotados-mensajero proyecto.

**MessagesViewManager**

Vamos a romper nuestra MessagesViewManager, poco a poco (se puede echar un vistazo a la clase completa en las muestras proporcionadas):

* **clase pública MessagesViewManager se extiende** SimpleViewManager<MessagesView> {

2**public static final** cadena REACT\_CLASS="RCTMessagesView";

3 **static final privado** **En t** CMD\_ADDITEM = 1; 4

* @Anular
* **público** CuerdagetNombre() {

7**regreso** REACT\_CLASS;

8}

Así es como empezamos. Heredar deSimpleViewManager, Que lleva nuestra MessagesView como un parámetro genérico, y definir diversas constantes para su uso posterior. REACT\_-CLASE para suministrar a través getNombrey CMD\_ADDITEM para nuestro mapa de mando (más sobre esto más adelante). Esto es sólo una ceremonia para cumplir con el puente que reaccionan nativo va a querer hacer en nuestro nombre.

1. <https://github.com/FaridSafi/react-native-gifted-messenger/issues/3>

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 149 |

* @Anular
* **protegido** MessagesViewcreateViewInstance(ThemedReactContext reactContext\

3 ) {

4**return new** MessagesView(reactContext);

5}

Aquí vend nuestro punto de vista, al igual que con el ejemplo de iOS. Nada demasiado emocionante excepto que estamos ante unaThemedReactContext que es especial (Os animo a echar un vistazo y ver lo diferente que es de un habitual Contexto).

* @ReactProp(nombre="Ultimo mensaje")
* **público vacío** setLastMessage(Ver MessagesView,@Nullablemsg cuerdas) {
* ver.setLastMessage(msg);

4}

Moviéndose a Accesorios, Así es como exponemos un puntal en la versión de Android de React nativo. Mientras que una opinión personal - que tiene un aspecto más limpio. Obviamente,Ultimo mensaje es el nombre de la hélice que estamos exponiendo aquí.

* @Anular
* **público vacío** receiveCommand(raíz MessagesView, **En t** CommandID,@NullableRe\3 args adableArray) {
* **cambiar**(CommandID) {

|  |  |
| --- | --- |
| 5 | **caso** CMD\_ADDITEM: |
| 6 | raíz.añadir artículo(args.getString(0)); |

* }
* **súper**.receiveCommand(raíz,CommandID,args);
* }

10

1. @Nullable
2. @Anular
3. **público** Mapa<Cuerda,Entero>getCommandsMap() {
4. **regreso** MapBuilder.de(
5. );
6. }
7. }

Por último, los comandos. Estamos diciendo dos cosas aquí. En primer lugar, estamos implementando un aspecto genéricoreceiveCommandque aceptará una vista, un identificador de comando, y cero o más argumentos. Nuestro trabajo es asegurarnos de que asignar cadaCommandIDa su comando real (¿alguien dijo las acciones de flujo? :-). En segundo lugar, tenemos que decirle Reaccionar nativo lo que estamos permitiendo que los comandos y cuáles son sus nombres y valores, y esto se hace a través degetCommandsMap. Esta es la forma en que puede asegurarse de que nadie le da una desconocidaCommandID en receiveCommand.

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 150 |

**NativeControlsPackage**

En el mundo de Android Reaccionar nativo, tenemos que decirle explícitamente cómo encontrar nuestro MessagesViewManager.Esto se hace mejor mediante el suministro de un ReactPackage, que se puede ver en como un conjunto de módulos personalizados que desea Reaccionar nativo a conocer. También es aconsejable invertir y hacer su propioNativeControlsPackage que especificará este MessagesViewManager así como cualquier futuros gestores Lo que quiere construir.

* **clase pública NativeControlsPackage implementos** ReactPackage{
* @Anular
* **público** Lista<NativeModule>createNativeModules(rea ReactApplicationContext\4 ctContext) {

5**regreso** Colecciones.lista vacía();

* }

7

* @Anular
* **público** Lista<Clase<? **se extiende** JavaScriptModule>>createJSModules() {

10**regreso** Colecciones.lista vacía();

1. }
2. @Anular
3. **público** Lista<ViewManager>createViewManagers(ReactApplicationContext reaccionar\
4. Contexto) {
5. **regreso** Las matrices. <ViewManager>asList(**nuevo** MessagesViewManager());
6. }
7. }

El único lugar que nos importa hasta el momento, es createViewManagers donde volvemos una lista con un solo elemento - nuestra MessagesViewManagerinstanciado. Si quieres ver un verdaderoReactPackage, Echar un vistazo a la MainReactPackage Reaccionar clase dentro nativo en sí:

* **clase pública MainReactPackage implementos** ReactPackage{

2

* @Anular
* **público** Lista<NativeModule>createNativeModules(ReactApplicationContext reaccionar\5 Contexto) {

6**regreso** Las matrices. <NativeModule>asList(

7**nuevo** AsyncStorageModule(reactContext),

8**nuevo** ClipboardModule(reactContext),

9**nuevo** DialogModule(reactContext),

1. **nuevo** FrescoModule(reactContext),
2. **nuevo** IntentModule(reactContext),
3. **nuevo** LocationModule(reactContext),

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 151 |

1. **nuevo** NetworkingModule(reactContext),
2. **nuevo** NetInfoModule(reactContext),
3. **nuevo** WebSocketModule(reactContext),
4. **nuevo** ToastModule(reactContext));
5. }

18

1. @Anular
2. **público** Lista<Clase<? **se extiende** JavaScriptModule>>createJSModules() {
3. **regreso** Colecciones.lista vacía();
4. }

23

1. @Anular
2. **público** Lista<ViewManager>createViewManagers(ReactApplicationContext reactCo\
3. ntext) {
4. **regreso** Las matrices. <ViewManager>asList(
5. ARTRenderableViewManager.createARTGroupViewManager(),
6. ARTRenderableViewManager.createARTShapeViewManager(),
7. ARTRenderableViewManager.createARTTextViewManager(),
8. **nuevo** ARTSurfaceViewManager(),
9. **nuevo** ReactDrawerLayoutManager(),
10. **nuevo** ReactHorizontalScrollViewManager(),
11. **nuevo** ReactImageManager(),
12. **nuevo** ReactProgressBarViewManager(),
13. **nuevo** ReactRawTextManager(),
14. **nuevo** RecyclerViewBackedScrollViewManager(),
15. **nuevo** ReactScrollViewManager(),
16. **nuevo** ReactSwitchManager(),
17. **nuevo** ReactTextInputManager(),
18. **nuevo** ReactTextViewManager(),
19. **nuevo** ReactToolbarManager(),
20. **nuevo** ReactViewManager(),
21. **nuevo** ReactViewPagerManager(),
22. **nuevo** ReactTextInlineImageViewManager(),
23. **nuevo** ReactVirtualTextViewManager(),
24. **nuevo** SwipeRefreshLayoutManager(),
25. **nuevo** ReactWebViewManager());
26. }
27. }

Obviamente hay mucho más que hacer allí, pero también se puede verificar a sí mismo que está utilizando la misma tecnología Reaccionar nativo está utilizando detrás de las escenas con el fin de extenderlo. El equipo Reaccionar nativo nos está mostrando dogfooding en su mejor!

Con esto, llegamos a la conclusión de nuestra discusión del gestor de vista Android.

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 152 |

**messagesview.android.js**

Vamos a pasar al mundo Javascript, y ver cómo enlazar ese punto de vista de nuestra opinión componentes nativos.

Estoy confiadamente que indica esto: tanto el .ios y .androideVer versiones de mes-sagesview pueden ser (bastante sorprendentemente) convergió a la misma clase! (O un archivo o base de código). Estoy manteniendo una.androide versión aquí para que podamos probar esto a nosotros mismos, ahora, vamos a ver cómo se desarrolla esta por romper el MessagesView clase (de nuevo, se puede obtener la lista completa de las muestras proporcionadas):

* **importar** Reaccionar,
* {RequireNativeComponent,

3PropTypes}

4 de 'Reaccionar nativo'

5 **importar** UIManager de 'UIManager'

6 **importar** Visto desde 'Ver'

Nada nuevo aquí, por lo mueve a lo largo.

* **clase** MessagesView **se extiende** React.Component {
* constructor (apoyos) {
* **súper**(accesorios)
* **esta**.\_onChange= **esta**.\_onChange.bind (**esta**)

5}

Aún así, nada nuevo. Vamos a echar un vistazo a\_onChange:

* \_onChange (evento) {
* **Si**(!**esta**.props.onItemTapped) {
* **regreso**
* }
* **esta**.props.onItemTapped (event.item)

6}

Aún no hay cambio, sólo estamos mostrando un poco de un sabor diferente. Estamos nombrando el puntal de destino diferente y asegurarse de que existe la hélice.

* addItem (elemento) {
* UIManager.dispatchViewManagerCommand (
* React.findNodeHandle (**esta**),
* UIManager.RCTMessagesView.Commands.addItem,
* [ ít ])

6}

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 153 |

Aquí está nuestra primera diferencia práctica entre el .androide y .iosversiones. Estamos utilizando unaUIManager.dispatchViewManagerCommand con el fin de transmitir nuestra com-Mand (en IOS, se utilizó la MessagesViewManager módulo nativo que requiere de la nativos-módulos paquete).

Si quisiéramos hacer converger el código base, ambas versiones se pudieron extraer fácilmente con una clase de servicio que podría haber sido llamado comandos.

* hacer(){
* **regreso**(
* <RCTMessagesView {...**esta**.props} onChange={**esta**.\_onChange} />
* )
* }

6

7 }

8 MessagesView.propTypes = {

9... View.propTypes,

1. onItemAdded: PropTypes.func,
2. Ultimo mensaje: PropTypes.string,
3. }

13

1. **dejar** RCTMessagesView=requireNativeComponent ('RCTMessagesView',

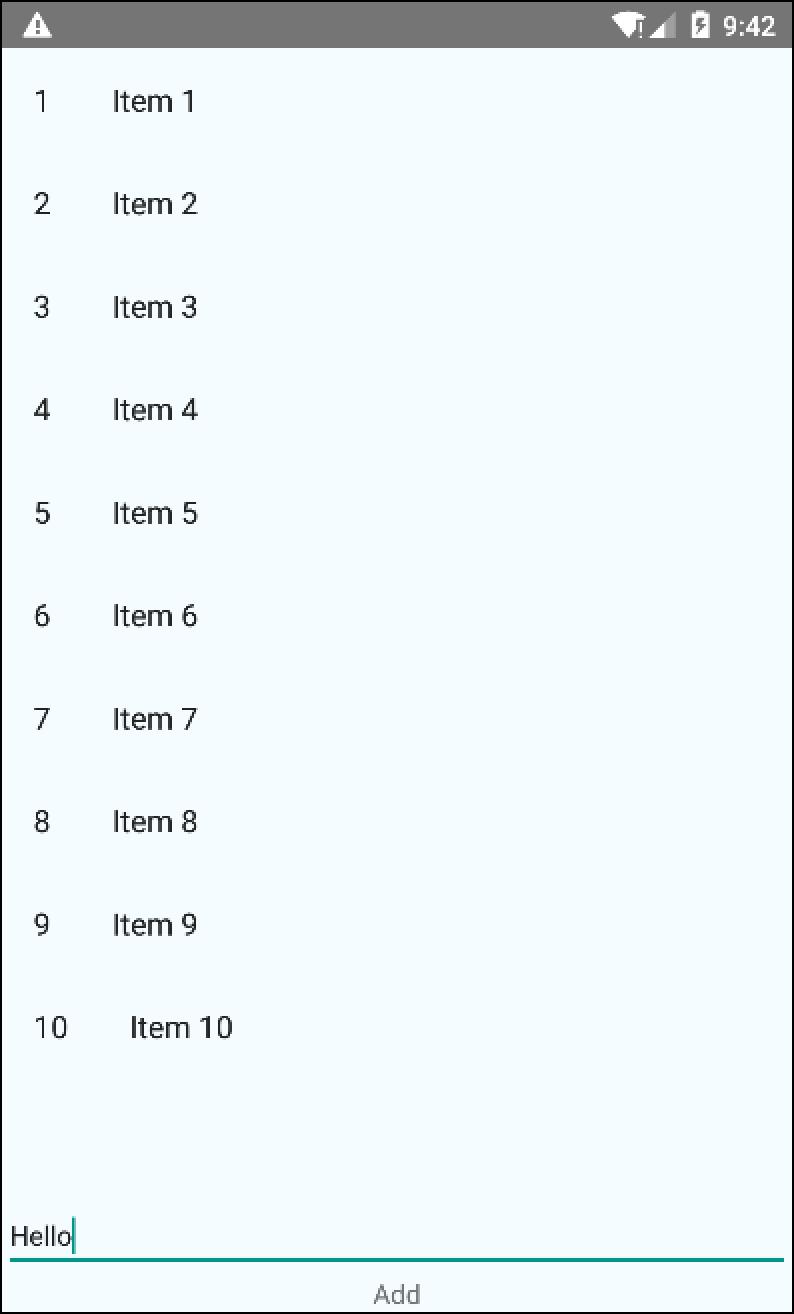
|  |  |
| --- | --- |
| 15 | MessagesView, |
| dieciséis | {nativeOnly:{onChange:**cierto**}}); |

1. module.exports = MessagesView

La última parte muestra que de nuevo, un mínimo o ninguna diferencia entre tanto el .ios y .androideversiones existe. Sólo por el bien del ejemplo, aquíUltimo mensaje está unido a través de los apoyos. También, propTypes se establece de manera diferente. Ninguno de estos son motivo de ruptura para converger el código, y algunos de ellos están aquí sólo para demostrar que tiene más de una manera de hacer las cosas.

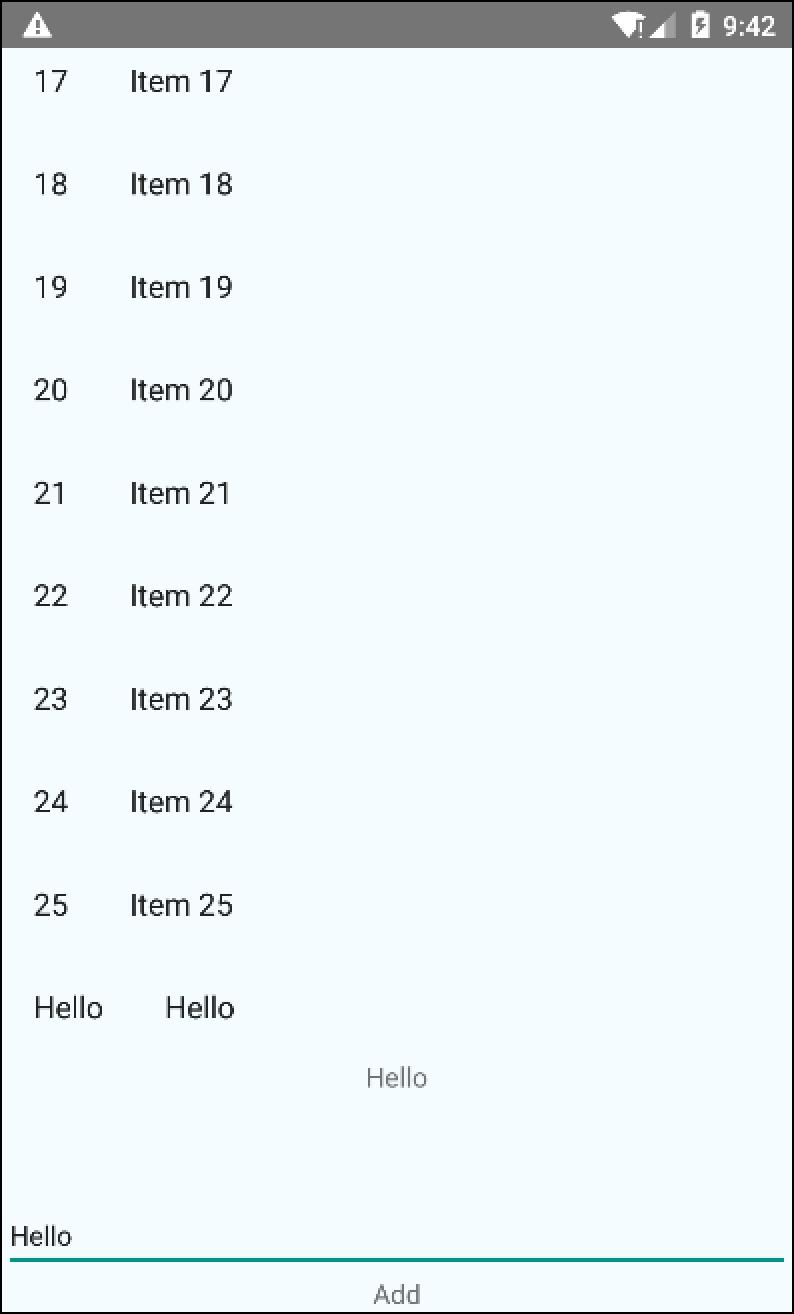
Y aquí es cómo todo se comporta, en Android:

Going Native: interfaz de usuario nativa 154



**Reaccionar de entrada es nativo, diciendo “Hola”**

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 155 |



**Al hacer clic en “Añadir” añade un elemento**

**usando MessagesView**

Y ahora, a disfrutar de la cereza en la parte superior. UsarMessagesView en nuestros puntos de vista de contenedores, podemos tratarlo como cualquier otro punto de vista, e incluso mejor, llegar a mantener una única base de código tanto para Android y iOS, utilizando una implementación nativa diferente a la vez!

Así es como se ve:

* <MessagesView
* árbitro="mensajes"
* Ultimo mensaje={**esta**.state.lastMessage}
* estilo={{flexionar: 1}}
* onChange={(Ev)=>console.log (ev)} />

Y, en cualquier lugar desea, puede enviar comandos a este punto de vista, así:

|  |  |
| --- | --- |
| Going Native: interfaz de usuario nativa | 156 |

* agregar mensaje(){
* **esta**.refs.messages.addItem (**esta**.state.text)
* **esta**.setState ({lastMessage:**esta**.state.text})

4 }

**Resumen**

En este capítulo, hemos cubierto uno de los temas más empedernidos de React nativo, este es un tema que afecta a cómo reaccionan los nativos en sí está construido y le permite hacer un buen uso de los mismos bloques de construcción con el fin de hacer su propio trampillas de evacuación, y asegurarse de que su aplicación lleva a cabo mediante la implementación de la interfaz de usuario de forma nativa por plataforma, si de repente se encuentra que Reaccionar nativo no tiene todas las respuestas.

Hemos aprendido que para hacer este tipo de puente, Reaccionar nativo espera:

* Una vista personalizada de su cuenta, por plataforma
* Un gestor de vista que tiene ese punto de vista, y se encarga de la venta de su
* Interacción remota y los conceptos de: Puntales, Eventos y Comandos y asegurándose de que el gestor de vista sabe cómo pegar estas arriba.

Y eso es básicamente la misma. Si te gusta para marcar cosas hasta once, asegúrese de leer el código de[ScrollView y WebView](https://github.com/facebook/react-native/tree/master/React/Views)[77](#page72) para iOS, y [lo mismo en](https://github.com/facebook/react-native/tree/master/ReactAndroid/src/main/java/com/facebook/react/views) [Androide](https://github.com/facebook/react-native/tree/master/ReactAndroid/src/main/java/com/facebook/react/views)[78](#page72).

1. <https://github.com/facebook/react-native/tree/master/React/Views>
2. [https://github.com/facebook/react-native/tree/master/ReactAndroid/src/main/java/com/facebook/react/](https://github.com/facebook/react-native/tree/master/ReactAndroid/src/main/java/com/facebook/react/views) [puntos de vista](https://github.com/facebook/react-native/tree/master/ReactAndroid/src/main/java/com/facebook/react/views)

**Going Native: Nativo Módulos**

En este capítulo veremos cómo exponer las capacidades nativas de la plataforma en iOS y Android, y que se puede poner a disposición de su Javascript mediante la construcción y la exposición de un módulo nativo reaccionar.

Si no ha leído la anterior Going Native: capítulo nativo de interfaz de usuario, por favor tome el tiempo para hacerlo ahora. Usted no tiene que hacerlo, pero incluso si no está la construcción de un componente de interfaz de usuario nativa, gran parte del material en ese capítulo es relevante, ya que por medio de la construcción de la componente de interfaz de usuario nativa tuvimos que construir también un módulo nativo.

Si realmente sólo quiere hacer un módulo de lo explicaré todo, así - por lo que no se preocupe; cuando sientes que quieres saber cómo funciona todo detrás de las escenas, revisar el capítulo anterior.

**Nuestro Ejemplo: cryptboard**

Puesto que ya hemos pasado por un camino bastante largo en el ejemplo anterior, cada cosa que quieres hacer ya debería estar claro para usted en términos de fontanería Reaccionar nativo y la interacción remota. Todo lo que debe faltar es para validar que la construcción de un módulo es exactamente lo mismo, y para ver cómo montar un módulo de este tipo en la infraestructura de aplicaciones.

Dado que los módulos son fáciles de hacer, muchos, si no todos los servicios nativos comunes, tales como contactos, geolocalización, notificaciones, almacenamiento, etc. ya están cubiertos ya sea por Reaccionar o nativa por la comunidad y que probablemente no se encontraría hacer una desde cero. Es posible, cómo la historia, encontrará mejorando o modificando uno y por eso, el conocimiento que ha adquirido en el capítulo anterior debería ser suficiente.

Así que teniendo en cuenta que, vamos a exponer el portapapeles nativo como un módulo con un toque - se almacenará el contenido cifrado usando un algoritmo AES. Queremos extender la mano y agarrar el contenido del portapapeles desde iOS o Android y, a continuación, ser capaz de cifrar y establecer un contenido de nuestra propia. Al igual que con cualquier módulo, estaremos acceder a esto como una API Javascript puro sin interfaz de usuario.

El código será muy idéntica a la existente RCTClipboardmódulo ya provisto de Reaccionar; Si en algún momento desea comparar y ver cómo un módulo portapapeles en texto claro se comportaría :-).

En este ejemplo, vamos a insistir en tener una sola base de código JavaScript, lo que significa tanto para iOS y Android se cargarán los mismos ficheros, sin .ios o .androide calificadores!

157

NSOb-

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 158 |

**Desglose iOS**

Vamos a mostrar algunas cosas en iOS:

* La construcción de un módulo
* Configuración del proyecto para bibliotecas externas con Cocoapods
* Señalando el proyecto en una sola índice paquete, no .ios índice

Vamos a empezar con la adición de una AESbiblioteca capaz. He elegido[AESCrypt](https://github.com/Gurpartap/AESCrypt-ObjC)[79](#page74) para empezar por su sencillez. Usando Cocoapods vamos a hacer esto:

* [Nativemodules] ios $ cd
* [Nativemodules / ios] init $ vaina

Y modifique el PODFILE siendo éste:

* objetivo '' nativemodules **hacer**

2vaina 'AESCrypt'

3 **fin**

4

5 objetivo '' nativemodulesTests **hacer**

6

7 **fin**

finalmente ejecutar vaina de instalar y abra el recién generado nativemodules.xcworkspace en lugar de la existente nativemodules.xcproject. Ahora debería tener acceso a laAESCrypt biblioteca a partir del código de Objective-C.

Vamos a proceder con nuestra nativa cryptboard módulo:

* *// RCTCryptboard.m*

2

3 *import "RCTBridgeModule.h"*

4 *import "RCTUtils.h"*

5 *import <UIKit / UIKit.h>*

6 *import "AESCrypt.h"*

7

8

9 *#define CRYPT\_PASS @ "jondot / impresionante reaccionan de forma nativa"*

10

1. **@interfaz RCTCryptboard** :NSObject <RCTBridgeModule>
2. **@fin**

Hasta ahora, si has leído el capítulo anterior esto debe estar familiarizado. Si no es así, esto es sólo una ceremonia que tenemos que realizar: heredar de

Ject <RCTBridgeModule>. También importamos el puente, Utiles, y nuestra AESCrypt encabezados

1. <https://github.com/Gurpartap/AESCrypt-ObjC>

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 159 |

* **@implementación RCTCryptboard**

2

3 RCT\_EXPORT\_MODULE ()

4

5 - (Dispatch\_queue\_t) methodQueue

6 {

7**regreso** dispatch\_get\_main\_queue ();

8 }

Con esto, estamos registrar nuestro módulo y la narración de qué cola hilo Reaccionar nativo debe utilizar. No todas las tareas son iguales y tenemos la libertad de decir lo que Reaccionar nativo prioridad para ejecutar las tareas en la cola mediante el suministro de coincidencia. Esto debe tener sentido para usted si usted es originalmente un desarrollador de iOS, y si no, le recomiendo leer acerca de las diversas colas de despacho iOS ofrece y el modelo de hilos y el razonamiento[detrás de estos](https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html) [aquí](https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html)[80](#page75).

* RCT\_EXPORT\_METHOD (conjunto:(NSString \*)contenido)

2 {

3UIPasteboard \*portapapeles = [UIPasteboard generalPasteboard];

4 clipboard.string = [AESCrypt cifrar:contenido contraseña: CRYPT\_PASS]; 5 } 6

7 RCT\_EXPORT\_METHOD (obtener: (RCTPromiseResolveBlock) determinación

8 que rechaza: (\_\_ RCTPromiseRejectBlock sin usar) rechazar)

* {

1. UIPasteboard \*portapapeles = [UIPasteboard generalPasteboard];
2. determinación (@ [RCTNullIfNil ([AESCrypt desencriptar: clipboard.string contraseña: CRYPT\_PA \
3. SS])]);
4. }
5. **@fin**

Esto también debe ser muy familiar, estamos usando el mismo EXPORT\_METHODmacros que hemos usado antes, y éstos reaccionan contamos nativo para exponer nuestros métodos en el puente. Tenga en cuenta que laobtener método hace uso de RCTPromiseResolveBlock que aparecerá como una promesa común de JavaScript en el otro extremo.

También estamos haciendo uso de la AESCrypt biblioteca para cifrar los datos del portapapeles, pero eso no es todo relacionado con Reaccionar nativo en sí.

Eso es. Desde mi experiencia en la elaboración de un módulo es muy fácil, y se siente más agradable que envolver un componente nativo en toda regla con la interfaz de usuario y la lógica.

Puesto que estamos utilizando una única base de código Javascript para ambas plataformas en esta ocasión, lo dejaremos para el final como un postre. Vamos a pasar a la parte de Android.

1. [https://developer.apple.com/library/ios/documentation/General/Conceptual/](https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html)

[ConcurrencyProgrammingGuide / OperationQueues / OperationQueues.html](https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html)

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 160 |

**Desglose Android**

Vamos a repasar la parte de Java. En mi opinión se ve más limpio y más estructurada que nuestra contraparte IOS.

* *// CryptboardModule.java*
* **clase pública CryptboardModule se extiende** ReactContextBaseJavaModule{

3 **final privado** cadena CRYPT\_PASS = "/ Jondot impresionante reaccionan de forma nativa"; 4

5**público** CryptboardModule(ReactApplicationContext reactContext) {

* **súper**(reactContext);
* }

8

* @Anular

1. **público** CuerdagetNombre() {
2. **regreso** "Cryptboard";
3. }

13

1. **privado** ClipboardManagergetClipboardService() {
2. **regreso** (ClipboardManager)getReactApplicationContext().getSystemServi\
3. ce(getReactApplicationContext().CLIPBOARD\_SERVICE);
4. }

Para la inicialización, la parte más importante es ampliar ReactContextBase-JavaModule y para proporcionar un nombre de módulo, aquí elegimos cryptboard de modo que coincidirá deliberadamente la parte IOS. El resto es el patrón típico de Android agarrar un servicio del sistema, tales como el portapapeles.

* @ReactMethod
* **público vacío** obtener(Promesa promesa) {
* **tratar** {

|  |  |
| --- | --- |
| 4 | portapapeles ClipboardManager = getClipboardService(); |
| 5 | ClipData clipData = portapapeles.getPrimaryClip(); |
| 6 | **Si** (clipData== **nulo**) { |
| 7 | promesa.resolver(""); |
| 8 | } |
| 9 | **Si** (clipData.getItemCount()> = 1) { |
| 10 | ClipData.ít. primer elemento = portapapeles.getPrimaryClip().getItemA\ |

1. t(0);

|  |  |
| --- | --- |
| 12 | promesa.resolver("" + primer elemento.getText()); |
| 13 | } **más** { |
| 14 | promesa.resolver(""); |
| 15 | } |

1. } **captura**(excepción correo) {
2. promesa.rechazar(mi);

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 161 |

1. }
2. }

20

1. @SuppressLint("DeprecatedMethod")
2. @ReactMethod
3. **público vacío** conjunto(cadena de texto) {
4. clipdata ClipData = ClipData.newPlainText(**nulo**, texto);
5. portapapeles ClipboardManager = getClipboardService();
6. portapapeles.setPrimaryClip(clipdata);
7. }
8. }

Todavía aquí todo lo que se ve no es sorprendente. Estamos métodos que serán expuestos al mundo Javascript con la marca@ReactMethodanotación. Aparte de eso, lo único que puedo pensar que vale la pena mencionar es el uso dePromesa en el obtenermétodo; tener esto en cuenta, ya que tendremos que discutir por separado más abajo.

Lo siguiente que vamos a hacer Reaccionar nativa para Android sabe acerca de este módulo. Una vez hecho esto mediante la creación y registro de unaReactPackage.

* **clase pública NativeModulesPackage implementos** ReactPackage{
* @Anular
* **público** Lista<NativeModule>createNativeModules(rea ReactApplicationContext\4 ctContext) {

5**regreso** Las matrices. <NativeModule>asList(

6 **nuevo** CryptboardModule(reactContext)

* );
* }

9

1. @Anular
2. **público** Lista<Clase<? **se extiende** JavaScriptModule>>createJSModules() {
3. **regreso** Colecciones.lista vacía();
4. }

14

1. @Anular
2. **público** Lista<ViewManager>createViewManagers(ReactApplicationContext reaccionar\
3. Contexto) {
4. **regreso** Colecciones.lista vacía();
5. }
6. }

Aquí, estamos simplemente dando una lista de un artículo, nuestra CryptboardModule en el createNativeModules anular, mientras que el resto de las anulaciones de repartir vacía listas - que no tienen nada que reaccionan necesidades nativos a saber acerca de allí.

Vamos plataforma de ese paquete con nuestra actividad principal (echar un vistazo a getPackages en su actividad):

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 162 |

* **protegido** Lista<ReactPackage>getPackages() {

2**regreso** Las matrices. <ReactPackage>asList(

|  |  |  |
| --- | --- | --- |
| 3 | **nuevo** | MainReactPackage(), |
| 4 | **nuevo** | NativeModulesPackage() |

* );

6}

Esto completa el ciclo! Ahora, vamos a ser capaces de utilizar nuestro módulo nativo del IOS y Android. Lo bueno es que vamos a utilizar la misma base de código Javascript muy sin cambios para ambos.

**Cryptboard.js**

Esta parte va a ser una sorpresa para ti. Listo? Aquí vamos:

* **importar** {} Cryptboard de'' NativeModules

2

3 **clase** CryptboardModule {

* obtener(){
* **regreso** Cryptboard.get ()
* }

7

* conjunto (str) {
* *// opcional: hacer virguerías en str?*

1. Cryptboard.set (str)
2. }
3. }

13

1. module.exports = **nuevo** CryptboardModule

Eso es. Y esto funciona tanto para Android y el IOS. Si alguna vez se preguntó acerca de lo que es el X-Factor con Reaccionar nativo a diferencia de otros SDK multiplataforma; Bueno, eso es todo.

Ahora, vamos a ver cómo utilizamos este módulo de código de aplicación real:

* **importar** cryptboard de'./cryptboard'

2

3 **clase** nativemodules **se extiende** Componente {

* constructor (apoyos) {
* **súper**(accesorios)
* **esta**.estado={texto:"módulos hola"}

7**esta**.dupdo= **esta**.copy.bind (**esta**)

8**esta**.pegar= **esta**.paste.bind (**esta**)

9}

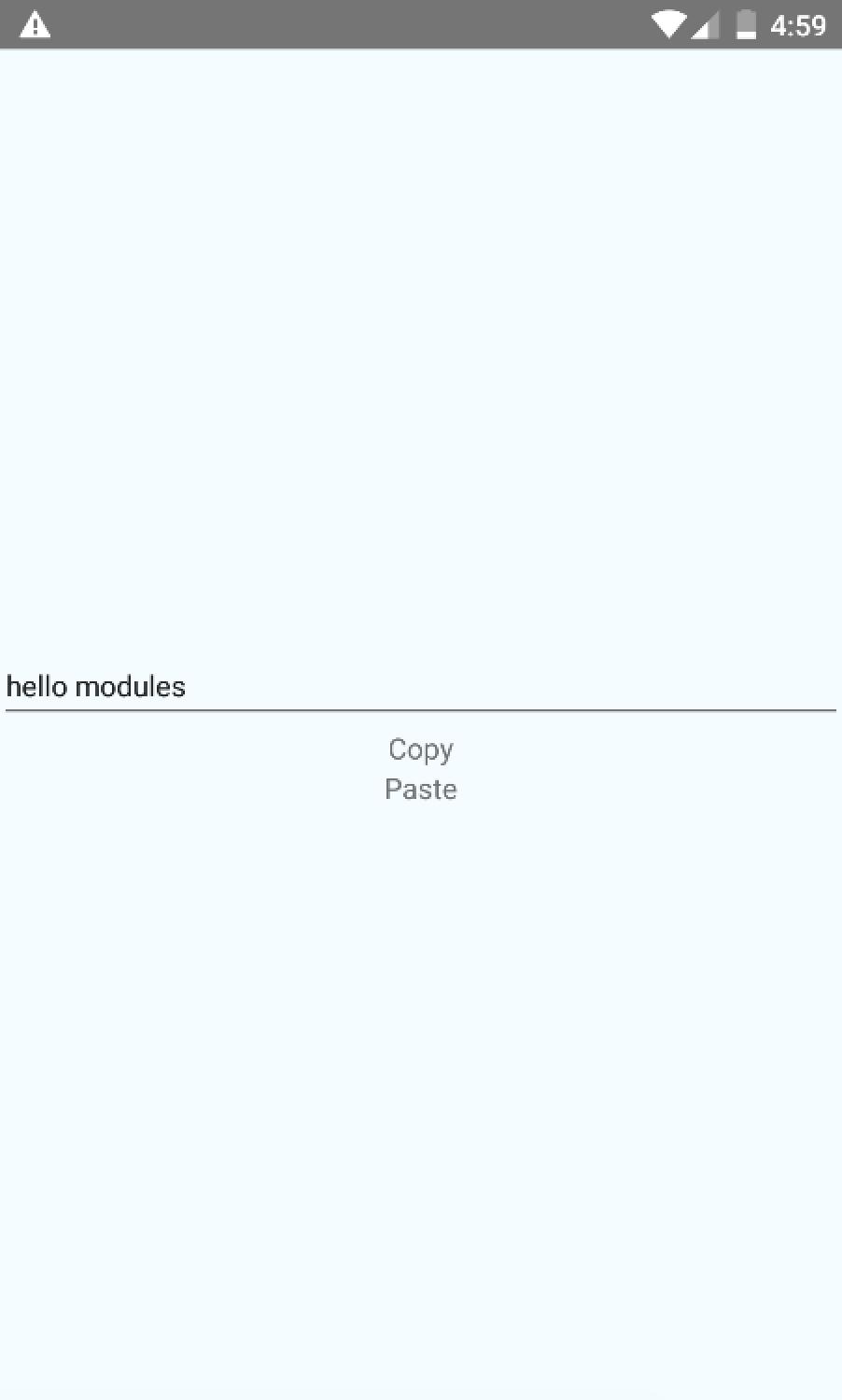
|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 163 |

1. dupdo(){
2. Cryptboard.set (**esta**.state.text)
3. }
4. pegar(){
5. Cryptboard.get (). Entonces ((args)=>{
6. *// Por desgracia, una promesa IOS es diferente de una promesa Android.*
7. *// El uno iOS devuelve una sola variable, que es una matriz de objetos*
8. *// mientras que el androide devuelve un único objeto.*
9. **var** texto=args
10. **Si**(args **en vez de** Formación) {
11. texto = args [0]
12. }
13. **esta**.setState ({texto})
14. })
15. }
16. render () {
17. **regreso** (
18. <Ver estilo={} Styles.container>
19. <Entrada de texto
20. estilo={{altura: 40, ancho del borde: 1}}
21. valor={**esta**.state.text}
22. onChangeText={(texto)=>**esta**.setState ({texto})} />
23. <TouchableOpacity onPress={**esta**.dupdo}>
24. <Texto>Dupdo</ Texto>
25. </ TouchableOpacity>
26. <TouchableOpacity onPress={**esta**.pegar}>
27. <Texto>Pegar</ Texto>
28. </ TouchableOpacity>
29. </ Ver>
30. );
31. }
32. }

He omitido las importaciones redundantes y definiciones de estilo, pero debería ser éste. Esto es otra vez la misma base de código para iOS y Android. Aquí es donde brilla Reaccionar nativo, ya que hace una cosa difícil - recuerde que estamos recibiendo nuestras manos sucias con código nativo aquí, no necesariamente Reaccionar código nativo - trivialmente fácil.

Y, por último, se trata de cómo debe comportarse tanto para iOS y Android:

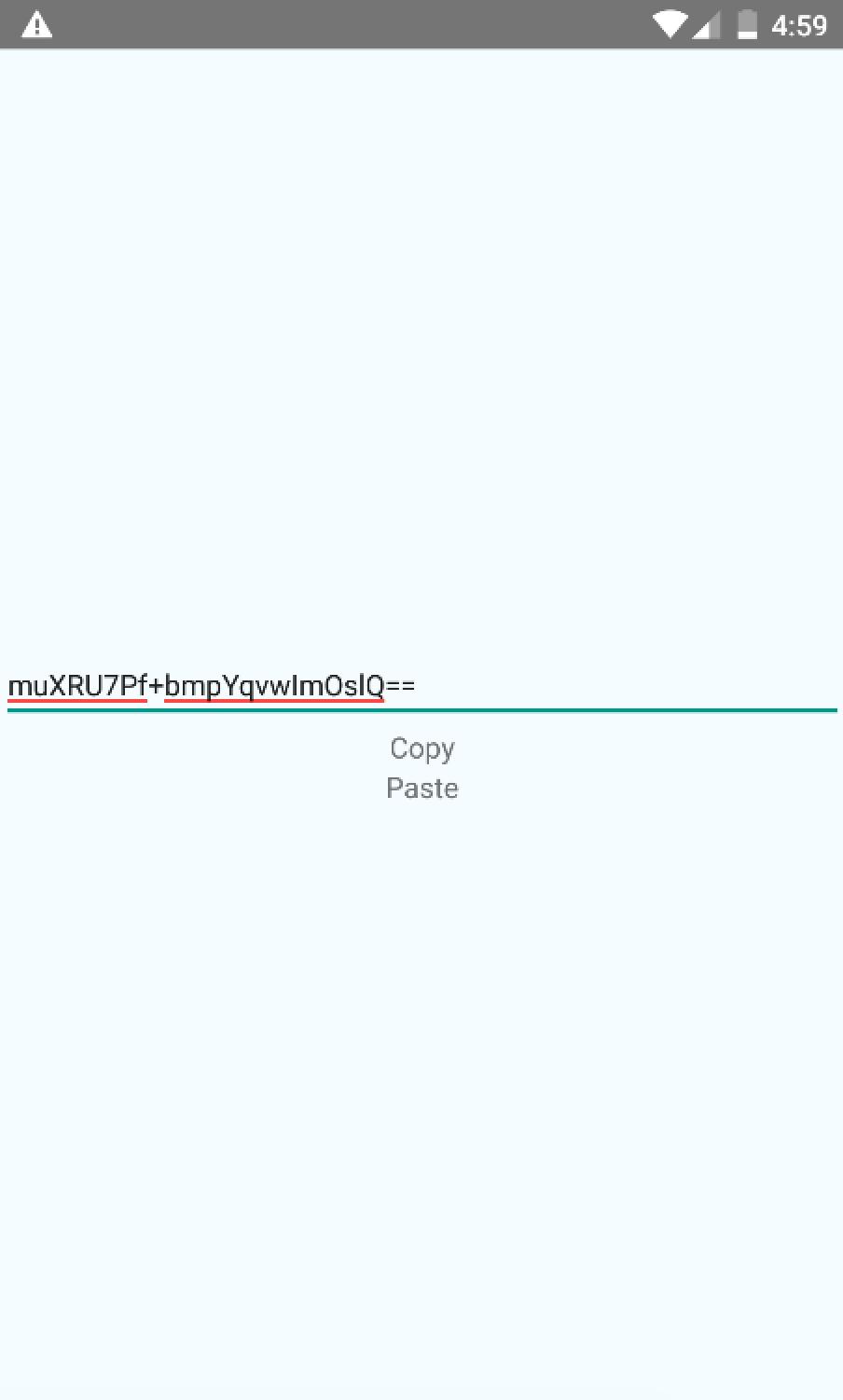
|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 164 |



**Nuestra simplista interfaz de usuario portapapeles**

Copiar y pegar trabajos, que copiar y pegar el mismo texto, en ambas plataformas. Vamos a pegar directamente desde el portapapeles sin golpearPegar:

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 165 |



**Nuestro texto cifrado (base 64)**

Así funciona el cifrado. Esto significa que nadie aparte de nosotros podemos entender todo lo que está siendo copiado o pegado de esta aplicación. Ahora todo lo que necesitamos es un sombrero de papel de aluminio!

**Las promesas de puente**

Pero espere un segundo. Hay una cosa de qué hablar, que es probablemente sólo una señal que reaccionan nativo está en 0,19, y es aún muy joven. La forma en que tiende un puente sobre el concepto de una promesa es diferente entre iOS y Android, por desgracia. En iOS Reaccionar nativo pasará una matriz con los objetos de una devolución de llamada promesa, mientras que en Android pasará un solo objeto.

Por ahora, estas fricciones se aborda de manera explícita como se puede ver en nuestro código. Esperemos que estas diferencias serán subsanadas - Lo alentador es que debería ser bastante fácil de limar así (como se puede ver en nuestro pequeño arreglo allí).

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 166 |

**Utilizando una base de código único**

En este ejemplo, estamos usando una única base de código nativo Reaccionar, con sólo dos archivos:

* index.js
* cryptboard.js

Normalmente, su Reaccionar proyectos nativos se carga una índice. [plataforma] .js y esperar un paquete de Javascript que también se infijo con el plataformaíndice. Vamos a ver cómo desactivar eso y que se cargue el mismo código base tanto en iOS y Android.

En iOS la solución es bastante simple. DentroAppDelegate.m localizar y cambio jsCodeLocation a esto (hemos eliminado el .ios infijo):

* jsCodeLocation = [NSURL URLWithString:@ "Http: // localhost: 8081 / index.bundle \ 2 = plataforma iOS y dev = true"];

En Android, tenemos que cambiar algunos lugares. En primer lugar, en suActividad principal, anulación getBundleAssetName y getJSMainModuleName:

* @Anular
* **protegido** CuerdagetBundleAssetName() {

3**regreso** "Index.bundle";

* }

5

* @Anular
* **protegido** CuerdagetJSMainModuleName() {

8**regreso** "índice";

9}

Esto le permite instruir convenientemente Reaccionar nativo qué archivo y agrupar a buscar. A continuación, ya que la parte Android incluye Gradle, también puede modificar las tareas de construcción para su conveniencia. localizar suapp / build.gradle presentar y dentro de ella uncomment y establecer esta parte de la siguiente manera:

* *// app / build.gradle*

2 proyecto.ext.reaccionar = [

3bundleAssetName: "Index.bundle",

4entryFile: "index.js"

5 ]

Ahora, ambos iOS y Android se basa va a utilizar la misma configuración de código base.

|  |  |
| --- | --- |
| Going Native: Nativo Módulos | 167 |

**Resumen**

En este capítulo, hemos breezed través de la implementación de módulos nativos. Estos son los tipos de módulos que usted necesita cuando se desea exponer parte de las capacidades específicas nativas de su Reaccionar código nativo Javascript.

Este es un resumen de lo que hicimos:

* Hemos implementado un divertido ejemplo - un cryptboard. Un portapapeles que guarda sus datos cifrados con el algoritmo simétrico de cifrado AES
* Y por el camino, hemos visto cómo tomar en dependencias para iOS con Cocoapods, y para Android con Gradle
* Hemos aprendido que es mucho más simple que la construcción de un componente en toda regla interfaz de usuario nativa
* Todo se reduce a registrar el módulo, la exposición de los métodos y la opción-aliado utilizando promesas
* Al utilizar módulos nativos, hay una mayor posibilidad de que el modo de usar estos de módulos tanto para iOS y Android Javascript será exactamente el mismo. Hemos visto cómo forzar Reaccionar nativo para utilizar la misma base de código en cada plataforma desde el primer momento.
* implementar el servicio portapapeles en ambos planos catastrales.
* ir sobre ellos.
* la diferencia entre los bloques promesa - iOS y Android
* cómo hacer que iOS y Android utilizar un index.js individuales (IOS - appmodule, androide - actividad de primer orden y de fijación de las tareas de construcción Gradle)
* capturas de pantalla - copiar, pegar, pegar encriptados