

**Reaccionar de programación nativo**

Dotan Nahum

Este libro está a la venta en <http://leanpub.com/programming-react-native>

Esta versión se publicó el 01/07/2016



Esto es un [Leanpub](http://leanpub.com) libro. Leanpub permite a los autores y editores con el proceso de publicación de Lean.[editorial magra](http://leanpub.com/manifesto) es el acto de publicar un libro electrónico en curso utilizando herramientas ligeras y muchas iteraciones para obtener retroalimentación de los lectores, de pivote hasta que tenga el libro adecuado y construir la tracción vez que lo hace.

© 2015 - 2016 Dotan Nahum

**Contenido**

|  |  |
| --- | --- |
| [**Sobre mi**](#page6) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **1** |
| [**Sobre este libro**](#page8) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **3** |
| [Cómo construir un libro de Bleeding Edge?](#page8) . . . . . . . . . . . . . . . . | 3 |
| [Cómo leer este libro?](#page9) . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| [**Introducción**](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **5** |
| [Sobre mi](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [La revolución ha comenzado](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [Cruz plataforma móvil](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [Reaccionar](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [reaccionar nativo](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [¿Por qué esta vez?](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [El estilo del libro](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [Público objetivo](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [Lo que este libro es](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [Lo que este libro no es](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [A través de breezing Reaccionar](#page10) . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [**Empezando**](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **6** |
| [diseño de proyecto](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| [El proyecto React nativo](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| [El funcionamiento de su proyecto](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| [Haciendo cambios](#page14) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 9 |
| [Su agrupación de aplicaciones](#page18) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 13 |
| [El Proyecto nativo de iOS](#page19) . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
| [El Proyecto nativo de Android](#page22) . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
| [Estampación](#page24) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 19 |
| [Simulador y emulador de Herramientas de Desarrollo](#page24) . . . . . . . . . . . . . . . | 19 |
| [Depuración nativa](#page28) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 23 |
| [Resumen](#page30) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 25 |
| [**Edificio Reaccionar componentes nativos**](#page31) . . . . . . . . . . . . . . . . . . . | **26** |
| [Una historia de Squashed Javascript Frameworks](#page31) . . . . . . . . . . . . . | 26 |
| [React.js](#page32) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 27 |
| [reaccionar componentes](#page32) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 27 |
| [Completar un Kata](#page33) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 28 |

|  |  |  |
| --- | --- | --- |
| CONTENIDO |  |  |
| [**La aplicación Contactos**](#page41) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | **36** |
| [Caracteristicas](#page41) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 36 |
| [Producto](#page41) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 36 |
| [Técnico](#page57) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 52 |
| [Potencial (o: Preparación)](#page59) . . . . . . . . . . . . . . . . . . . . . . . . | . . | 54 |
| [Resumen](#page59) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 54 |
| [**Tutorial**](#page60) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | **55** |
| [Estilo walkthough](#page60) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 55 |
| [Las opciones técnicas](#page60) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 55 |
| [javascript](#page61) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 56 |
| [Reaccionar](#page61) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 56 |
| [Flujo](#page61) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 56 |
| [Estructura de carpetas](#page62) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 57 |
| [Pruebas](#page63) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 58 |
| [La disección de nuestra primera pantalla](#page63) . . . . . . . . . . . . . . . . . . . . . . | . . | 58 |
| [Lista móviles Vistas](#page66) . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 61 |
| [La pantalla de Grupos](#page68) . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 63 |
| [Maestro-Detalle](#page70) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | sesenta y cinco |
| [ListView y Nuestra principal Vista](#page72) . . . . . . . . . . . . . . . . . . . . | . . | 67 |
| [La tienda](#page76) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 71 |
| [Secuencia de arranque y de navegación](#page80) . . . . . . . . . . . . . . . . . . . . . | . . | 75 |
| [Styling](#page86) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 81 |
| [La imagen detallada](#page88) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . | 83 |
| La gente (Contactos) de la pantalla. . . . . . . . . . . . . . . . . . . . . | . . | 93 |
| La fabricación de componentes reutilizables. . . . . . . . . . . . . . . . . . . . | . . | 98 |
| Utilización de componentes de la Comunidad. . . . . . . . . . . . . . . . . . . . | . . 100 | |
| Vincular los proyectos de iOS. . . . . . . . . . . . . . . . . . . . . . . . . | . . 100 | |
| La vinculación de Proyectos Android. . . . . . . . . . . . . . . . . . . . . . | . . 105 | |
| Componentes Javascript. . . . . . . . . . . . . . . . . . . . . . . . | . . 108 | |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 109 | |
| **Navegación y enrutamiento** . . . . . . . . . . . . . . . . . . . . . . . . . | . 0,110 | |
| ¿Por qué la navegación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 110 | |
| ¿Por qué Navegación da miedo. . . . . . . . . . . . . . . . . . . . . . . . | . . 110 | |
| La navegación en React nativo. . . . . . . . . . . . . . . . . . . . . . . | . . 111 | |
| Navigator vs NavigatorIOS. . . . . . . . . . . . . . . . . . . . . . | . . 112 | |
| Navigator. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 112 | |
| Cableado Navigator. . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 112 | |
| ToolbarAndroid y Navigator.NavigationBar. . . . . . . . . . | . . 114 | |
| ToolbarAndroid. . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 115 | |
| El botón Atrás del androide. . . . . . . . . . . . . . . . . . . . . . . . | . . 118 | |
| NavigatorIOS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 119 | |
| Pasando de datos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 121 | |
| Patrones . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 122 | |
| Buscar en barra de navegación. . . . . . . . . . . . . . . . . . . . . . . . . . . | . . 122 | |

|  |  |
| --- | --- |
| CONTENIDO |  |
| De contenido personalizado en el título | . . . . . . . . . . . . . . . . . . . . . . . . . 123 |
| Enrutada contenido barra de navegación | . . . . . . . . . . . . . . . . . . . . . . . . . 123 |
| Reactiva contenido barra de navegación | . . . . . . . . . . . . . . . . . . . . . . . . 124 |

Obtención de entrada. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 124 Apoyos extendiendo. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125

|  |  |
| --- | --- |
| Resumen . . . . . . . | . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125 |
| **Going Native: interfaz de usuario nativa** | . . . . . . . . . . . . . . . . . . . . . . . . . . 0,126 |

¿Por qué ir nativo? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126 Rendimiento. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . utilización 126 Fabricación de trabajo existentes. . . . . . . . . . . . . . . . . . . . . . 127

|  |  |
| --- | --- |
| Herramientas mejor. . . . . . . . . . . | . . . . . . . . . . . . . . . . . . . . 127 |
| Interfaz de usuario personalizada y la interfaz de usuario Complejo Trabajo | . . . . . . . . . . . . . . . . . . . 127 |
| Envolviendo Componentes existentes | . . . . . . . . . . . . . . . . . . . . 127 |

|  |  |
| --- | --- |
| Un general escotilla de escape. . . . . . . . . . . . . . . . . . . . . . . . . 127 | |
| Cuando no ir nativo? | . . . . . . . . . . . . . . . . . . . . . . . . . . 128 |
| Los bloques de construcción de una vista personalizada. . . . . . . . . . . . . . . . . . 128 | |
| El ViewManager. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 129 | |
| Nuestro Ejemplo: MessagesView. . . . . . . . . . . . . . . . . . . . . . . 130 | |
| Desglose: IOS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132 | |
| MessagesView. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132 | |
| RCTMessagesViewManager. . . . . . . . . . . . . . . . . . . . . . . 134 | |
| messagesview.ios.js. . . . . . . . . . . . . . . . . . . . . . . . . . . . 138 | |
| Desglose: Android | . . . . . . . . . . . . . . . . . . . . . . . . . . . 143 |
| MessagesView. . . . | . . . . . . . . . . . . . . . . . . . . . . . . . . . 143 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MessagesViewManager | | | . . . . . . . . . . . . . . . . . . . . . . . . . 148 | |
| NativeControlsPackage | | | . . . . . . . . . . . . . . . . . . . . . . . . . 150 | |
| messagesview.android.js. . . . . . . . . . . . . . . . . . . . . . . . . 152 | | | | |
| usando MessagesView | | . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155 | | |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 156 | | | | |
| **Going Native: Nativo Módulos** | | | | . . . . . . . . . . . . . . . . . . . . . . 0,157 |
| Nuestro Ejemplo: cryptboard | | | . . . . . . . . . . . . . . . . . . . . . . . . . 157 | |
| Desglose iOS | . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158 | | | |
| Desglose Android. . . . . . . . . . . . . . . . . . . . . . . . . . . . 160 | | | | |
| Cryptboard.js | . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 162 | | | |
| Las promesas de puente | | . . . . . . . . . . . . . . . . . . . . . . . . . . . . 165 | | |
| Utilizando una base de código único | | | . . . . . . . . . . . . . . . . . . . . . . . . 166 | |
| Resumen . . . . . . . . . . | | | . . . . . . . . . . . . . . . . . . . . . . . . . 167 | |

**Sobre mi**

Mi nombre es Dotan Nahum, y mi historia con la informática comenzó en algún lugar alrededor del 95' .

Mientras yo tenía un Apple IIe desde la primera infancia, realmente comenzó a los 14 años, en la entonces muy vibrante [demoscene](https://en.wikipedia.org/wiki/Demoscene)[1](#page6). Llegué a unirse a algunos grupos y encontré un grupo, y la mayor parte de mi codificación estaba en ensamblador y C a través de conexión a Internet 14k de basura, y un UENTA ISP que me las arreglé para compartir con alguien. Me pasaba los días mods del juego de construcción (cuando no era tan popular como hoy) y las noches de la depuración de código ensamblador dolorosamente opaco de otras personas que utilizan herramientas de núcleo duro, como SoftICE; Creo que esto fue lo que forjó mi personalidad como un hacker.

Durante el boom de las puntocom que hice el trabajo en C, Perl, Python, VB, Java, para terminar profundamente en .NET en 2005, la construcción de una nueva empresa con sede en .NET 2007 y, finalmente, unirse a una empresa para ser el plomo tecnología en torno a 2008. Si marca, en una retrospectiva, esa línea de tiempo de las tecnologías coincide punto de inflexión de cada tecnología (suena increíble, pero una vez, Java era muy inservible).

A partir de ahí, me mudé al auge web Facebook-era, y se unió a una compañía de adtech para servir como la infraestructura de un solo hombre-espectáculo. Con los años, me uní a una división que construyó un navegador, formé una división que construyó una rica bloqueo de pantalla Android basado en la recomendación de contenidos, que compitió con Facebook Inicio (empezamos camino delante de Página principal), y 3 años después de eso, me moví para ser el director de tecnología de una empresa la construcción de una plataforma para aplicaciones móviles de bricolaje: una especie de Wix para aplicaciones nativas.

Durante todo ese tiempo, no dejaba de hacer [fuente abierta](https://github.com/jondot)[2](#page6). Empecé en torno al 99' , cuando la mayoría de la gente no sabía lo que era eso.

La razón por la que le estoy diciendo todo esto es que pasé todos mis años buenos y tiempo libre, para mejor o peor, llegando a estar familiarizado con el alto nivel y de bajo nivel de un enorme conjunto de tecnologías de si se trata de backend, escritorio o móvil.

Hoy en día, que me encuentre enterrado en otra pila de tecnologías: Reaccionar y reaccionar nativo. No es la primera multiplataforma bala de plata pila de desarrollo móvil He intentado tampoco. La plataforma móvil He hablado acerca de ser un director de tecnología de? Que se construye en Córdoba, y más tarde nativa iOS y Android nativo, y se generaron alrededor de 2 millones de aplicaciones móviles hasta la fecha. He intentado Xamarin, Appcelerator, y Ruby movimiento. También he intentado CrossWalk y Córdoba. También he utilizado Cocos-2dx para los juegos. Y la Unidad.

1

<https://en.wikipedia.org/wiki/Demoscene>

* <https://github.com/jondot>

1

|  |  |
| --- | --- |
| Sobre mi | 2 |

¿Quieres saber cuál era el tema general de todos éstos? Simplemente, uno de estos tres:

1. Mal desempeño
2. Mala API, documentos, y / o de apoyo
3. Mismatch y la fricción contra SDK móvil aguas arriba (Android y el IOS)
4. muestras rotas o rancios, APIs

De buenas a primeras, Reaccionar nativo en su versión 0.19, y menos de un año de ser multiplataforma (es decir, cuando Reaccionar nativa para Android se produjo), ya hace mejor en todas estas cuentas.

**Sobre este libro**

Escribir un libro acerca de una tecnología de punta sangrado es difícil. Alguien dijo: “Si se construye con la punta de lanza, es de esperar que haya derramado un poco de sangre”. Aunque yo no iría tan lejos, quiero decir lo siguiente:

* Reaccionar nativos actualizados 6 versiones al escribir este libro (varios meses)
* La primera versión de React nativo tenía una estructura de proyecto completamente diferente, y las muestras de libros se veía completamente diferente
* API se rompieron cada una y otra vez, y las muestras consiguieron volver a escribir hasta que sintieron un poco sucio

Al principio, este libro se veía completamente diferente. Solía ​​tener este aspecto:

* Introducción
* reaccionar principios
* Reaccionar nativo para empezar
* Una acumulación paso a paso para salir de la aplicación Escaparate
* Temas avanzados

Pronto me di cuenta que esta estructura es muy poco fiable, ya que los cambios en React nativo y el ecosistema Javascript condenadas a cambiar con mucha frecuencia.

**Cómo construir un libro de Bleeding Edge?**

Bueno, esto es lo que he hecho, con el tiempo.

1. Acabar con el paso a paso y asumir los lectores están llegando con experiencia y tienen una mentalidad de piratería informática. Desde mi experiencia de escribir este libro, puedo decir definitivamente que el principiante “paso a paso” y “aprendizaje” libros de tipos que están ahí fuera en este momento, por ahora, son probablemente vale nada (a menos que se actualizan cada dos semanas).
2. Tire a la basura todos los componentes Javascript y el uso de la biblioteca que no sirven no di- rectamente con el fin de reaccionar nativo por la puerta. Esto significaba:
3. El ensayo no parece en este libro (en el momento de escribir la forma de probar todavía se está inventado. Actualmente las personas utilizan Mocha, una costumbre Reaccionar componentes simulados nativos, y la enzima)

3

|  |  |
| --- | --- |
| Sobre este libro | 4 |

1. No se incluye un marco de flujo, sin embargo, el código base se asemeja claramente un flujo de sabor menos
2. Centrarse en: ¿por qué, el contexto, la forma y los patrones. Concéntrese en las cosas que se mantendrá, mientras que el polvo se asiente borde de la sangría, incluso dentro de un año.
3. Invertir más tiempo en temas avanzados, de bajo nivel, ya que estos van a cambiar con menos frecuencia que la superficie del área de la API de cara al usuario.

**Cómo leer este libro?**

Con eso fuera del camino, se puede leer este libro de principio a cubrir, o puede echarle un vistazo, o simplemente puede leer los capítulos avanzados y se basan en los diversos tutoriales en marcha hasta la fecha por ahí que van sobre los fundamentos de la construcción Reaccionar un nativo. Mantengo la[Impresionante-React-Nativo](https://github.com/jondot/awesome-react-native)[3](#page9) lista que sostienen muchos de estos, por lo que sólo puede ir tan y recoger un tutorial o una aplicación de ejemplo que te gusta.

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

**Introducción**

**Sobre mi**

**La revolución ha comenzado**

**Cruz plataforma móvil**

**Reaccionar**

**reaccionar nativo**

**¿Por qué esta vez?**

**El estilo del libro**

**Público objetivo**

**Lo que este libro es**

**Lo que este libro no es**

**A través de breezing Reaccionar**

5

**Empezando**

**diseño de proyecto**

Reaccionar nativo es un SDK multiplataforma para el desarrollo móvil. Como tal, usted necesita saber cómo un proyecto que el Reaccionar nativo CLI genera ve como en términos de archivos clave, carpetas y así sucesivamente; pero usted, sin duda, también es necesario conocer a los lados nativos específicos para cada plataforma; iOS y Android, y usted puede estar seguro de que cada uno de ellos son completamente diferentes en términos de herramientas de construcción, gestión de la dependencia, la gestión de recursos y convenciones de los proyectos.

Vamos a tratar de cubrir todas estas aquí de una manera suficiente que le llevará a través de su futuro Reaccionar proyectos nativos. Mi consejo para usted, como un programador Reaccionar es nativo de aceptación. Por ejemplo, si usted es originalmente un desarrollador de iOS, ser tolerante hacia el entorno de desarrollo de Android, herramientas y estructura del proyecto, y al revés. No es fácil, porque algunas cosas se construyen mejor que otros y es posible que se han acostumbrado a utilizar las cosas mejores de toda su carrera.

**El proyecto React nativo**

En primer lugar, tome un momento para configurar el entorno de acuerdo con el [último](https://facebook.github.io/react-native/docs/getting-started.html) [mejores prácticas](https://facebook.github.io/react-native/docs/getting-started.html)[4](#page11) en el funcionario Reaccionar sitio de documentación nativo. Esa voluntad caminar a través de la creación de Node.js, el Android SDK, vigilante, el flujo y herramientas adicionales que hacen que el Reaccionar pila nativa grande para el desarrollo. A continuación, para iniciar la discusión, vamos a generar un proyecto nativo que tendremos finalmente tiramos React:

* projectlayout init $ reaccionar nativo

Esperar un poco, y nos pondremos lo siguiente superestructura:

* <https://facebook.github.io/react-native/docs/getting-started.html>

6

|  |  |
| --- | --- |
| Empezando | 7 |

* .
* ├── androide /

3 ├── ios /

4 ├── index.android.js

5 ├── index.ios.js

6 └── package.json

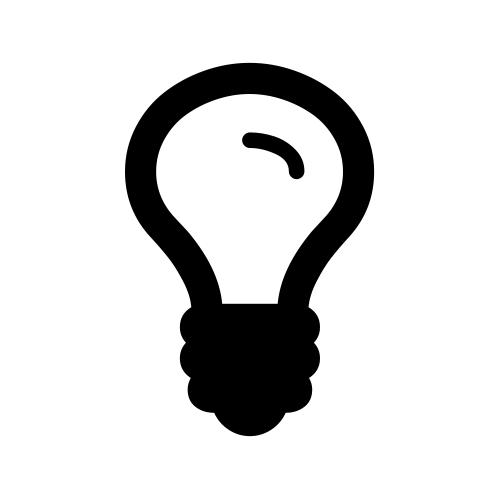
En el primer nivel del árbol del proyecto no hay complejidad en absoluto, y creo que esto es muy bueno. Reaccionar nativo no obliga a ninguna estructura convención o proyecto y sólo expone lo esencial, tales como nuestropackage.json que se utiliza para la dependencia y Gestión de Proyectos a través NPMY los dos puntos de partida para iOS y Android - index.ios.js y index.android.js. Además de éstos cuidadosamente unaandroide/ y ios / carpetas se colocan y vamos a tocar estas muy breve.

Para leer sobre gestión de la dependencia con Javascript NPM mirar [esta](https://docs.npmjs.com/files/package.json)[5](#page12) y una o más amigable [empezando](https://docs.npmjs.com/getting-started/using-a-package.json)[6](#page12) guía.

Ambos index.ios.js y index.android.js están cableados en código de arranque de cada proyecto nativa, y cada plataforma sabe cómo filtrar su propio código específico de la plataforma por la .ios o .androideinfijos. En otras palabras, un archivo etiquetado confile.ios.js nunca será visible para Android al importar archivo y al revés para el .androide infijo.

Ahora sería un buen momento para leer más sobre [Plataforma código específico](https://facebook.github.io/react-native/docs/platform-specific-code.html#content)[7](#page12) en el sitio oficial de documentación.

Siempre que sea posible, te consulte la documentación oficial. No hay punto de re-articular algo que ya se encuentra perfectamente y libremente escrito en otra parte. Al ser un libro de Indy me permite hacer el contenido aquí para ser justo sobre el tema y evitar recurrir a la página de inflado tácticas utilizan las grandes editoriales.



**El funcionamiento de su proyecto**

Por ahora, puede ejecutar ambos proyectos Android y iOS desde la línea de comandos:

* $ Reaccionar nativo run-ios

Para Android, asegúrese de que su ANDROID\_HOME variable de entorno ya está establecido para la sesión o el usuario y ejecutar de este modo:

5<https://docs.npmjs.com/files/package.json>

6<https://docs.npmjs.com/getting-started/using-a-package.json>

* <https://facebook.github.io/react-native/docs/platform-specific-code.html#content>

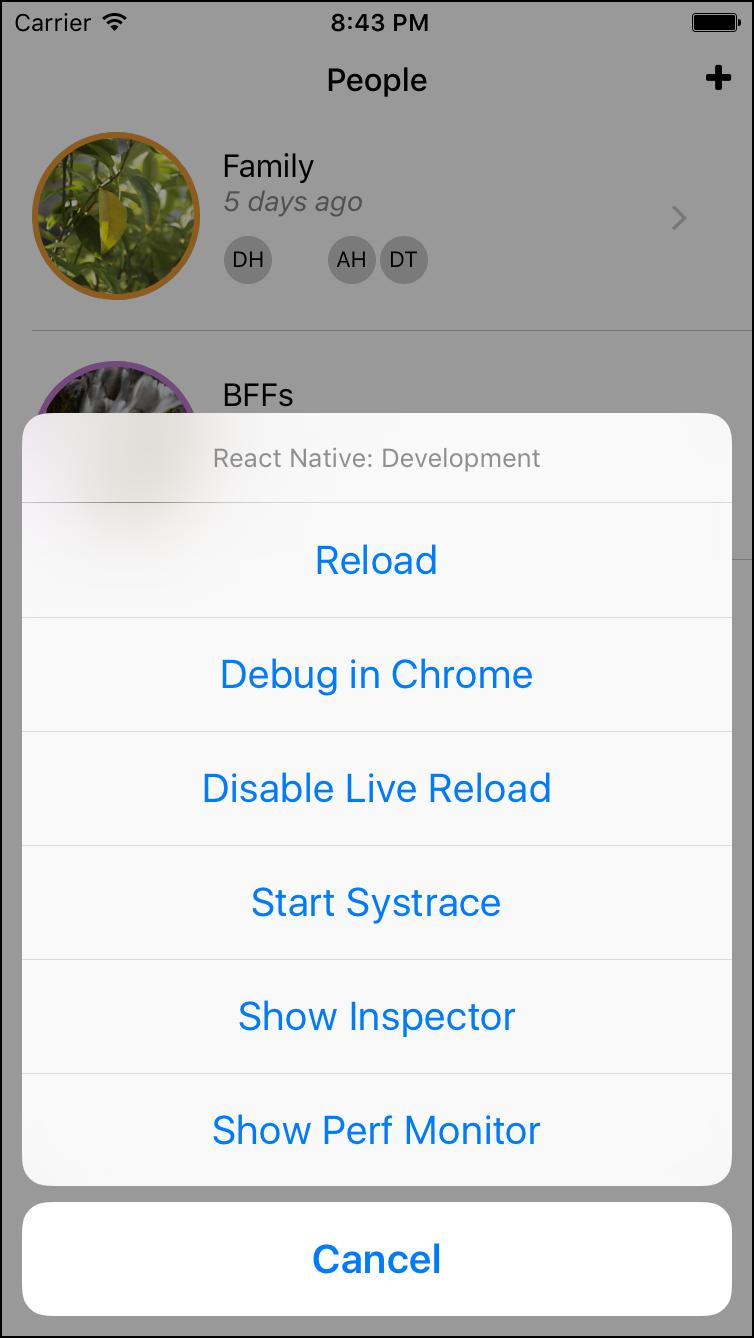
|  |  |
| --- | --- |
| Empezando | 8 |

* ps exportar ANDROID\_HOME=<Su-android-casa-path>

2 $ Reaccionar nativo carrera androide

Para obtener una bodega de la ruta de inicio de Android, puede que tenga que recordar donde se ha instalado la distribución de Android SDK a si lo hizo de forma manual, o si está usando Android Studio echar un vistazo a Archivo-> Proyecto de Estructura-> SDK de Android.

El siguiente, usted debe ser capaz de cambiar el código Javascript y vuelve a cargar con facilidad ctrl-CMD-Z (Gesto de agitar) el simulador de iOS que se abre la acción de depuración hoja, o el botón de menú en Android que hace lo mismo. Por ahora, también se sienten libres para explorar el resto de este menú, nos pondremos en contacto con él más adelante.



**El menú de Desarrollo**

Si ha notado, que no compilar o ejecutar cualquier código nativo, y si has jugado con la opción “Actualizar vivo” (debe), entonces se llega a ver su aplicación reflejan automáticamente los cambios en el código de una vez le das “ guardar”en el editor. Esta magia sucede porque en el modo de desarrollo Reaccionar nativo

|  |  |
| --- | --- |
| Empezando | 9 |

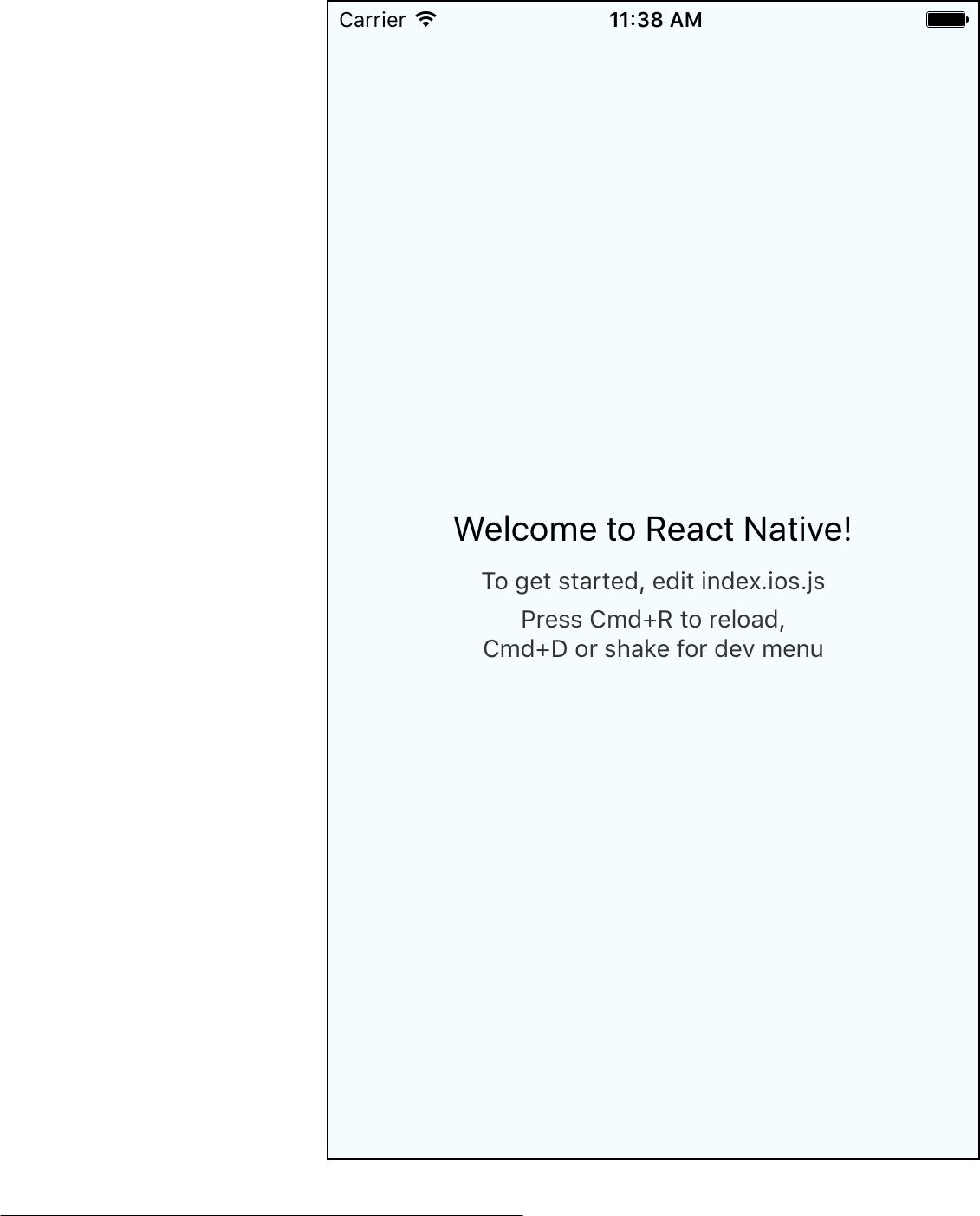
se comunica con un pequeño servidor de empaquetado y está trabajando en un “[RPC](https://en.wikipedia.org/wiki/Remote_procedure_call)[8](#page14)" modo. Este servidor que usted puede haber notado a aparecer sin invitación cada vez que se ejecute el proyecto (no matarlo!) Relojes de los archivos de origen, y comunica los cambios de nuevo a la aplicación nativa que se ejecuta en el dispositivo o simulador.

**Haciendo cambios**

Ahora vamos a tocar el violín un poco con el proyecto predeterminado generado. Ejecutar el simulador de iOS, así:

* $ Reaccionar nativo run-ios

Esto tomará un poco, y verá el IOS simulador de incendios y un terminal con el empaquetador Reaccionar nativo pop-up también. Debería ver lo siguiente:



* <https://en.wikipedia.org/wiki/Remote_procedure_call>

|  |  |
| --- | --- |
| Empezando | 10 |

ahora edite index.ios.js, Localizar el componente principal llamado projectlayout y editarla para que usted termina con esto, arranque sustancial de un grupo de componentes:

* **clase** projectlayout **se extiende** Componente {

2render () {

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

5<Estilo de texto={} Styles.welcome>

* Bienvenido a reaccionar nativo!
* </ Texto>
* </ Ver>
* );

10}

11 }

Golpear Cmd-R cuando estás frente a la simulador de iOS para recargar (o, aún mejor golpe Ctrl-Cmd-Zy seleccione “Activar Actualizar vivo”). Cuando vuelve a cargar el contenido, vamos a terminar con esto:

|  |  |
| --- | --- |
| Empezando | 11 |



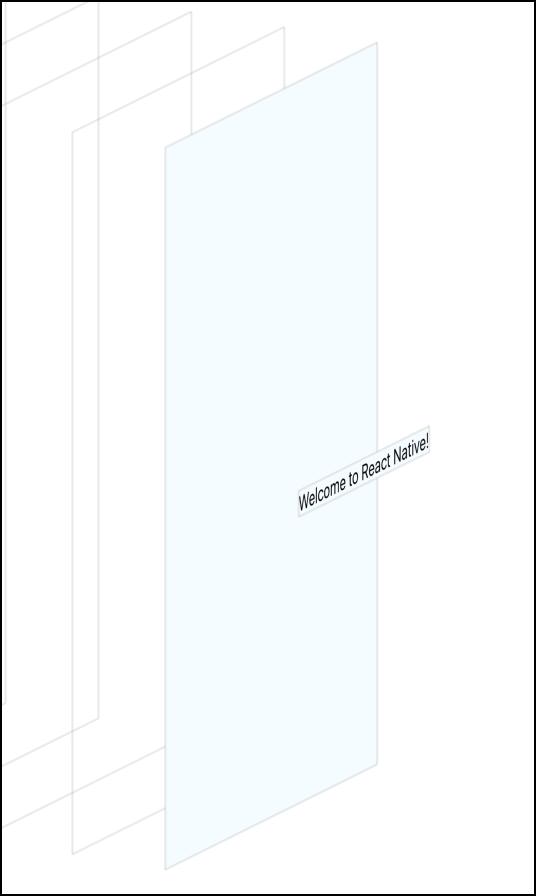
Vamos a ver cómo esto se ve como desde el punto de vista de la plataforma iOS. ¿Es esto realmente nativa?

Vaya a su terminal y en el directorio de trabajo del proyecto haga lo siguiente:

* ios abiertas / projectlayout.xcodeproj

Xcode se abrirá, ahora golpeado Cmd-Rpara ejecutar el mismo proyecto a través de Xcode. Después de la aplicación se ejecuta en el simulador de iOS, volver a Xcode y por el menú principal golpeDepurar -> Ver Depuración -> Captura Ver Jerarquía. Después de un par de segundos que va a terminar con un modelo 3D de la vista de jerarquía, agarrar y la incline, y esto es lo que obtenemos:

|  |  |
| --- | --- |
| Empezando | 12 |



Se ve increíblemente limpio. Desde mi experiencia hay un montón de posibilidades de una plataforma cruzada SDK nativo a la basura realmente la jerarquía de vistas:

* Ver abstracciones hechas para hacer la vida más fácil para los desarrolladores de la plataforma - en este caso se obtiene un árbol con los componentes de la envoltura y el soporte sin sentido, que los desarrolladores de plataformas utilizan para rastrear el árbol en sí. No existe Reaccionar con nativos, y se obtiene una prima, plana, árbol que es muy importante para el rendimiento.
* abstracciones lógicas hechas para hacer la vida más fácil para la plataforma docu-menters - en este caso los fabricantes de plataformas deciden construir sus propios componentes personalizados y que utilice estos exclusivamente porque les da más control sobre cómo la plataforma lleva a cabo en general y la documentación también más fácil historia. Reaccionar nativo no trata de resolver eso y le da acceso a un conjunto limitado de controles y componentes nativos desnudos de cada plataforma.
* Árboles errores de jerarquía - simplemente porque dicha plataforma puede generar código nativo de su código Javascript abstracto, puede hacerlo sin mucha inteligencia y generar enormes árboles vista ofuscado. De nuevo -

|  |  |
| --- | --- |
| Empezando | 13 |

Reaccionar nativo no hace eso, más sobre ella utiliza reaccionar para ayudar a calcular y hacer que los árboles y las diferenciaciones.

Encontrará que voy a hablar mucho sobre el factor X que mantiene Reaccionar nativo, aparte de todos los SDK móvil multiplataforma que hemos visto hasta ahora - eso es porque yo he utilizado un buen número de ellos y las cicatrices aún muestran. Reaccionar nativo hace una diferencia real y que acabamos de ver otro de estos factores.

**Su agrupación de aplicaciones**

No es relevante para el futuro inmediato, pero muy relevante para cuando usted quiere enviar su solicitud, usted realmente no quiere depender de un servidor local cuando se ejecuta en un dispositivo real en la red principal de un usuario. Debido a eso, Reaccionar nativo proporciona una utilidad de línea de comandos para empaquetar su aplicación para la producción (o, fuera de línea) de trabajo:

* psreaccionarhaz -native

2 opciones:

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | --entry-archivo | Ruta al archivo JS raíz, ya sea absoluta o relativa t \ | |
| 4 | O raíz JS | [necesario] |  |
| 5 | --plataforma | Ya sea "ios" o "androide" |  |
| 6 | --transformador | Especificar un transformador de medida para | utilizar (ruta absoluta) \ |
| 7 |  | [defecto: | "/ Users / Dotan / proyectos / \ |

* programación reaccionar nativo-muestras / projectlayout / node\_modules / reaccionar nativo / pack \

9 Ager / transformer.js "]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | --dev | Si | falso, advertencias están desactivados | y el haz se minif \ |
| 11 | IED |  | [defecto: | cierto] |
| 12 | --prepack | Si | verdadero, el haz de salida se | utilizar el formato de preempaque. \ |
| 13 |  |  | [defecto: | falso] |
| 14 | --bridge-config | Nombre del archivo de exportación aa JSON de | | \_\_fbBatchedBridgeConfig \ |

1. . Utilizado por preempaque. Ex. ./bridgeconfig.json

|  |  |  |  |
| --- | --- | --- | --- |
| dieciséis | --bundle-output | Nombre del archivo donde almacenar el paquete resultante, ex. / Tmp \ | |
| 17 | /groups.bundle |  | [necesario] |
| 18 | --bundle-codificación | Codifica el paquete debe ser escrito en (https: // nodej \ | |
| 19 | s.org/api/buffer.html*#*buffer\_buffer). | | [Predeterminado: "UTF-8"] |

1. --sourcemap-producto Nombre del archivo donde almacenar el archivo Sourcemap para resultadoi \
2. ng paquete, ex. /tmp/groups.map

22 --assets-dest nombre del directorio donde almacenar los activos que se hace referencia en el \

1. haz

|  |  |  |  |
| --- | --- | --- | --- |
| 24 | --verboso | Habilita el registro | \ |
| 25 |  |  | [Por defecto: false] |

Que va a utilizar esa herramienta para crear su main.jsbundle que se adjuntan a la binarios de la aplicación (IPA y en iOS apk en Android), por lo que para iOS, por ejemplo, - proporcionar los siguientes argumentos: ios --platform, --entry-archivo index.ios.jsy ios --bundle-salida / main.jsbundle. Al igual que:

|  |  |
| --- | --- |
| Empezando | 14 |

* $ Reaccionar nativo index.ios.js paquete --entry en archivos --platform ios --bundle-Outpu**\**

2 t ios / main.jsbundle

3 bundle: Creado ReactPackager

4 paquete: inicio

5 paquete: Acabado

6 paquete: Escritura de salida paquete a: IOS / main.jsbundle

7 paquete: Cierre cliente

8 carpeta de destino de los activos no se establece, saltando ...

9 bundle: Hecho escritura de salida haz

El mismo proceso se lleva a cabo para Android (por supuesto, sustituir ios con androide en su caso y asegurarse de que la salida del haz entra en el proyecto Android).

Ni que decir tiene, que va desde la línea de comandos es muy conveniente y crea un flujo “Zen”, mientras que el desarrollo, ya que no sólo no te molesta con la lentitud de manejar una interfaz gráfica de usuario (apuntar y hacer clic), pero también se puede automatizar las tareas de línea de comandos . Sin embargo, llega un momento en que usted necesitará un poco más de potencia de fuego, para depurar el código nativo, añadir su propio código nativo, o configurar la aplicación de concha, los recursos o permisos. Por todo esto, necesitará por lo menos a conocer cada uno de sus proyectos individuales o iOS Android y herramientas.

**El Proyecto nativo de iOS**

A continuación se muestra un volcado del árbol que nuestra init reaccionar nativo comando genera para la parte de IOS de nuestro proyecto.

* ├── ios

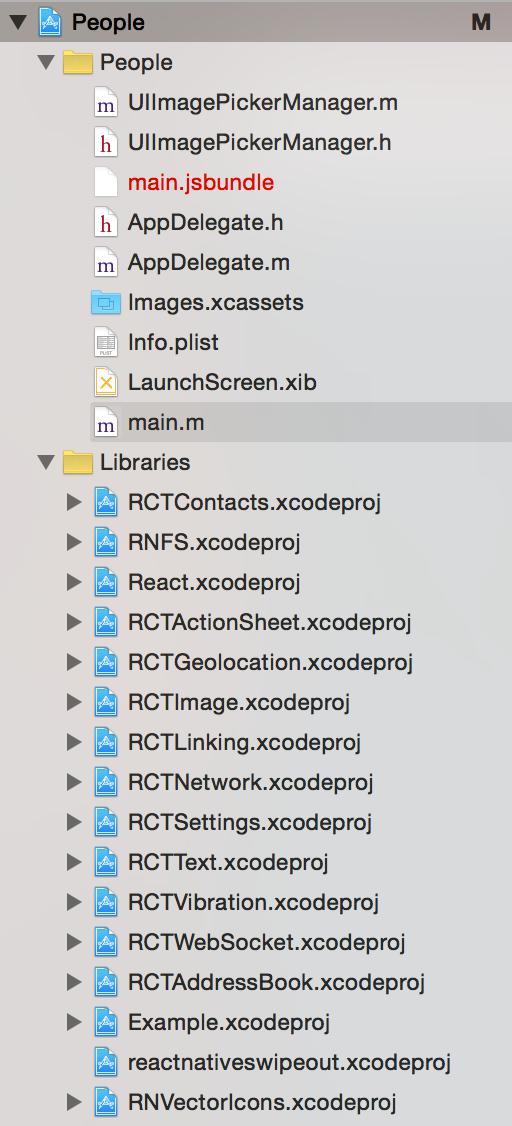
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | │ | ├── | projectlayout | |
| 3 | │ | │ | ├── | AppDelegate.h |
| 4 | │ | │ | ├── | AppDelegate.m |
| 5 | │ | │ | ├── | Base.lproj |
| 6 | │ | │ | │ | └── LaunchScreen.xib |
| 7 | │ | │ | ├── | Images.xcassets |
| 8 | │ | │ | │ | └── AppIcon.appiconset |
| 9 | │ | │ | │ | └── Contents.json |
| 10 | │ | │ | ├── | info.plist |
| 11 | │ | │ | └── | main.m |
| 12 | │ | ├── | projectlayout.xcodeproj | |
| 13 | │ | │ | ├── | project.pbxproj |
| 14 | │ | │ | └── | xcshareddata |
| 15 | │ | │ |  | └── xcschemes |
| dieciséis | │ | │ |  | └── projectlayout.xcscheme |
| 17 | │ | └── | projectlayoutTests | |

|  |  |
| --- | --- |
| Empezando | 15 |

18 │ ├── Info.plist

19 │ └── projectlayoutTests.m

Nuestros principales puntos de contacto serán AppDelegate.m para el código de arranque (en el caso de que queramos personalizar nuestro mecanismo de carga, lo veremos más adelante), varios plists y nuestra xcassetspara los activos. Como se espera de un marco de plataforma cruzada como Reaccionar nativo, no hay mucha superficie nativa de manejar.



**Proyecto Xcode**

El diseño del proyecto Xcode no se correlaciona directamente a la estructura de directorios

|  |  |
| --- | --- |
| Empezando | dieciséis |

acabamos reviewd. Xcode mantiene una capa de metadatos para eso. Como se puede ver que hay elbibliotecas grupo que no existe físicamente, pero que existe en el diseño del proyecto dentro de Xcode. bibliotecas es donde va a agregar nuevas bibliotecas de la comunidad (o propio) proyectos que desea utilizar que tiene una parte nativa de ellos (ya veremos cómo hacer esto en el uso de sub-tema Componentes de la Comunidad), y main.jsbundlees un archivo que contiene la totalidad de su código de aplicación “compilado” estáticamente cuando se quiere enviar su aplicación a la producción.

Vamos a dar un rodeo y ver el código de arranque utiliza iOS con el fin de cargar la aplicación real Javascript.

* - (**BOOL**)solicitud:(UIApplication \*)solicitud didFinishLaunchingWithOption \

2 s:(NSDictionary \*)opciones de lanzamiento

3 {

4NSURL \*jsCodeLocation;

5jsCodeLocation = [NSURL URLWithString:@ "Http: // localhost: 8081 / index.ios.bun \

6 dle? = plataforma iOS y dev = true"];

* *// jsCodeLocation = [[NSBundle mainBundle] URLForResource: @ "principal" withExten \*

8 *sión: @ "jsbundle"];*

9

1. RCTRootView \*rootView = [[Alloc RCTRootView] initWithBundleURL: JsCodeLocati \
2. en

12 Nombre del módulo:@ "Projectlay \

1. fuera"

|  |  |
| --- | --- |
| 14 | initialProperties:nulo |
| 15 | opciones de lanzamiento: LaunchOption \ |

1. s];
2. yo.ventana = [[UIWindow alloc] initWithFrame: [UIScreen mainScreen] .bounds];
3. UIViewController \*RootViewController = [UIViewController nuevo];
4. rootViewController.view = rootView;
5. yo.window.rootViewController = RootViewController;
6. [yo.window makeKeyAndVisible];
7. **regreso** SÍ;
8. }
9. **@fin**

He quitado los comentarios muy útiles de este archivo para guardar las páginas, así que por favor encontrar el tiempo para leerlo textualmente de su propio árbol de código fuente. Se trata de un módulo de eventos generales de la aplicación, diseñado para controlar los eventos de aplicaciones céntricas comodidFinishLaunchingWithOptionsaquí, y eventos de notificaciones, eventos de fondo y así sucesivamente. Reaccionar tapones nativos endidFinishLaunchingWithOptions y configura el principal ventana y rootView.

La parte importante es la jsCodeLocationresolución. En nuestro caso, se apareja a una dirección IP, señalando y suponiendo que esta es su desarrollo local

|  |  |
| --- | --- |
| Empezando | 17 |

máquina. En otros casos este será el archivo de paquete actual hemos discutido antes, y si te gusta mantener las cosas ordenadas que querrá hacer una recopilación de destino, variable de entorno, o pragma para cambiar entre los dos (tales como “desarrollo” y “ lanzamiento"). Por ahora, que sea como es.

Aparte de eso, si usted es un extraño para iOS, recomiendo tomar un día para [conocer Xcode](http://www.raywenderlich.com/tutorials)[9](#page22). Por lo que yo sé, si se toma la cantidad de código y aplicaciones que se ha generado en esa plataforma, entonces es un IDE que le ofrezca tanto “tiempo aire” que es, probablemente, sólo segundo a Visual Studio.

Android es una bestia completamente diferente. Vamos a echar un vistazo a eso ahora.

**El Proyecto nativo de Android**

Aquí está el listado para nuestro proyecto React nativo generado, centrándose en el androide/ directorio.

* ├── androide

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | │ | ├── | aplicación |
| 3 | │ | │ | ├── build.gradle |
| 4 | │ | │ | ├── proguard-rules.pro |
| 5 | │ | │ | ├── react.gradle |
| 6 | │ | │ | └── src |
| 7 | │ | │ | └── principal |
| 8 | │ | │ | ├── AndroidManifest.xml |
| 9 | │ | │ | └── java |
| 10 | │ | │ | └── com |
| 11 | │ | │ | └── projectlayout |
| 12 | │ | │ | └── MainActivity.java |
| 13 | │ | ├── | build.gradle |
| 14 | │ | ├── | Gradle |
| 15 | │ | │ | └── envoltorio |
| dieciséis | │ | │ | ├── Gradle-wrapper.jar |
| 17 | │ | │ | └── gradle-wrapper.properties |
| 18 | │ | ├── | gradle.properties |
| 19 | │ | ├── | gradlew |
| 20 | │ | ├── | gradlew.bat |
| 21 | │ | └── | settings.gradle |

Androide hizo el movimiento de Eclipse y Ant uno o dos años atrás, en el nuevo y brillante Android estudio basado en la plataforma IntelliJ, y menos brillante (pero muy potente) Gradle proyecto. Si usted está viniendo de Java, a continuación, Gradle reemplaza Ant, Maven, y otras soluciones de administración de dependencias de Java diferentes. Se proporciona una flexibilidad sin fin y el poder en términos de dependencia

* <http://www.raywenderlich.com/tutorials>

|  |  |
| --- | --- |
| Empezando | 18 |

configuración y gestión de proyectos, mientras que la exposición de una esponjosa DSL (con la ayuda de Groovy, el lenguaje basado en JVM).

Esto es por qué hay tantos archivos como archivos de origen Gradle normales en el directorio del proyecto generado (que es una buena o una mala cosa, entonces? :-). Tome un momento para explorar el contenido desettings.gradle, app / build.gradle y app / react.gradle ya que estos son los archivos clave que va a la manipulación durante el desarrollo con todo lo que se refiere a la gestión de la dependencia - en otras palabras, a través de estos configuraremos bibliotecas externas, componentes y tareas de compilación de proyectos a medida y configuraciones.

Para tener sentido de él, recuerde estos puntos:

* settings.gradle - mantiene punteros a diversos subproyectos vamos a querer configurar. Claramente, esto será mayormente componentes externos con partes nativas; vamos a señalar a sunode\_modulescamino para que Gradle tendrá en cuenta sus proyectos de Java como parte de la construcción en general. Esto, además de las definiciones dentro deapp / build.gradle resolverá cualquier “símbolo indefinido” errores.
* app / build.gradle - aquí es donde especificaremos dependencias de nuestra aplicación; Tarros externos que nos gustaría utilizar (en el caso que estamos desarrollando partes nativas), las bibliotecas y los componentes de la comunidad y así sucesivamente. También vamos a querer configurar elementos específicos Android allí, como la versión del SDK, la compilación de destino y así sucesivamente. Esto sólo tiene sentido si ya está familiarizado con la plataforma Android. Si no es así, por favor, pasar un día[la construcción de una aplicación hola-mundo](https://developer.android.com/training/basics/firstapp/index.html)10 con Android nativo.
* app / react.gradle - Gradle le permite ejecutar tareas personalizadas de su propia. Va a mostrarles a través de la línea de comandos. Si tu$ Cd androide && ./gradlew Tareas usted será capaz de echar un vistazo a todas las tareas disponibles para usted y dentro de esas tareas que habrá reaccionarTareas. losreact.gradlearchivo es lo que los define; como tal también se puede modificar (pero probablemente

Nunca necesitará) lo que hay dentro: a saber, la bundleDebugJsAndAssets y bundleReleaseJsAndAssets Tareas.

Nuestra MainActivity.java archivo es el paralelo al de iOS AppDelegate.marchivo, que es agradable de ver; es elegante. Vamos a ver:

1. <https://developer.android.com/training/basics/firstapp/index.html>

|  |  |
| --- | --- |
| Empezando | 19 |

* **clase pública Actividad principal se extiende** ReactActivity{
* @Anular
* **protegido** CuerdagetMainComponentName() {

4**regreso** "Projectlayout";

* }

6

* @Anular
* **protegido booleano** getUseDeveloperSupport() {

9**regreso** BuildConfig.DEPURAR;

1. }
2. @Anular
3. **protegido** Lista<ReactPackage>getPackages() {
4. **regreso** Las matrices. <ReactPackage>asList(
5. **nuevo** MainReactPackage()
6. );
7. }
8. }

La eliminación de los comentarios, que no se ve mucho. La mayor parte de la obra gira en tornogetPackages donde se le pide con frecuencia que agregar componentes personalizados propia paquetes (en caso de que tengan una parte nativa). Reaccionar modelo de extensión nativa nativa y la infraestructura es fascinante por lo que no se preocupe - que inmersión profunda en esto y en la construcción de sus propios componentes nativos (UI) y módulos posteriores, por lo que mantener apretado.

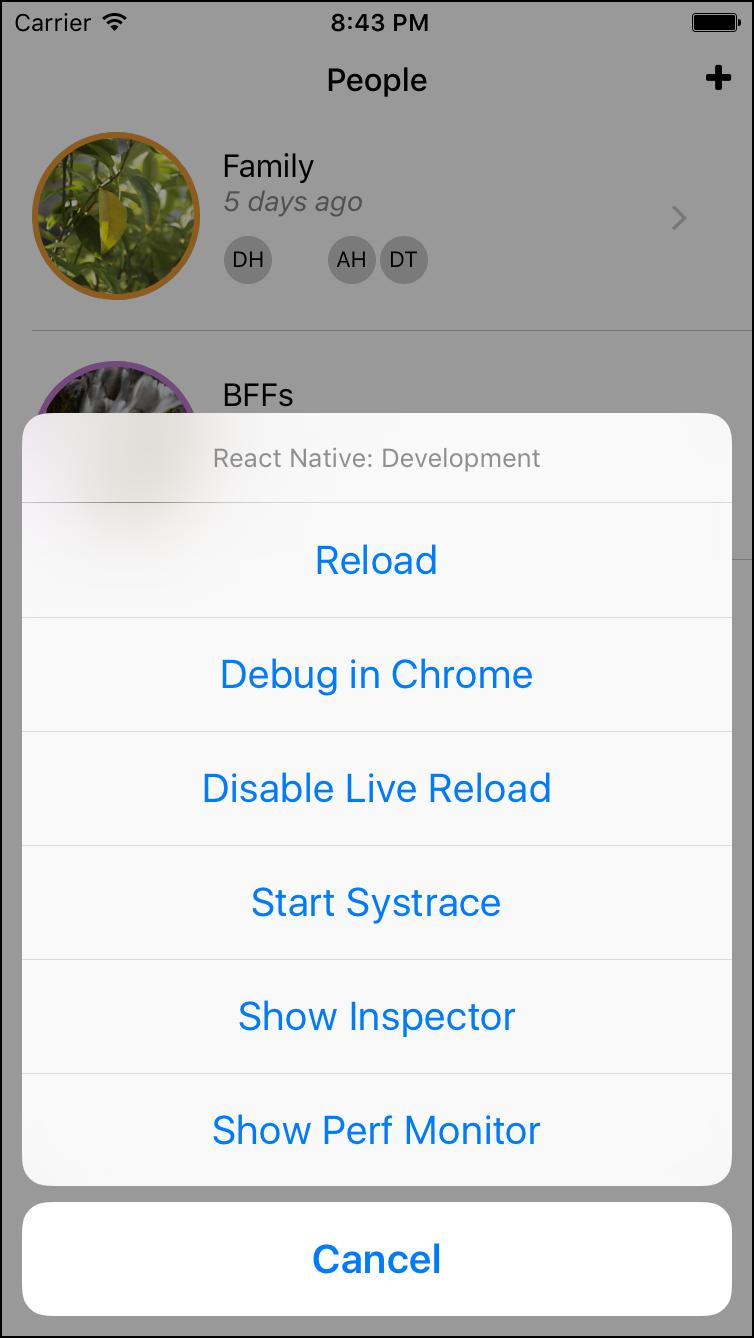
**Estampación**

Fuimos el Reaccionar nativo CLI, y cubre el período previo y construir las fases del ciclo de desarrollo típico muy bien. Además, ya que realmente estamos manejando tres bases de código diferentes: Javascript, y es de esperar muy poco de Objective-C y Java, sería prudente tener un vistazo a las diversas herramientas de depuración y alrededor de ellos.

**Simulador y emulador de Herramientas de Desarrollo**

Desde sus primeros días Reaccionar nativo proporciona un desarrollador de experi-encia sin emparejar en el móvil si se toma el hecho de que se trataba de un proyecto tan joven. Los mensajes de error que son tan útiles que se siente telepática, una función de herramientas de desarrollo, depuración con Chrome, y el inspector y perfilado utillaje allí mismo, en el dispositivo; si pasó a ocuparse de los problemas de rendimiento antes, entonces usted sabe que tiene las herramientas de perfilado allí mismo, en su aplicación y en su dispositivo no tiene precio (herramientas externas pueden introducir una sobrecarga, y, obviamente, tratando de medir el rendimiento de un simulador es un no-go) .

|  |  |
| --- | --- |
| Empezando | 20 |

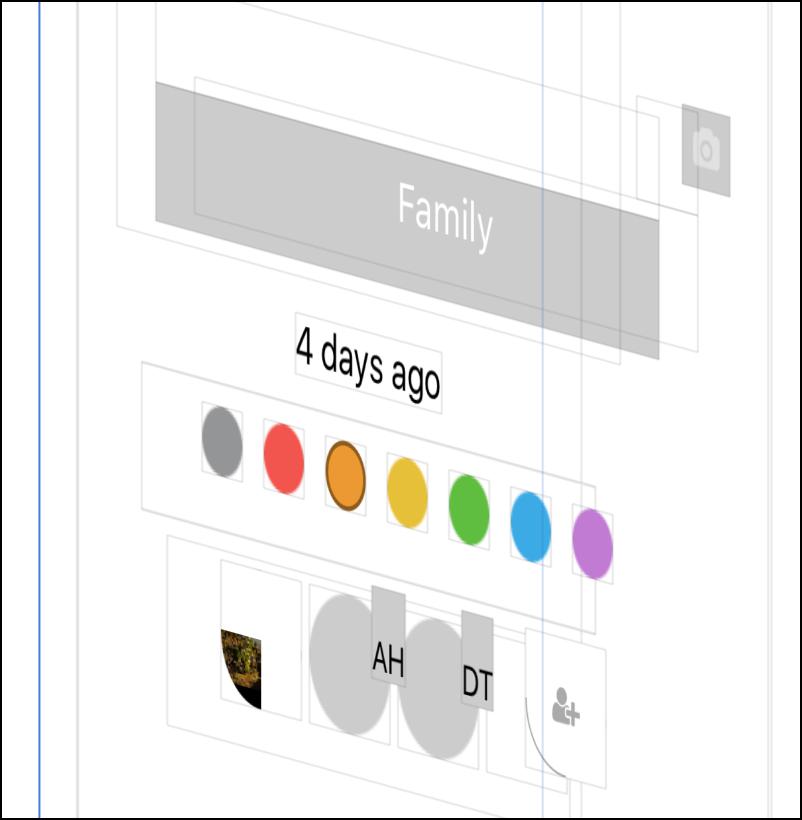


**Una función de depuración Herramientas**

La herramienta inspector le permite golpear a la prueba áreas de interfaz de usuario en vivo y ver un desglose de sus propiedades y la jerarquía, al igual que la herramienta de inspección de cromo (ir en su navegador Chrome y pulsa Shift-Cmd-C y luego intente hacer clic en el interior de un sitio web en algún lugar).

En iOS, se puede combinar eso con Xcode increíbles herramientas de depuración visuales (mientras la aplicación se está ejecutando a través de Xcode Cmd-R, ir Depurar -> Ver depuración -> Capturar vista Jerarquía), Se puede volcar el árbol visual nativo de la siguiente manera:

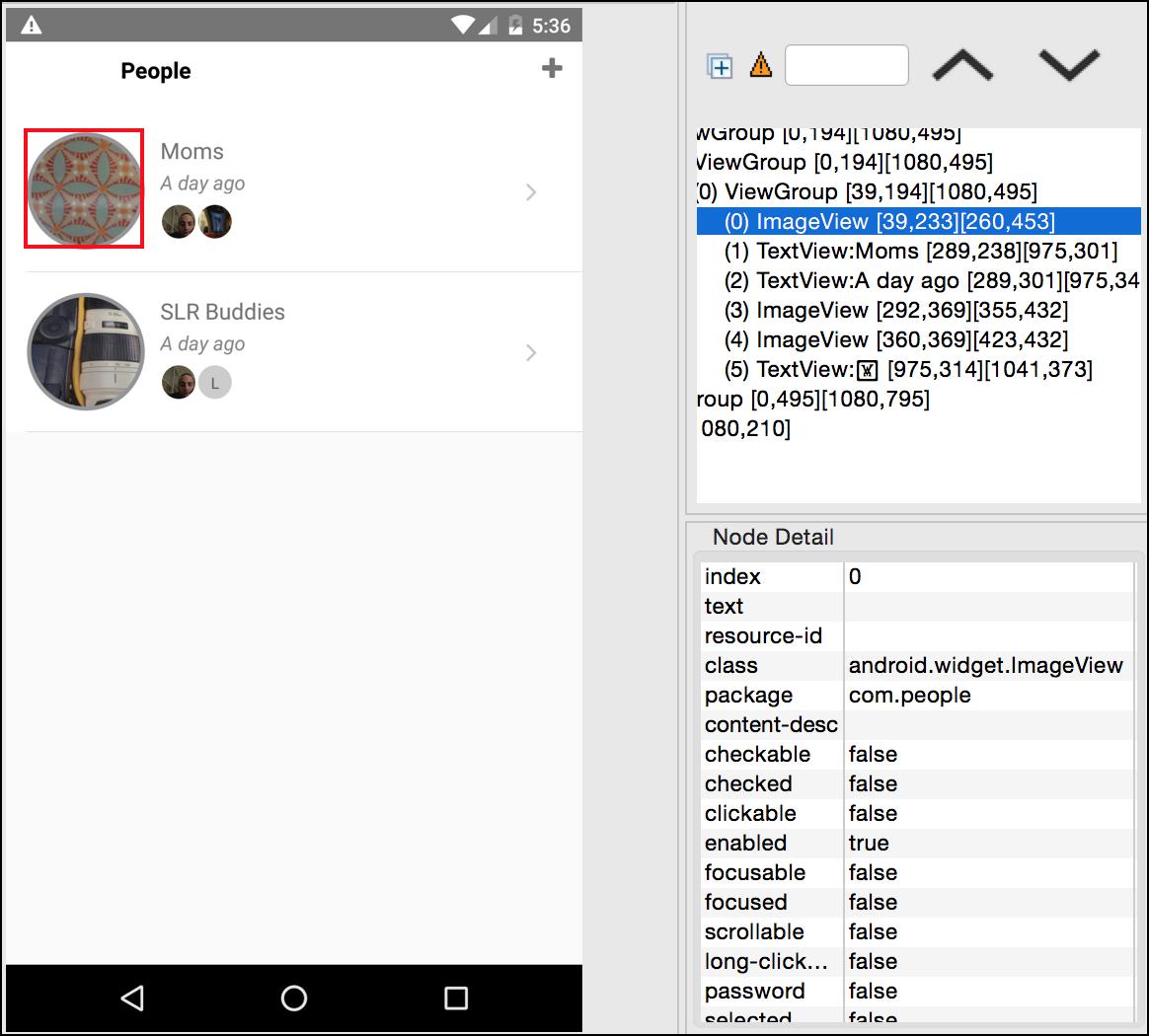
|  |  |
| --- | --- |
| Empezando | 21 |



**Desglose de los componentes visuales**

Y en Android, que se puede combinar con DDMS (Herramientas núcleo duro de Android para el rendimiento, depuración, análisis y así sucesivamente), seleccionando el dispositivo en el menú de dispositivos y haciendo clic en el icono que se parece a una pila de dispositivos móviles (no me pregunte por qué ...):

|  |  |
| --- | --- |
| Empezando | 22 |

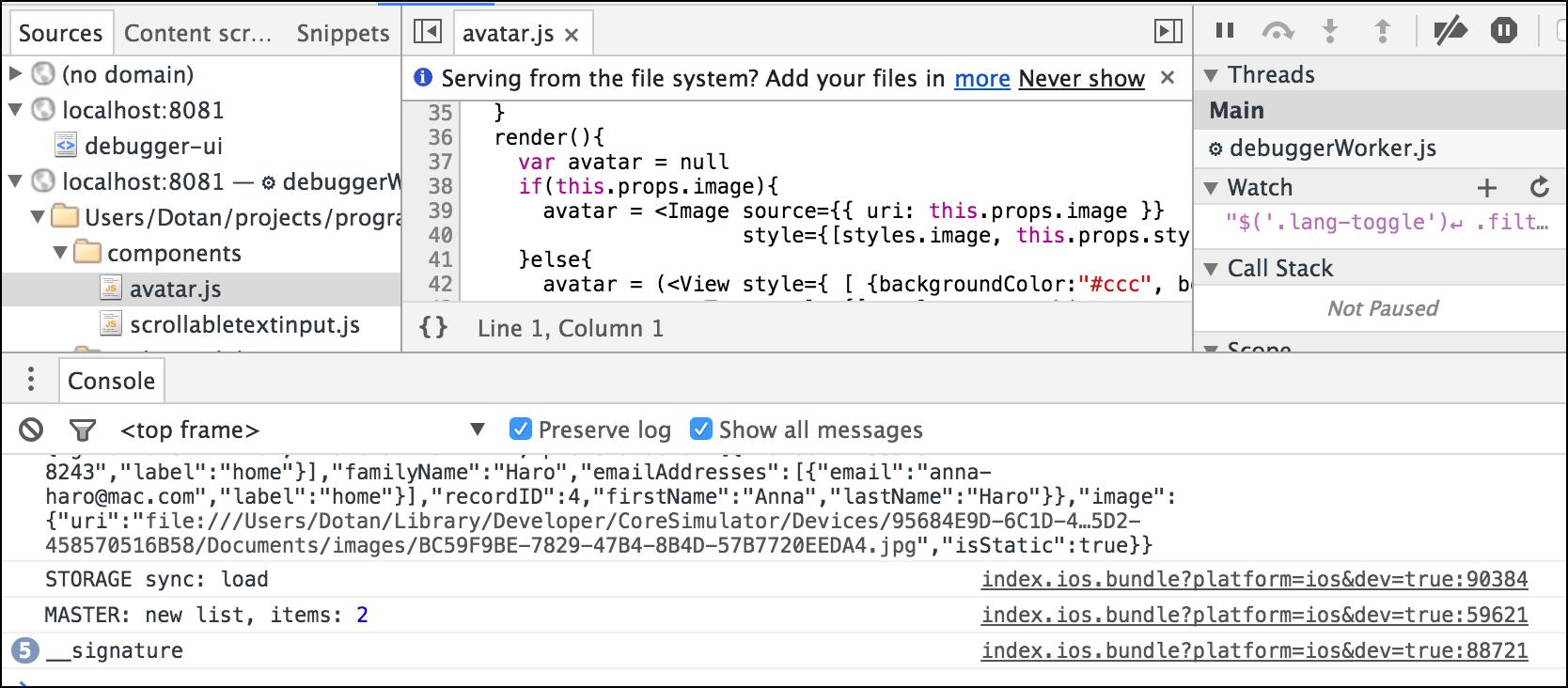


**Volcado de diseño de interfaz de usuario de Android DDMS**

Esta técnica permite realizar un vicio maniobra donde se puede atacar un problema visual, tanto en el lado nativo y Javascript y el uso de cada uno para complementar la otra la información que falta.

Por último, podemos utilizar Chrome como un depurador para todos nuestros Reaccionar código nativo Javascript:

|  |  |
| --- | --- |
| Empezando | 23 |



**Herramientas para desarrolladores de Chrome**

Esto significa echar un vistazo a los mensajes de registro, establecer puntos de interrupción en el código real Reaccionar, relojes, pasando a través y más. Incluso hay una manera de volcar una huella con herramientas Systrace y visualización de perfiles de Google, pero vamos a dejar las cosas espeluznante para un tipo diferente de libro o entrada de blog, es probable que no se necesita ese tipo de ayuda bajo nivel de optimización del rendimiento :)

No hablamos acerca de lo más sencillo posible hacerlo - la tala. En la vida real que va a utilizarconsole.log mucho, y la salida es en sí deja ver a través de Xcode mientras se ejecuta la aplicación, o en el caso de Android, ya sea con Android Studio (widget de la ventana Logcat) oa través de la línea de comandos con ADB, La herramienta súper androide (en realidad, ADBes realmente útil para conocer, sobre todo si eres un usuario de Android a sí mismo; puede automatizar muchas de las tareas diarias con él). Ejecutar esto en una ventana de terminal libre:

* psANDROID\_HOME/ Plataforma de herramientas / LogCat ADB \*: S ReactNative: ReactNativeJS V: V

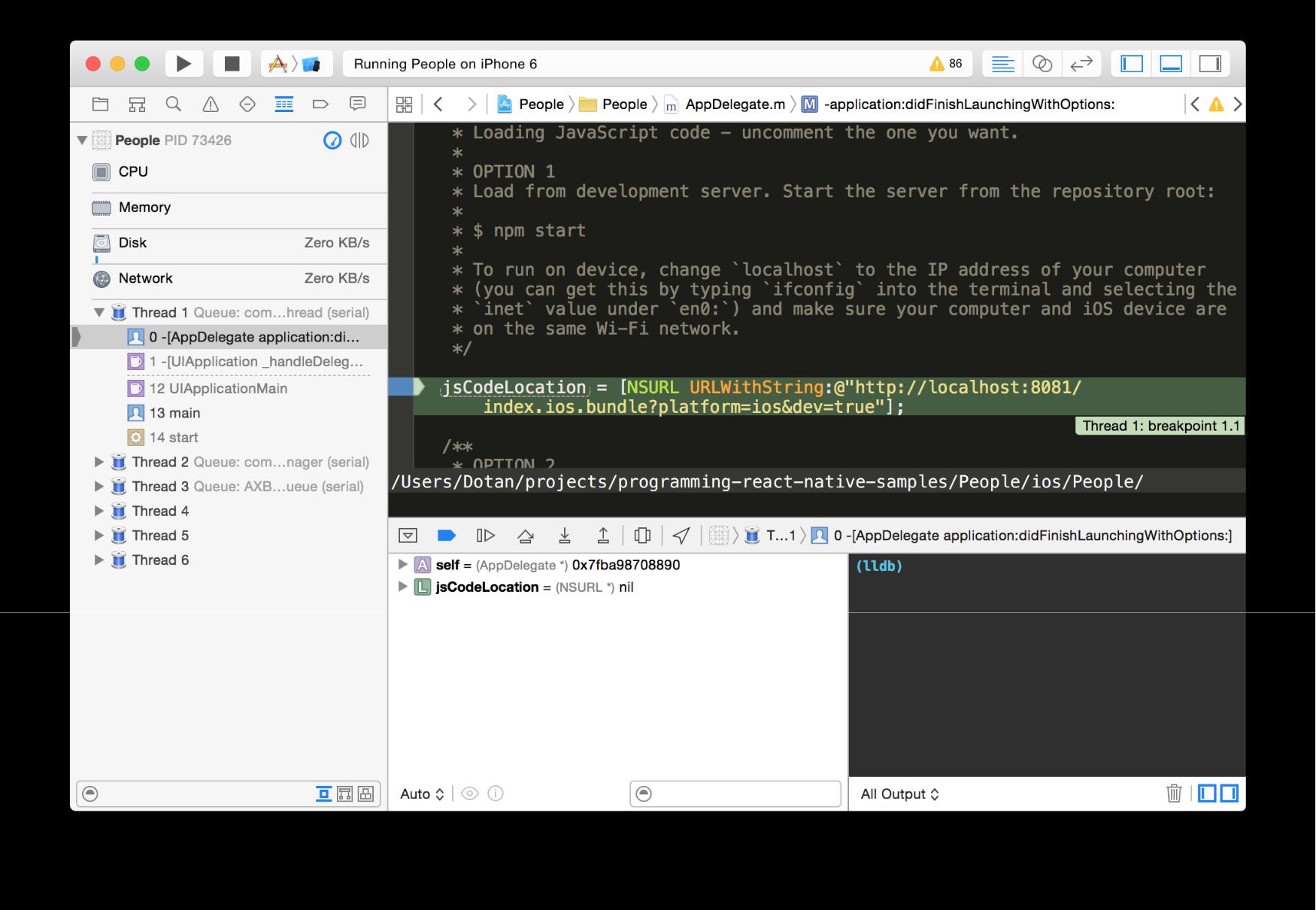
Si no le gusta tanto para la visualización de registros en vivo, hemos acabamos de aprender que se puede utilizar Chrome para la depuración y también ver los registros de - tratar en su lugar.

**Depuración nativa**

Hubo un tiempo en que yo temblar antes de cubrir un tema tan. depuración nativa en un SDK móvil multiplataforma. Sin embargo, esto no es un tiempo, y el equipo Reaccionar nativo ha hecho un gran trabajo de tal manera que todo lo que tengo que decir es esto: si se está ejecutando la aplicación y utiliza una API nativa a través de Javascript, y que corrió la aplicación a través de la IDE (Xcode o Android Studio) simplemente pulsando “Ejecutar”, y establecer un punto de interrupción en cualquier código nativo - que se romperá en ese código, y que hará que el sentido del mundo (es decir, las variables locales, los contextos, hilos, etc. ). ¡Prestigio!

Así es como se ve en Xcode:

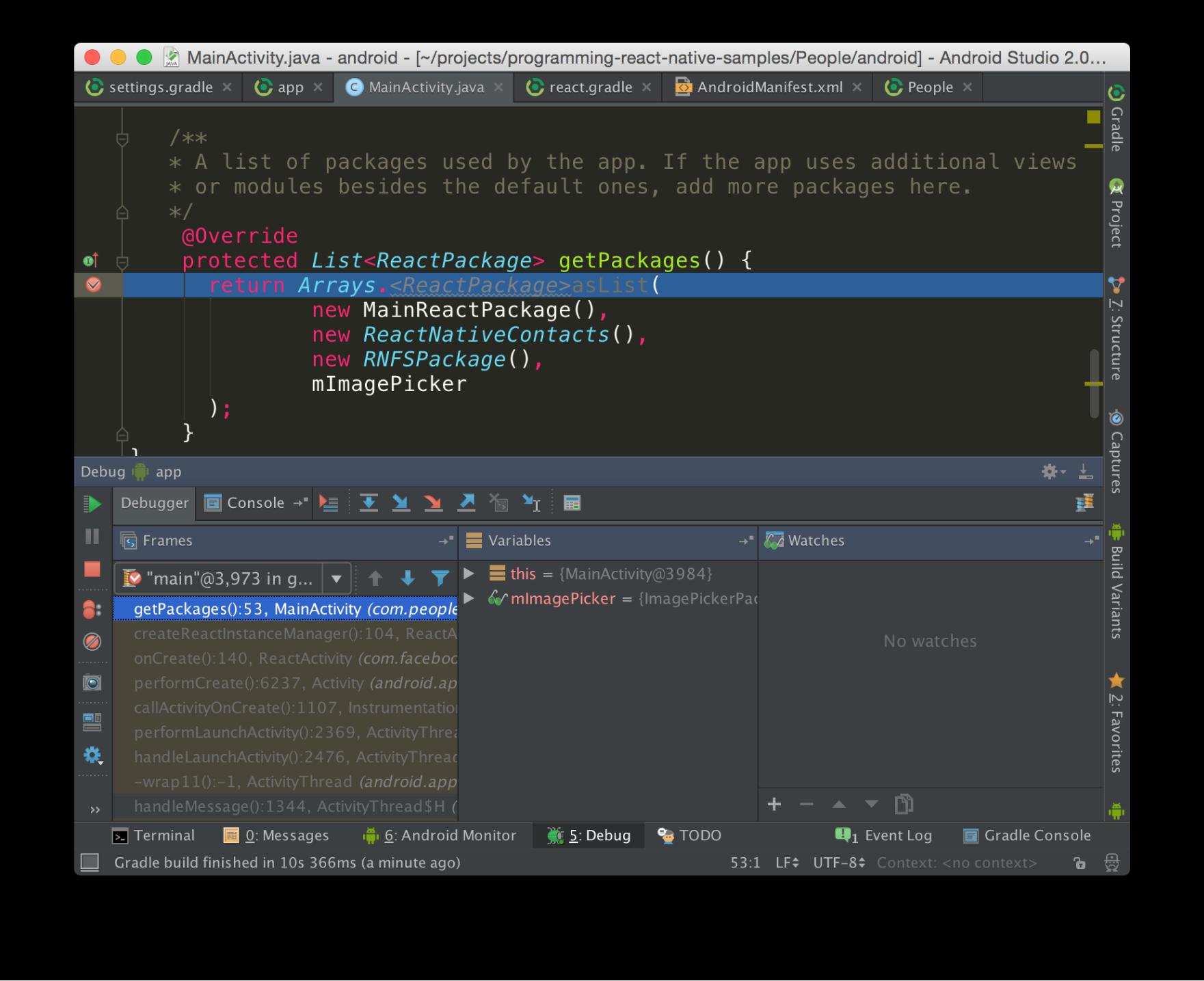
|  |  |
| --- | --- |
| Empezando | 24 |



**Depuración con Xcode**

Y su homólogo de Android, Android Studio:

|  |  |
| --- | --- |
| Empezando | 25 |



**Depuración con Android Studio**

Por lo tanto, si usted recuerda atrás cuando discutimos la gestión androide y Run-ios SubCOM-mands en el reaccionar nativo CLI, esto significa que hay todavía un gran valor para el uso del IDE detrás de cada plataforma móvil, por lo que no renunciar a ella tan rápidamente.

**Resumen**

En este capítulo hemos cubierto el diseño del proyecto React nativo, construida en los útiles, las herramientas plataformas nativas y cómo encaja todo. También hablamos sobre algunas técnicas que puede utilizar para mejorar su experiencia de desarrollo, además de lo utillaje está disponible para hacerlo.

Con todo, lo mejor que puedes hacer ahora es en esoUn par de proyectos, la basura hacia abajo por ajustar con todo lo que se ve tweakable, ejecutarlos a través de Xcode o Android de estudio, establecer puntos de interrupción, y explorar estos grandes entornos de desarrollo. Realmente, no puedo recordar un momento en que eran IDE esta grande - y yo soy un usuario de Vim !. Por lo tanto, no dude en tener un ir en él y nos encontraremos de nuevo aquí para más.

**Edificio Reaccionar componentes nativos**

En este capítulo, vamos a pasar por una breve historia de los marcos de Javascript, destacamos cómo Reaccionar es diferente, introducir Reaccionar Katas nativos - una experiencia de aprendizaje lúdico para Reaccionar nativo que he creado, y lo sigue a uno de los katas.

**Una historia de Squashed Javascript Frameworks**

Reaccionar cambiamos la forma en que construimos aplicaciones Javascript. En primer lugar, hubo jQuery, que resumieron cómo trabajamos en contra de un navegador. jQuery se aseguró de que fuimos capaces de manipular elementos DOM sin preocuparse por todo el DOM a la vez (Document Object Model), atravesarlo, cambiarlo, eventos de palo y devoluciones de llamada en él, y todo ello con una magnífica API que es sorprendente - hasta ese momento en el tiempo, en realidad no existe.

Con HTML5 cambio en, el cambio de los navegadores, las normas y prácticas de desarrollo en torno a 2011, hubo una necesidad de gestionar aplicaciones aún más grandes. Si todavía se utiliza jQuery para eso, usted se encontraría el manejo de cubos de Estado derramado todo su código, y lo más importante, todo el DOM - y por desgracia, que pertenecía a la DOM del navegador, y no usted. Y, usted puede apostar cada navegador gustado tener sus propias peculiaridades de todo el DOM. El título de “desarrollador front-end” surgió poco después, porque ahora se necesitaba una persona entera con un cuerpo lleno de habilidades para manejar ese tipo de complejidad.

Y luego comenzó la era de los marcos. los desarrolladores de aplicaciones para usuario necesarios para refinar su herramienta de correa recién nacido. Que tenían que hacer un trabajo eficiente y consolidar y abstraer las tediosas tareas repetitivas, que eran-ing realizar cada día. como marcos[Backbone.js](http://backbonejs.org)[11](#page31), [Batman.js](http://batmanjs.org)[12](#page31), [Ember.js](http://emberjs.com)[13](#page31), y después [angularjs](https://angularjs.org)[14](#page31) estaban todos corriente principal de convertirse en un lapso de tan sólo 2 años, la creación de una inundación masiva de ideas y soluciones.

Pero también, un caos imposible. Estos marcos abstraídas a cabo los navegadores, el trabajo DOM, presentó MVC en el lado del cliente, pero en mi opinión, al margen de tal Ember.js, realmente no 'resolver' el desarrollo del lado del cliente. Hasta Reaccionar dio la vuelta.

1. <http://backbonejs.org>
2. <http://batmanjs.org>
3. <http://emberjs.com>
4. <https://angularjs.org>

26

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 27 |

**React.js**

En el lanzamiento, Reaccionar tomó un camino menos ambiciosa que cualquiera de sus rivales de gran peso como angularjs o Ember.js. A primera vista, sólo había una capa de vista, algún tipo de mecanismo de eventos, y JSX - Reaccionar el lenguaje de marcado sabor. Muchos pensaron que era un reemplazo Reaccionar vista de capas en la arquitectura común de entonces MVC.

Luego, en un segundo aspecto y un año más tarde, verías que reaccionan estado también se maneja, composición de componentes, el estilo, mientras que la optimización del rendimiento y la manipulación DOM se resumieron y hecho por ti. Esta es la bonanza API libro de texto. La trampa de éxito.

Y, en un tercer aspecto y un par de años más tarde, nos dimos cuenta de que por ser mínima, Reaccionar permitió la comunidad para evolucionar de forma creativa y desarrollar opiniones y un ecosistema masivo creció en una velocidad nunca antes vista. Con los nuevos patrones de diseño como el flujo y patrones funcionales tales como Redux, todos los que rinde homenaje a[cosas](https://bitquabit.com/post/the-more-things-change/)[15](#page32) [ese](https://en.wikipedia.org/wiki/Lisp_(programming_language))[dieciséis](#page32) ya trabajar en el software de los últimos 30 años. Muy pronto, incluso el marco respaldado por Google empresarial adoptada angular llega a experimentar un descenso.

**reaccionar componentes**

Reaccionar le permite hacer los componentes de una manera muy individual e independiente. Nunca antes había un marco de JavaScript que le permiten hacer los componentes declarativa y funcional de tal manera divertida. Puede crear vistas y no se preocupan por el navegador o DOM, estilo y no se preocupan por CSS, e implementar la funcionalidad de una manera completamente encapsulado no preocuparse por ninguna de sus dependencias.

Mientras que el objetivo de este libro no era enseñar que reaccionar, me solté con una “falta algo” sentimiento. Quería que le permite experimentar la alegría de construir una aplicación móvil que sólo reaccionan nativo permite, incluso antes de sumergirse en la construcción de una aplicación real - y no se puede hacer en un libro porque no es interactiva - es demasiado “seco”; no deje que cualquier otro libro te engañe.

Es por eso que he creado [Reaccionar nativo Katas](https://github.com/jondot/ReactNativeKatas)[17](#page32). Es un proyecto de código abierto que sirve como una experiencia de aprendizaje complementario de este libro (y también una experiencia independiente), que se puede ir a través de sí mismo y tener sus amigos probar - ya sean diseñadores, programadores principiantes, o sólo la gente del producto (con una introducción mínima de Javascript, por supuesto).

Reaccionar nativo Katas es una experiencia práctica, de inmersión y lúdico, que enseña sobre la construcción de reaccionar componentes nativos mientras se hace. [Usted puede](https://github.com/jondot/ReactNativeKatas)

1. [https://bitquabit.com/post/the-more-things-change/](https://github.com/jondot/ReactNativeKatas)
2. <https://en.wikipedia.org/wiki/Lisp_(programming_language)>
3. <https://github.com/jondot/ReactNativeKatas>

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 28 |

descargar el repositorio aquí[18](#page33).

Vamos a dedicar el resto de este capítulo para mostrar cómo funciona en un solo Kata porque la mayoría de la diversión no está en estas páginas: está en Github y en su editor. Incluso se puede ignorar el resto del capítulo, tienen fantásticas tiempo jugando con los Katas en este momento!

**Completar un Kata**

Vamos a pasar por la primera Kata, sólo para probar las aguas. Confío en que usted continuará por su cuenta después de esto. Asumo que ya tiene una[reaccio- nado de trabajoconfiguración nativa](https://facebook.github.io/react-native/docs/getting-started.html)[19](#page33).

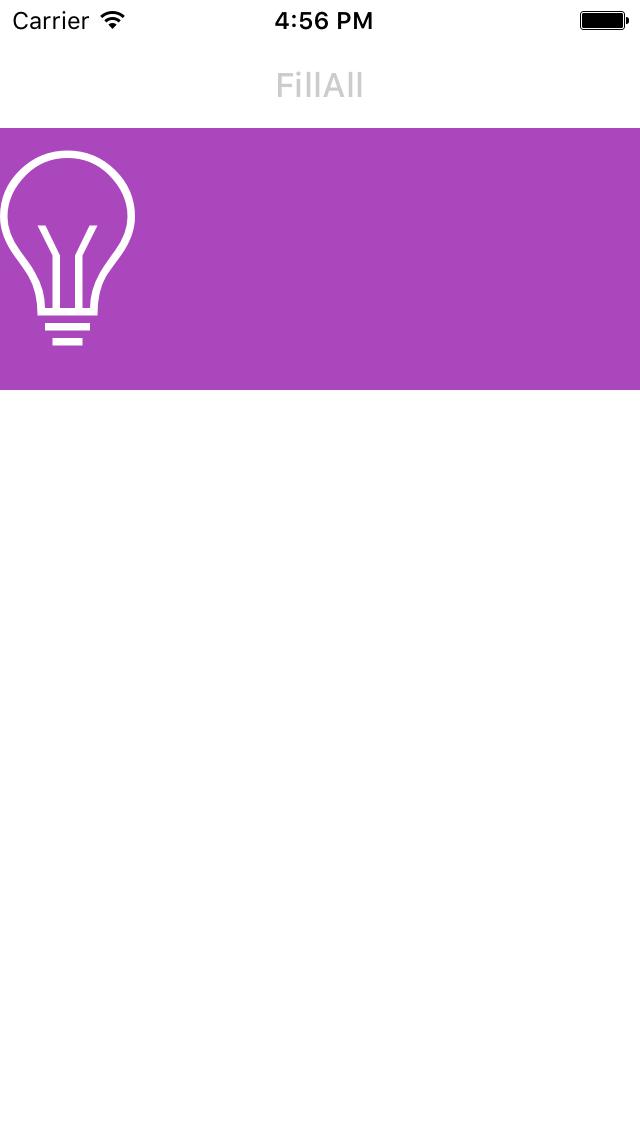
* proyecto clon con el proyecto $ Git clone https://github.com/jondot/ReactNativeKatas
* cd ReactNativeKatas y i NPM para la instalación de las dependencias.
* Ejecutar el proyecto a través de Xcode o reaccionar nativos run-ios
* Después de ejecutar el proyecto, encender Recargar vivo (Ctrl + Cmd + Z para

menú de desarrollador en simulador)

A continuación, se enfrenta con el primer Kata, la versión de prueba o la versión “sin resolver”:

1. <https://github.com/jondot/ReactNativeKatas>
2. <https://facebook.github.io/react-native/docs/getting-started.html>

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 29 |



**Un Kata espera de ser resuelto**

Toque en cualquier lugar, para ver lo que la solución debe verse como:

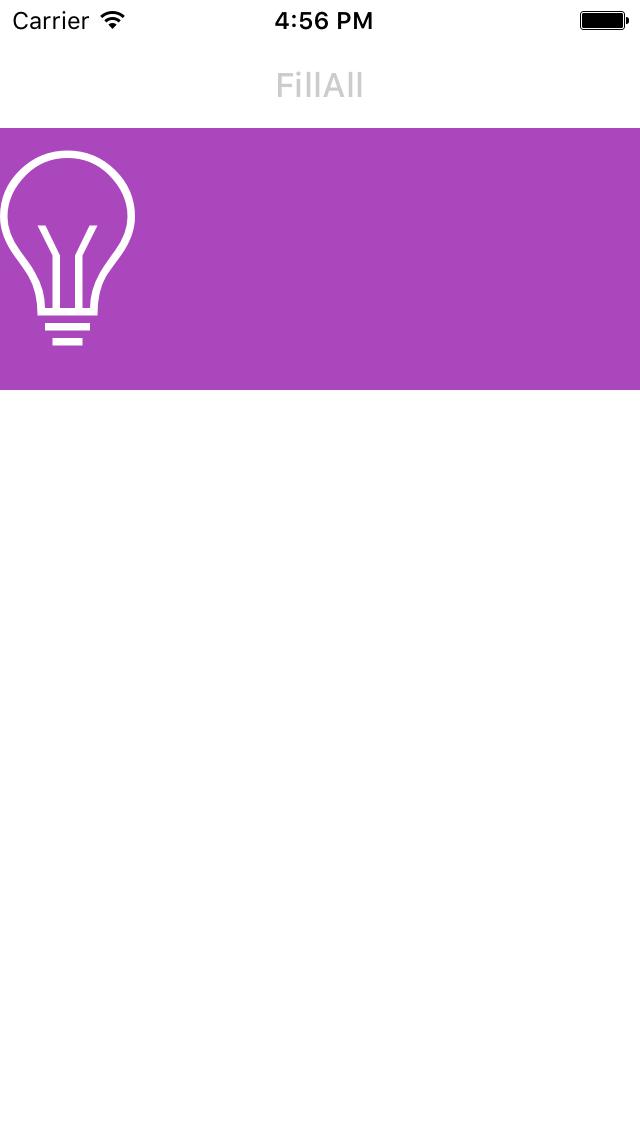
|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 30 |



**Un Kata resuelto**

Y pulse de nuevo para mostrar la versión de prueba:

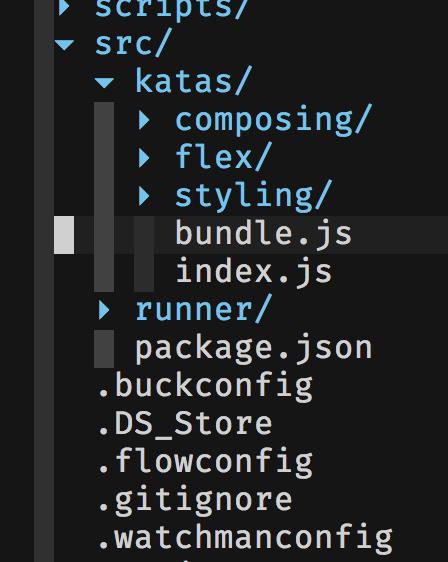
|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 31 |



**Un Kata espera de ser resuelto**

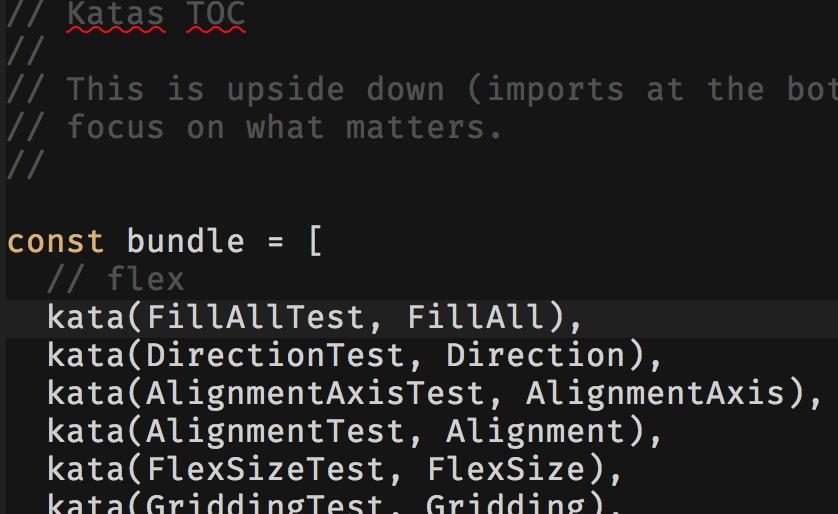
El siguiente paso, vamos a orientado. Localizarbundle.js con el árbol de código fuente correspondiente, en src / katas. Esto listará todos sus Katas pendiente en varios temas: peinar, flexión, y la composición.

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 32 |



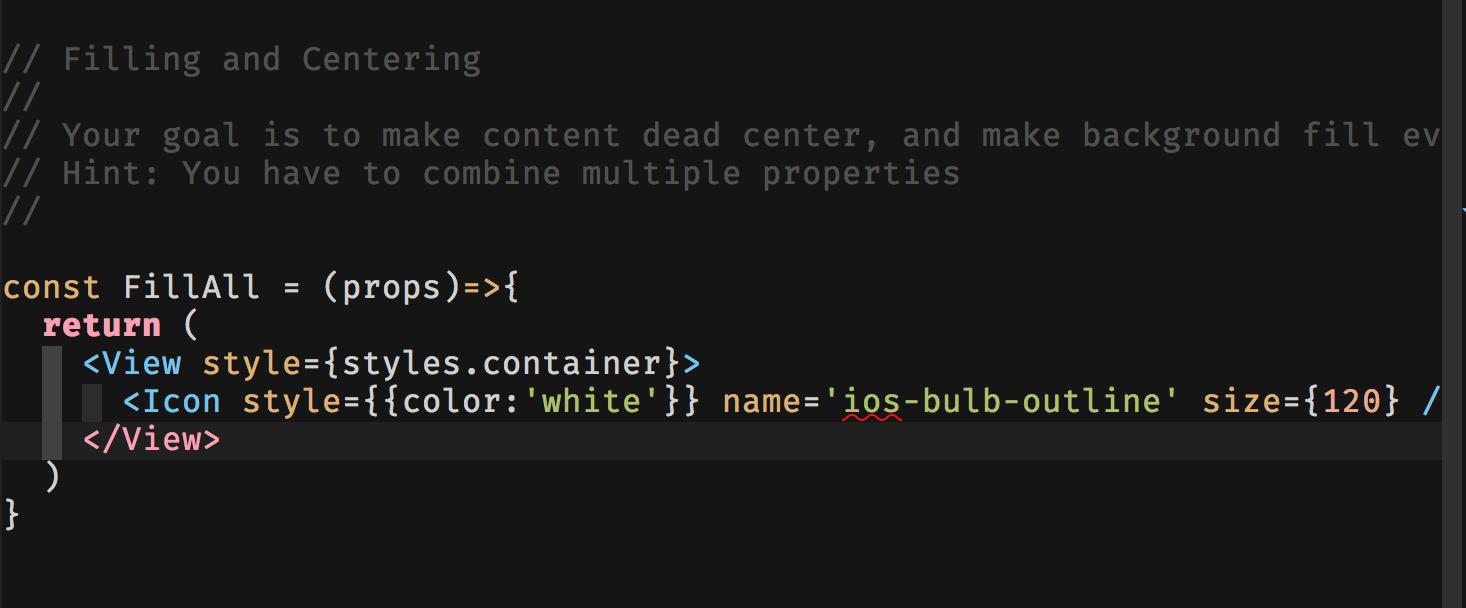
**El archivo de paquete**

La apertura del paquete, se encuentra esta:



**Listado de todos los Katas**

Y entonces, vemos la primera es la Kata FillAllKata. Queremos abrir el.prueba versión de la misma, que es lo que hay que modificar.



**Código para la prueba de Kata**

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 33 |

Si edita nada y guardar en ese archivo, Reaccionar nativo lo detectará y Live Recargar entrará en funcionamiento y volver a cargar la aplicación. El resultado final: el simulador de iOS se actualiza automáticamente. Reaccionar nativo Katas tiene un mecanismo incorporado para detectar si se ha resuelto el Kata correctamente, y avanzará a la siguiente Kata una vez que tuvo éxito hasta que todos los katas están acabados.

Vamos a tratar de resolver el primer Kata.

* // Su objetivo es hacer que el contenido del punto muerto, y hacer relleno de fondo everyth \

2 En g

3 // Consejo: Usted tiene que combinar varias propiedades

Así que vamos a llenar todo. Flex es nuestro modelo de diseño, por lo que no dude en Google 'diseño flex' ahora. Katas tiene en cuenta que no viene preparado, por lo que se encontrará una gran cantidad de buscar en Google - esto es exactamente lo que queremos que haga. Se trata de cómo se realiza el aprendizaje por la práctica :)

*Pausa.*

Después de buscar en Google para el diseño de la flexión, estoy seguro de que se enteró de Flex: 1. Esta es la instrucción básica de flexión, para que la vista en virtud de que el estilo asume ninguna realestate que puede, siempre y cuando no hay puntos de vista que compiten con los pesos de la competencia.

Por lo tanto, vamos a cambiar los estilos a esto:

* estilos const = StyleSheet.create ({

2envase: {

3backgroundColor: colores [0],

4flex: 1,

* },
* texto:{
* color blanco',
* textAlign: 'centro'

9}

1. });

Guardar, y ver el Kata se actualiza. Ahora el fondo cubre todo, pero estas viendo este Kata - lo que significa que no se resolvió. Se ha dicho que también tenemos que centrar todo. Google es tu amigo ahora, trate 'de centrado flex'.

*Pausa.*

Bueno eso es justifyContent, Pero sólo te estoy diciendo esto ahora - y ni siquiera se supone que tienes que leer esta parte del capítulo, ya que está lleno de spoilers. Los insto a seguir adelante y hacer el solo de Katas.

¿Aún aquí? Muy bien, vamos a seguir adelante!

La fijación de los estilos de nuevo:

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 34 |

* estilos const = StyleSheet.create ({

2envase: {

3backgroundColor: colores [0],

4flex: 1,

5justifyContent: 'centro',

* },
* texto:{
* color blanco',
* textAlign: 'centro'

10}

11 });

Guardar de nuevo. Eso hizo que el contenido de alinear al centro, pero no en ambos ejes. La pista se dice que hay que combinar algunos estilos para conseguir eso. Una vez más, Google.

*Pausa.*

Googlear alrededor, nos damos cuenta que nos estamos perdiendo alignItems ¡también!

Seguramente lo suficiente, esta es la solución definitiva:

* estilos const = StyleSheet.create ({

2envase: {

3backgroundColor: colores [0],

4flex: 1,

5justifyContent: 'centro',

6alignItems: 'centro',

* },
* texto:{
* color blanco',

1. textAlign: 'centro'
2. }
3. });

Y ya está! Hay un nuevo Kata de resolver.

|  |  |
| --- | --- |
| Edificio Reaccionar componentes nativos | 35 |



**Un Kata resuelto**

Así que acaba de terminar una diversión, aprender por las malas, la experiencia de la construcción de un componente nativo reaccionar. Y ahora usted está en su propia (no realmente: siempre se puede mirar el código para la solución de Kata!).

Vaya por delante y divertirse, y cuando haya terminado se puede volver a leer este libro con más confianza que antes! (O simplemente puede seguir leyendo, y hacer los katas más tarde. O no hacer los katas. Uhm .. usted consigue el punto :).

**La aplicación Contactos**

En este capítulo vamos a cubrir la aplicación Contactos, que es la aplicación que usaremos a lo largo de las partes “elaborar” del libro. Que vamos a analizar, y luego avanza sobre ella poco a poco para que usted comprenderá todas las decisiones y consideraciones es necesario aplicar mientras se construye una aplicación nativa reaccionar.

También vamos a cubrir lo que la hace funcionar - sus características y componentes básicos técnicos, y también a discutir hasta qué punto se puede llevar adelante por su cuenta. En los siguientes capítulos vamos a empezar la creación de nuestro entorno, examinamos nuestra aplicación, y hacer una inmersión profunda.

**Caracteristicas**

Vamos a repasar nuestras características:

* Crear y administrar grupos de contactos
* Personalizar los grupos de imagen personalizada código de colores, etc.
* Escoja una imagen de la cámara en marcha o tomar uno vivo
* Elija los contactos en la libreta de direcciones existente del teléfono
* Interfaz de usuario consistente tanto para iOS y Android con animaciones y enrutamiento
* Deslizar para eliminar (IOS-like)

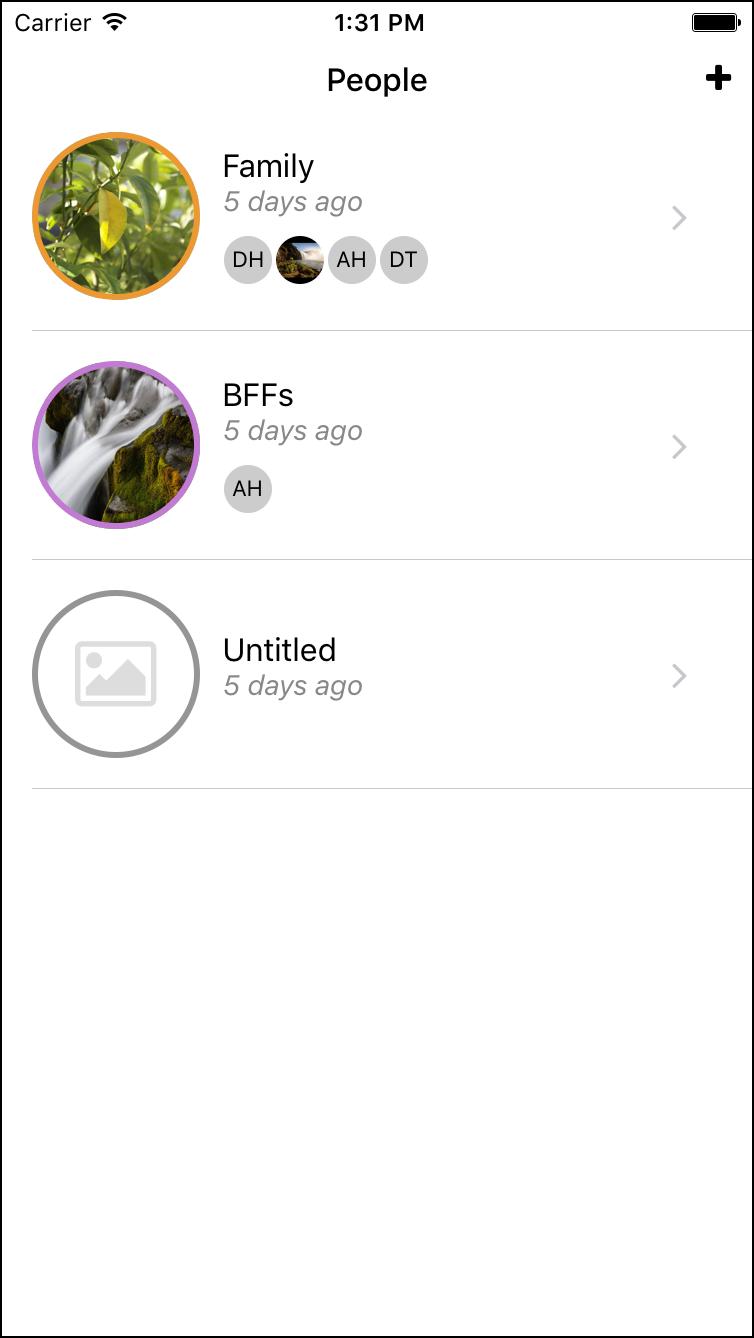
**Producto**

Vamos a pasar rápidamente por el producto, al observar las capturas de pantalla, agarrarse a algo!

Nuestra pantalla de la aplicación principal. Esta es la lista de grupos, o ver “Maestro” en la relación maestro-detalle (más sobre esto más adelante). visuales notables son la codificación de la imagen de usuario principal del grupo (o cubierta), un diseño bastante complejo lista de células con fotos de contacto integrados y más color.

36

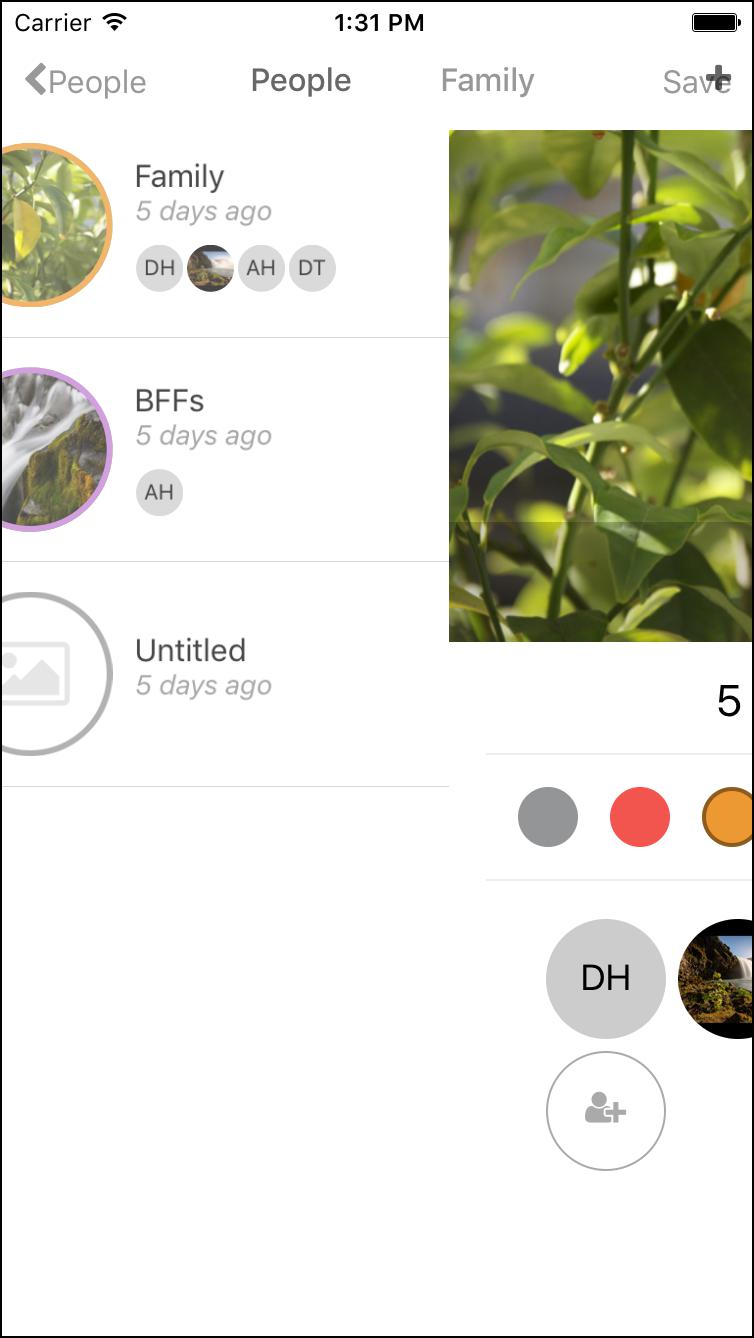
|  |  |
| --- | --- |
| La aplicación Contactos | 37 |



**Lista grupos (Master)**

La transición a la pantalla de edición de grupo, que es nuestro punto de vista detalle en la relación maestro-detalle. Animaciones corren mantequilla suave, con los típicos izquierda facilidad de salida y fade-out, seguido de fundido de entrada de la caja.

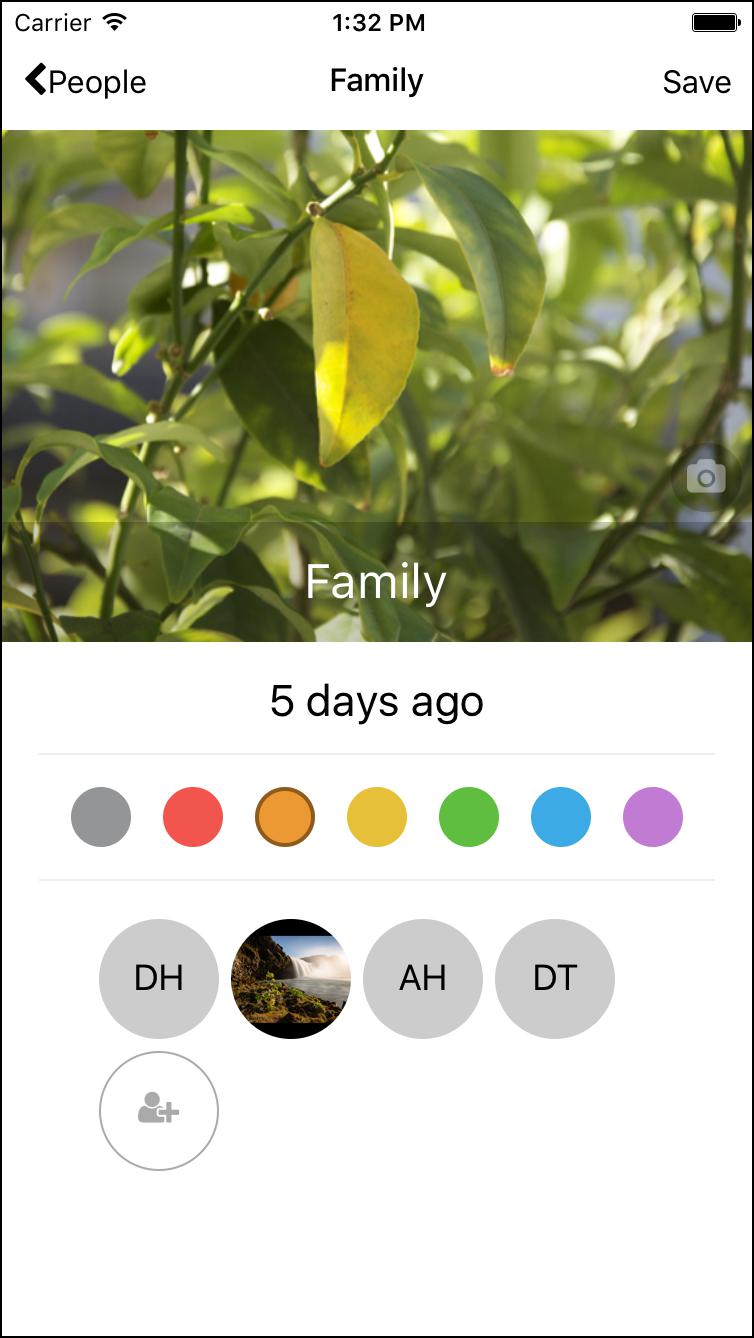
|  |  |
| --- | --- |
| La aplicación Contactos | 38 |



**Transición al Detalle**

Nuestra pantalla grupo listo para ser editado. El comportamiento típico iOS dicte el botón derecho “Guardar” en la barra de navegación, vamos a llevar esto a la versión de Android también.

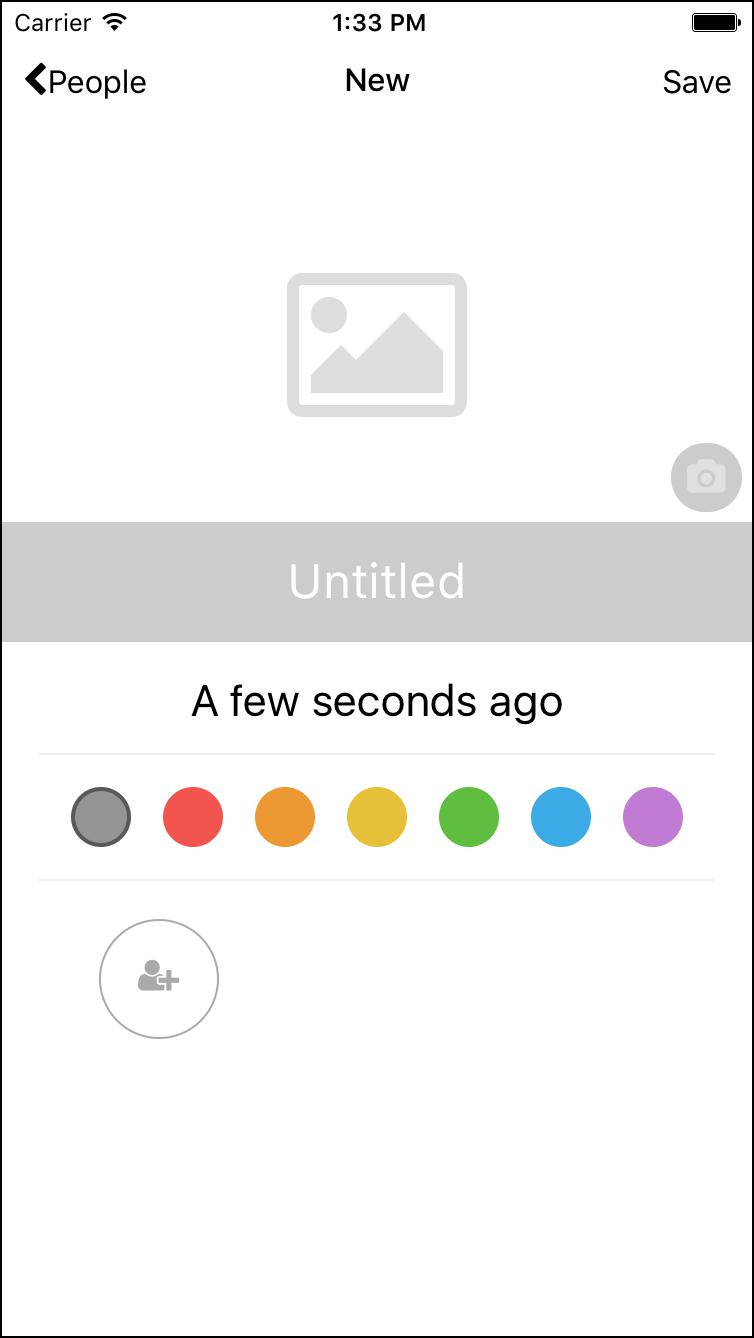
|  |  |
| --- | --- |
| La aplicación Contactos | 39 |



**Grupo de pantallas (Detalle)**

Y así es como la misma pantalla grupo parece vacío, al crear una nueva.

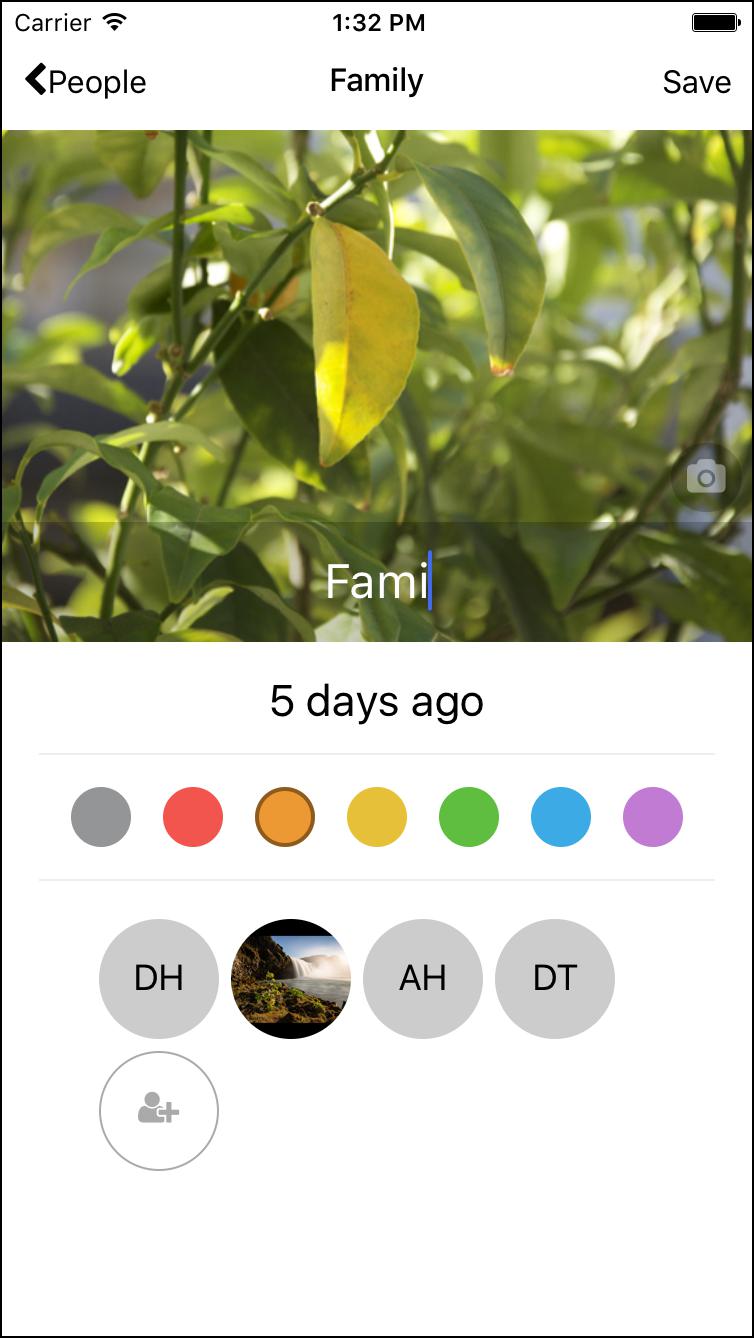
|  |  |
| --- | --- |
| La aplicación Contactos | 40 |



**Grupo de pantallas, Vacío (Detalle)**

Podemos editar el nombre del grupo con bastante facilidad con la edición en el lugar.

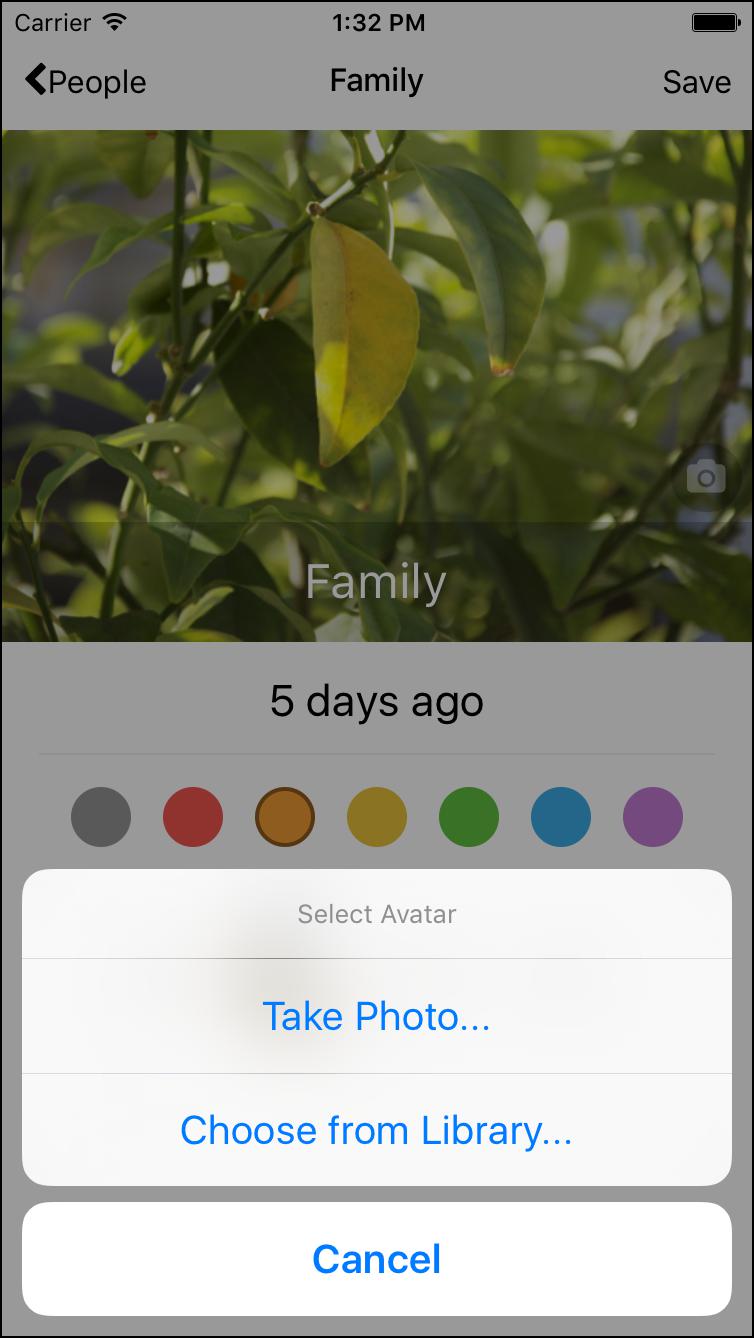
|  |  |
| --- | --- |
| La aplicación Contactos | 41 |



**Nombre Grupo de edición**

Y recoger una imagen con algo que se parece a la norma actionsheet IOS para seleccionar una imagen.

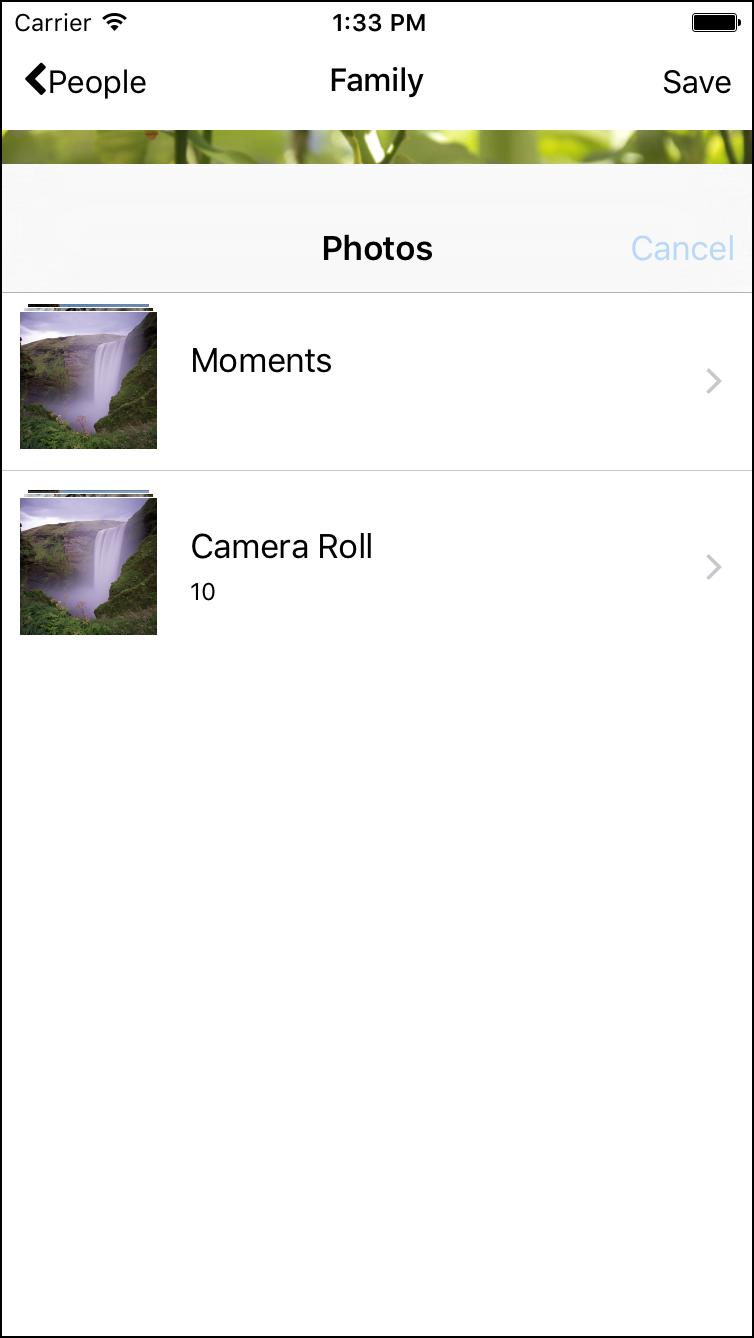
|  |  |
| --- | --- |
| La aplicación Contactos | 42 |



**Escoja imagen**

Entonces, estamos recogiendo una imagen del propio rollo de la cámara de iOS'. No hay necesidad de interfaz de usuario personalizada que se desarrolló aquí.

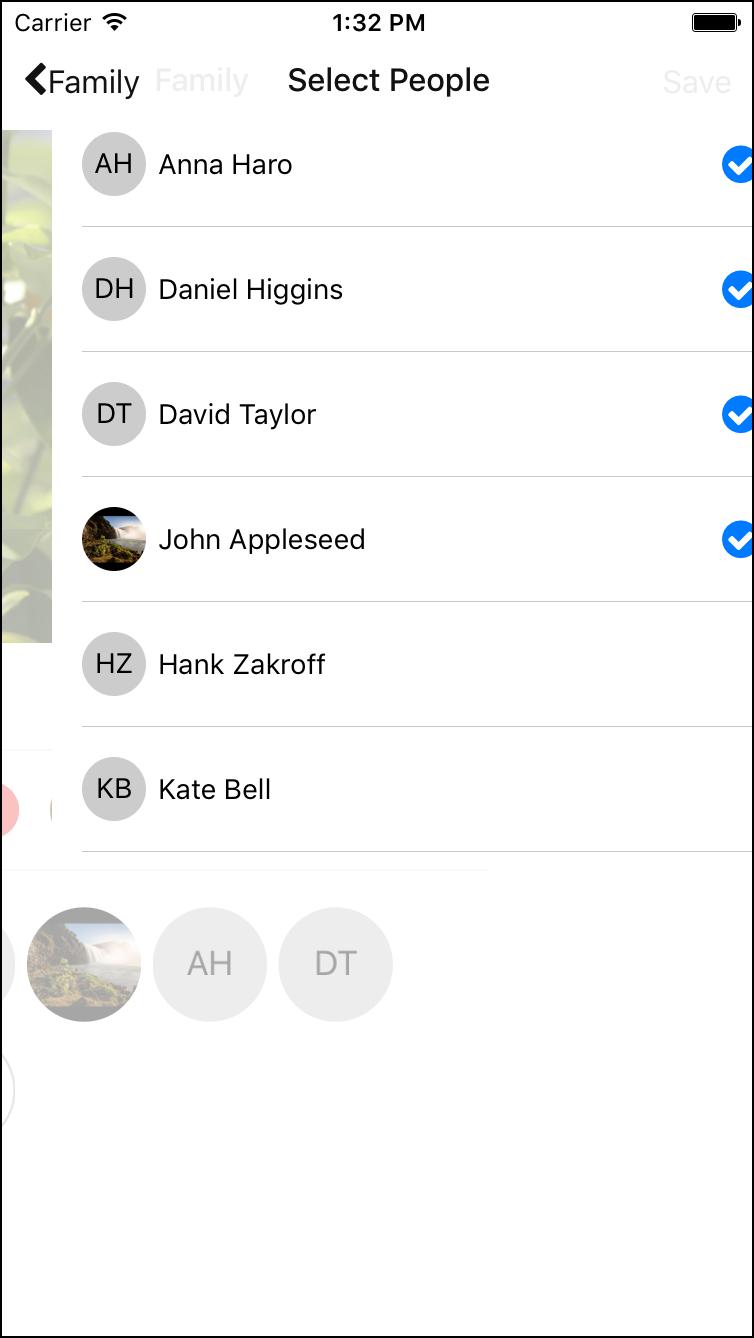
|  |  |
| --- | --- |
| La aplicación Contactos | 43 |



**Seleccionar imagen de la cámara de rollo**

Golpear el icono de persona-más para gestionar los contactos, que la transición a la pantalla de contactos.

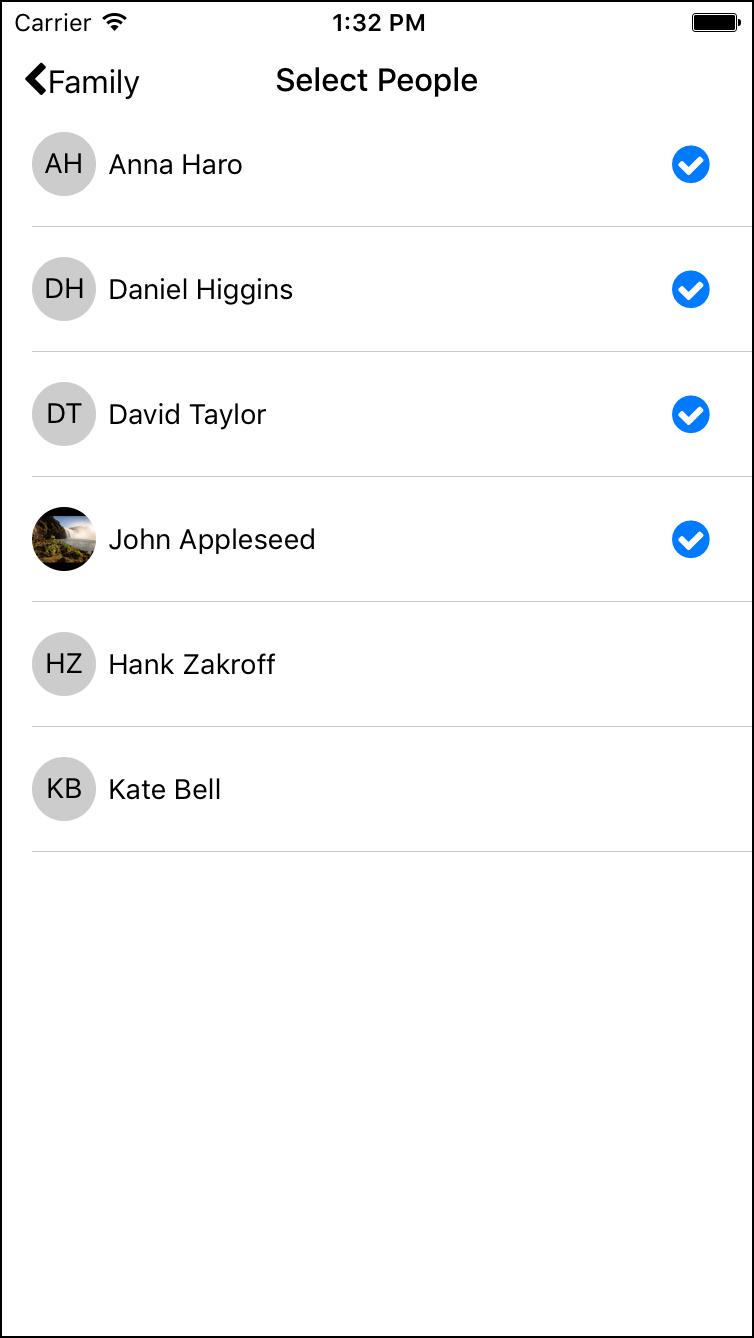
|  |  |
| --- | --- |
| La aplicación Contactos | 44 |



**Transición a Contactos**

Y así es como la pantalla de contactos se parece. Es muy vivo y cada contacto que seleccionar o eliminar hace que todo el punto de vista de reordenación vez se basa en la selección.

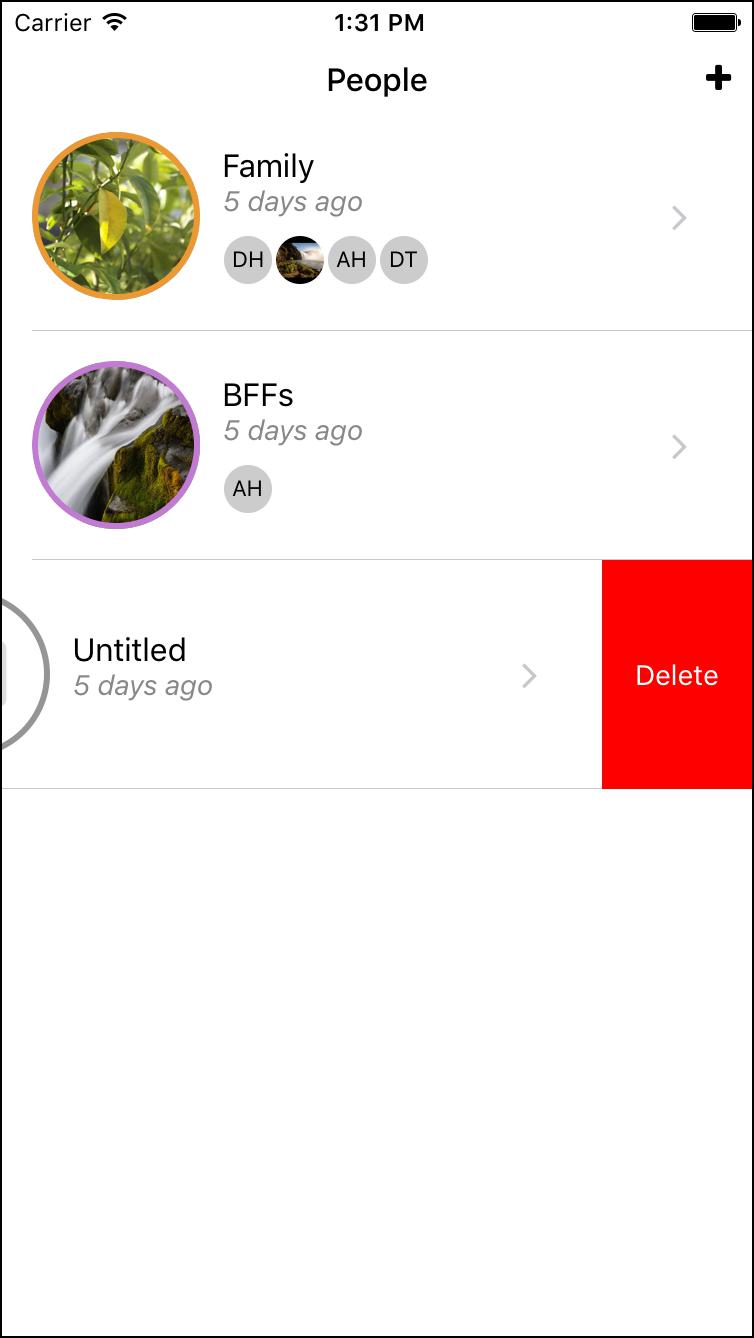
|  |  |
| --- | --- |
| La aplicación Contactos | 45 |



**Nuestra pantalla Contactos**

Volver a nuestra lista de grupos, se muestra fuera de la deslizar para eliminar el comportamiento de IOS, que vamos a llevar a Android también.

|  |  |
| --- | --- |
| La aplicación Contactos | 46 |

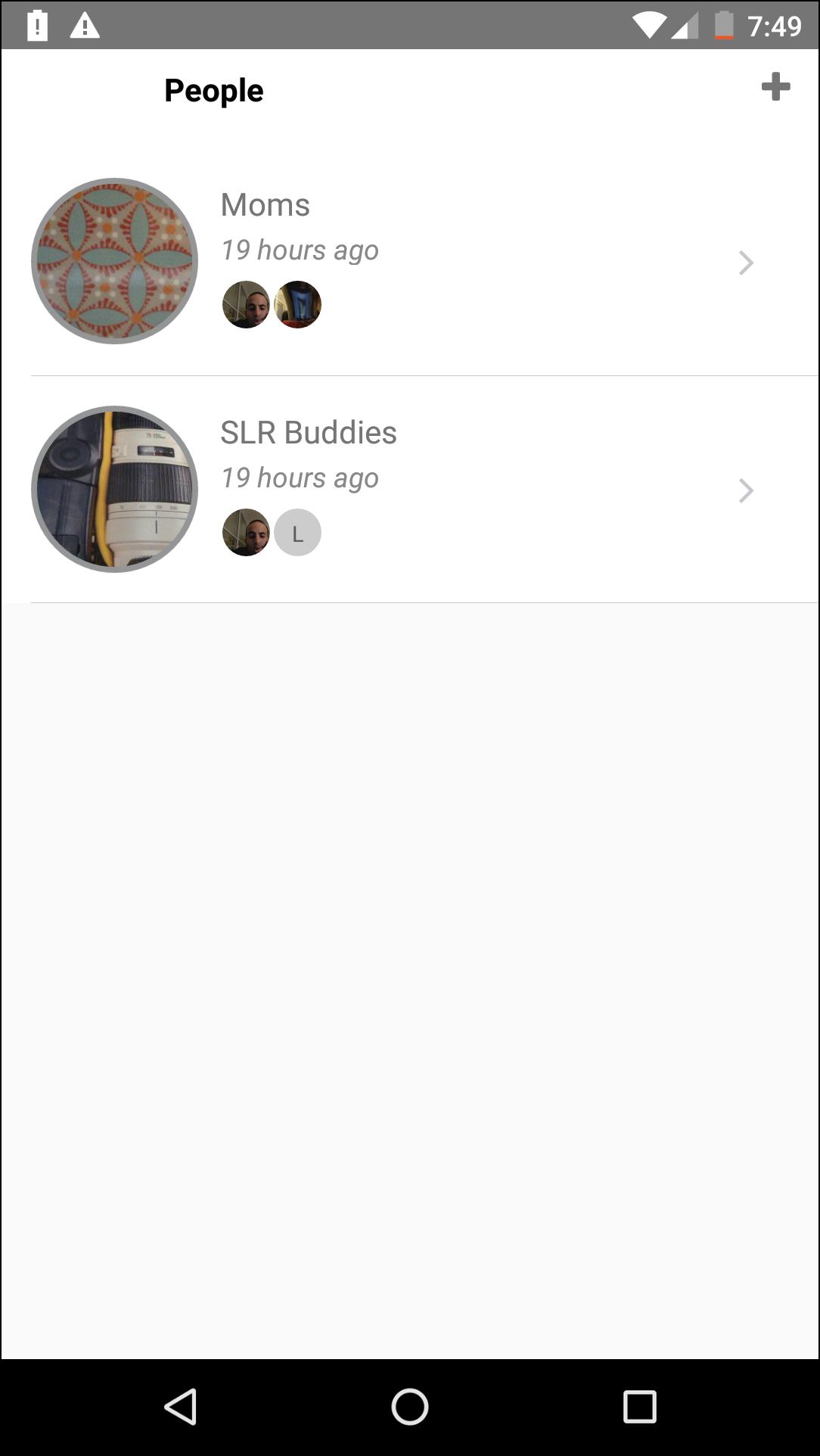


**Deslizar para eliminar**

Justo lo que me puede creer, vamos a repasar las capturas de pantalla Android de la misma base de código. Se hizo prácticamente ningún esfuerzo para hacer que el aspecto de interfaz de usuario y se comportan de la misma en Android, y esto es tan cruda como se pone. Con unas cuantas horas de puesta a punto tanto interfaces de usuario podría tener el mismo aspecto por el píxel, pero, de nuevo, en caso de que el mismo aspecto? Esperar que las discusiones de la diversión con sus gerentes de producto, pero al menos usted tiene el poder de hacer las dos cosas!

Lista de nuestros grupos, en Android.

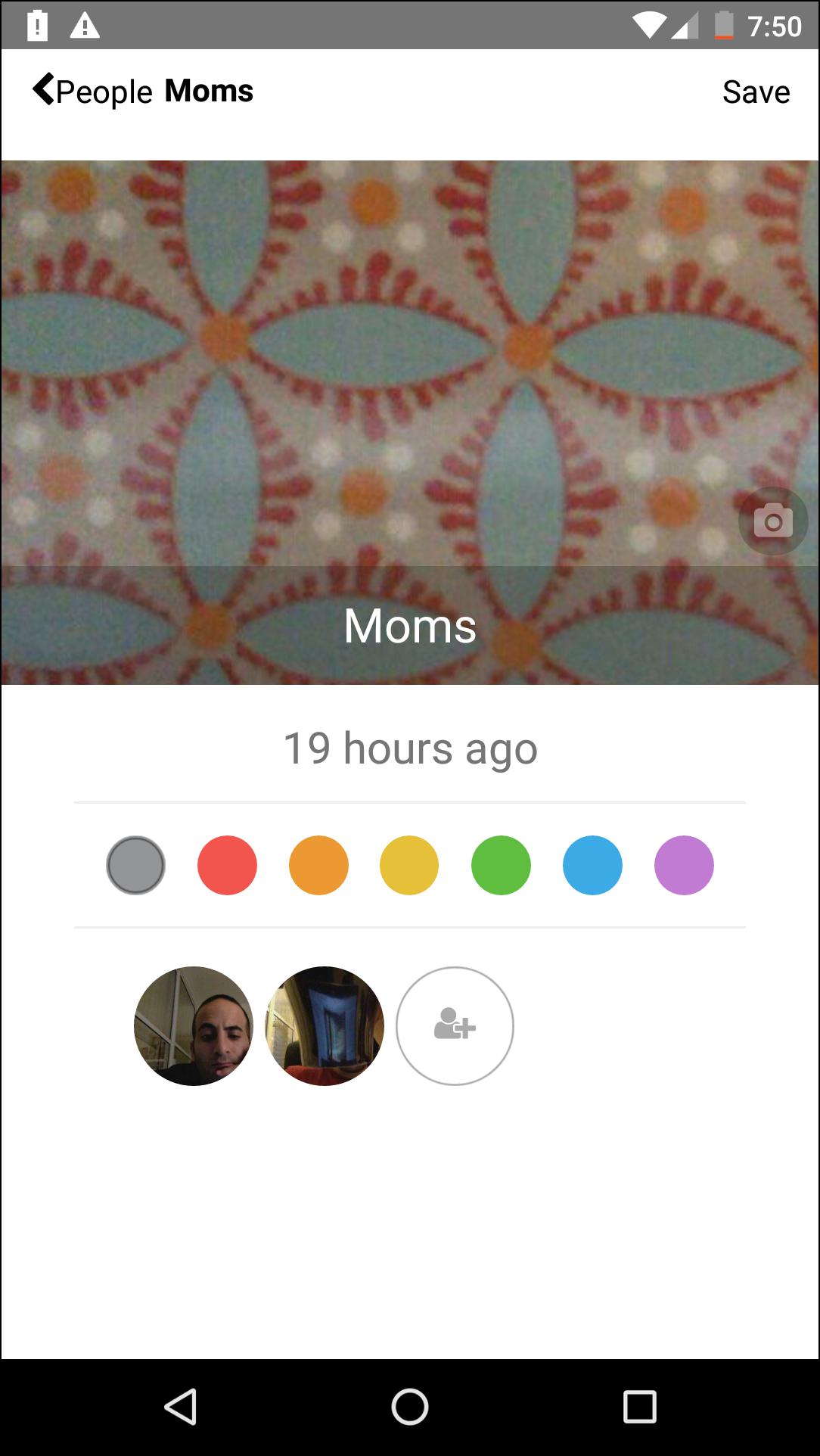
|  |  |
| --- | --- |
| La aplicación Contactos | 47 |



**Android: pantalla Lista de grupos (Master)**

La pantalla de grupo, en Android.

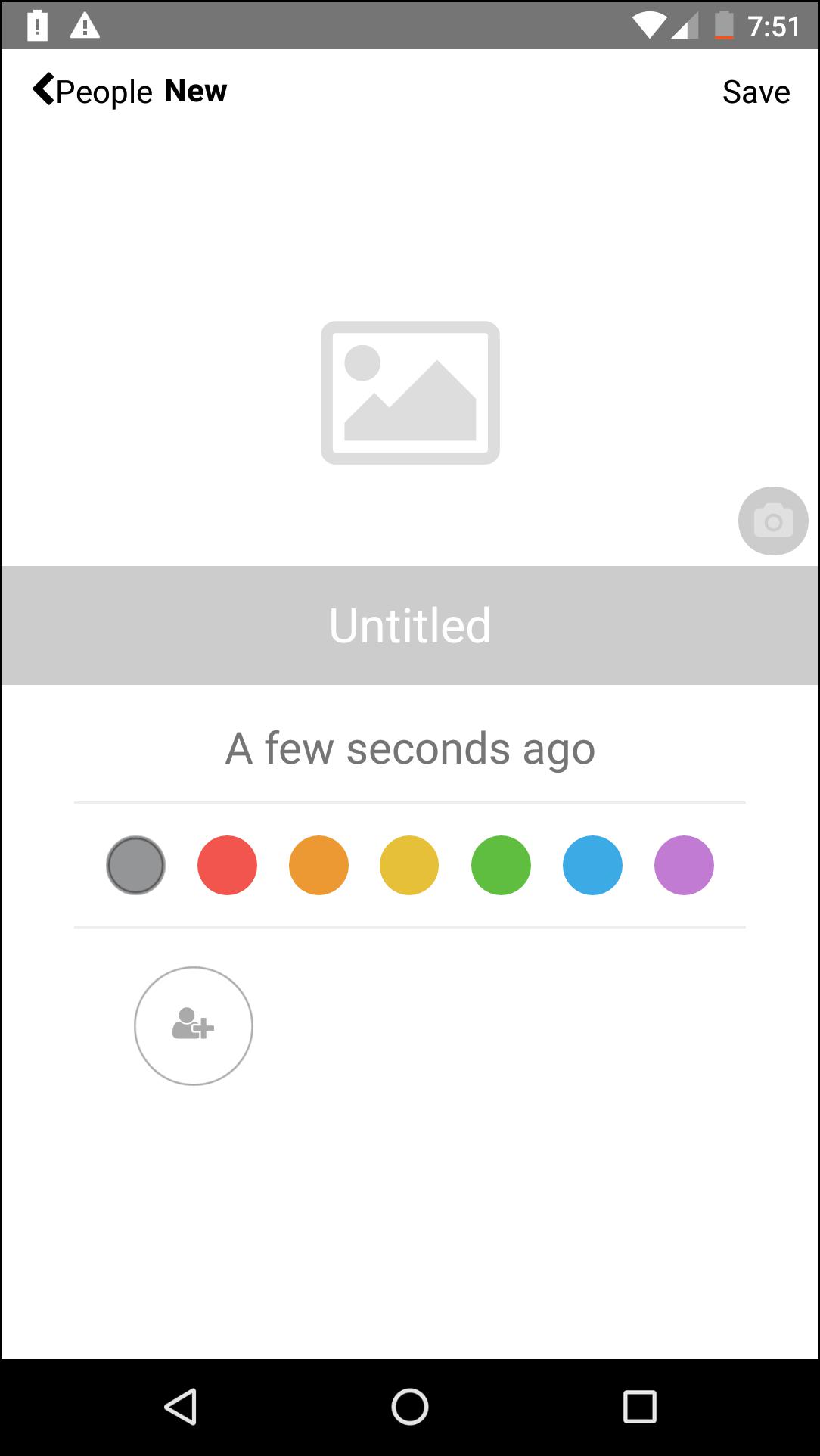
|  |  |
| --- | --- |
| La aplicación Contactos | 48 |



**Grupo de pantallas (detalle): Android**

Y aquí es cómo se ve vacía.

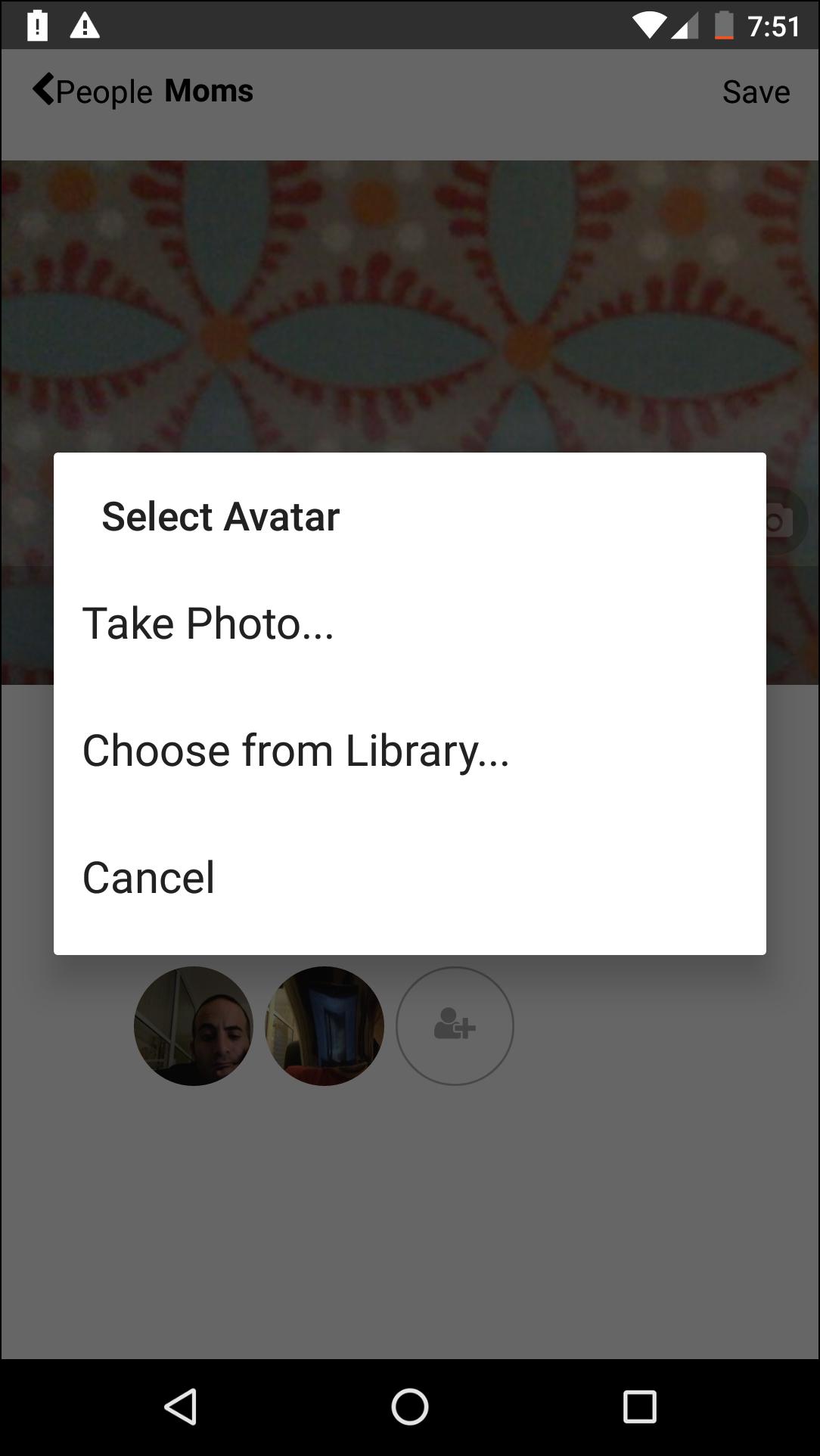
|  |  |
| --- | --- |
| La aplicación Contactos | 49 |



**Android: Grupo de vacío (Detalle)**

Recogiendo una imagen.

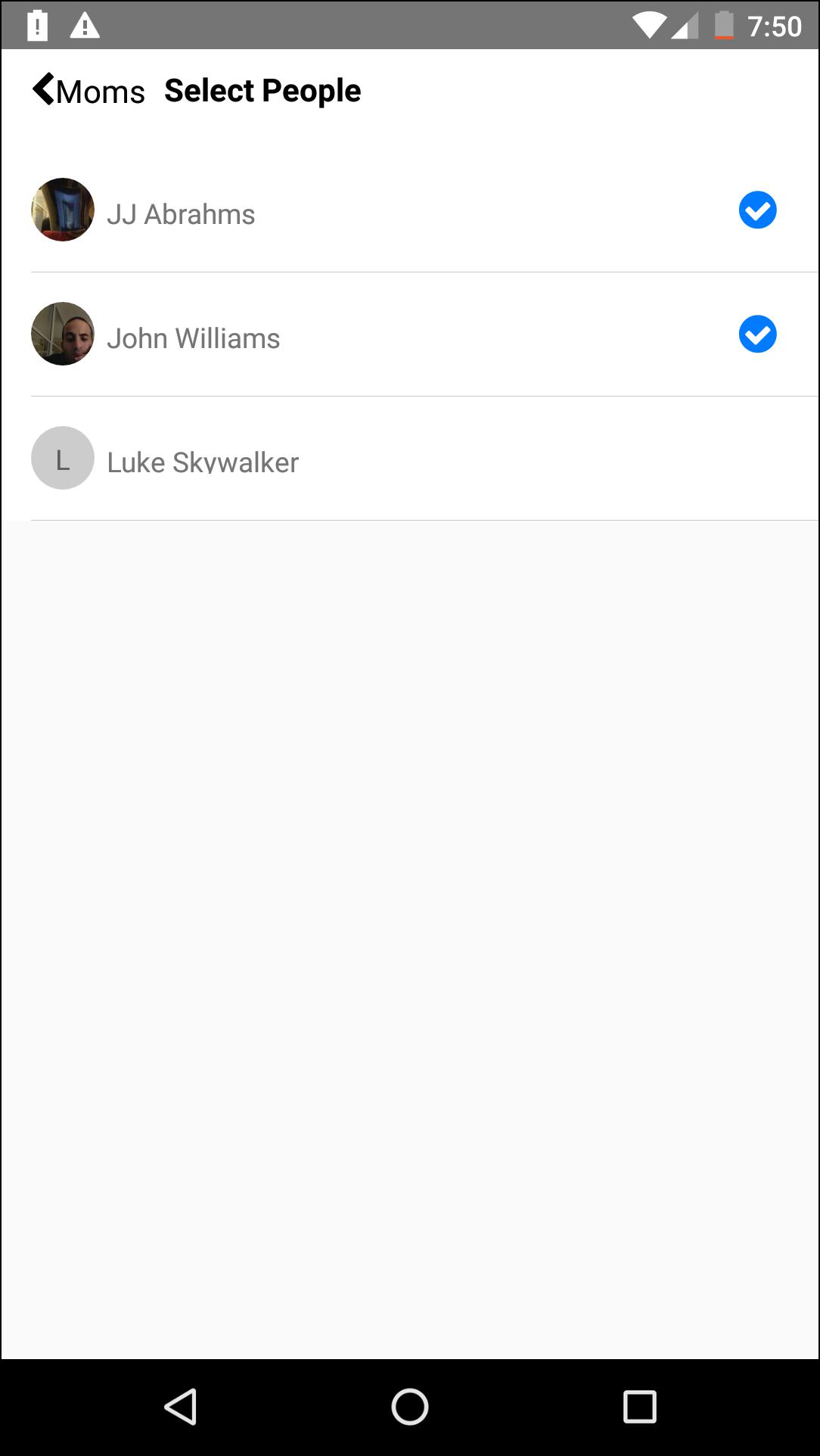
|  |  |
| --- | --- |
| La aplicación Contactos | 50 |



**Hoja selector de imágenes: Android**

Y la pantalla de contactos.

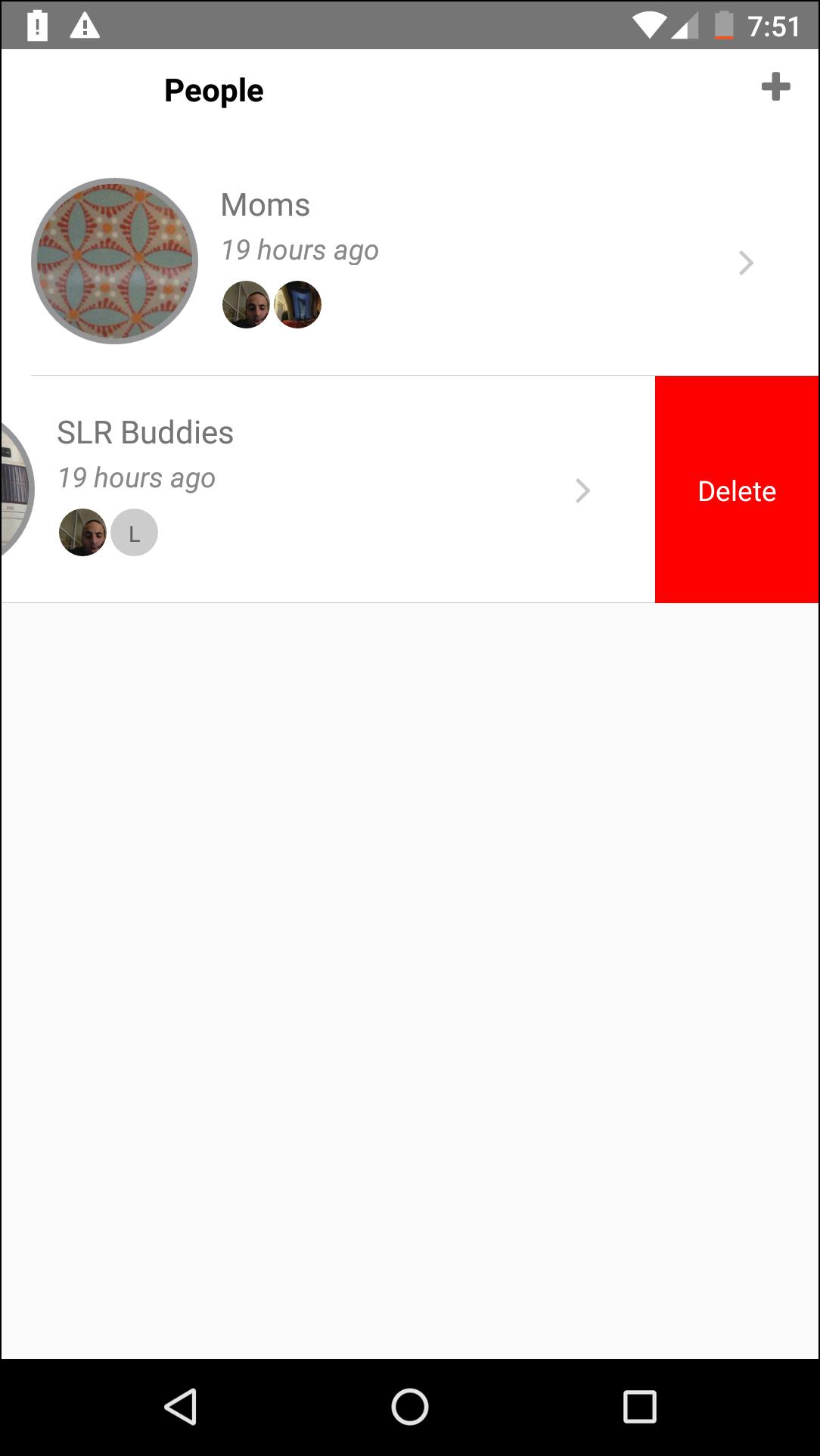
|  |  |
| --- | --- |
| La aplicación Contactos | 51 |



**Android: Contactos de la pantalla**

Deslizar para eliminar en Android.

|  |  |
| --- | --- |
| La aplicación Contactos | 52 |



**Android: Menú Flagelo**

Por lo tanto, todas las pantallas de Android son muy parecidos, se dará cuenta de que la barra de navegación se vuelve un poco diferente, que es muy esperado. Muchas plataformas móviles han elegido para crear diferentes sistemas de navegación y la interfaz de usuario y es tan diferente, tanto en el modo de pensar y de ejecución que vamos a cubrir la navegación como un tema general más tarde. Por ahora, mirando a estas capturas de pantalla se puede decir que es similar y que las diferencias son intencionales.

**Técnico**

En el subtema anterior hemos cubierto cómo la aplicación se parece y puede que tenga sentido un poco cómo se comporta de las transiciones y la redacción. Ahora, vamos a tomar un momento y cubrir los aspectos técnicos de lo que hicimos, o lo que tenemos que hacer con el fin de llegar a ese tipo de aplicación.

En primer lugar, se obtiene Reaccionar nativo como un verdadero off-the-shelf cruz plataforma SDK. Esto significa que tenemos bastante[algunos componentes](https://github.com/facebook/react-native/tree/master/Libraries/Components)[20](#page57) Ya cuando empezamos a usar

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

|  |  |
| --- | --- |
| La aplicación Contactos | 53 |

eso. Esto nos permite componer nuestras pantallas principales, nuestros puntos de vista de lista (en pantalla la lista de grupos, pantalla de contactos), se acumulan los elementos de interfaz de usuario más intensivos, como la selección de variables de color, varios cuadros de imagen de usuario para los contactos y grupos, y así sucesivamente.

A continuación, utilizamos varios componentes de la comunidad contribuyó a hacer que los componentes de interfaz de usuario imple-Menting más fácil:

* [Swipeout](https://www.npmjs.com/package/react-native-swipeout)[21](#page58) - para hacer la experiencia-golpe-a algo en nuestros grupos pantalla de la lista. En nuestro caso sólo deslizar para eliminar
* [listitem](https://www.npmjs.com/package/react-native-listitem)[22](#page58) - para rendir un elemento de la lista IOS-como en términos de interfaz de usuario
* [Los iconos del vector](https://www.npmjs.com/package/react-native-vector-icons)23 - en disponer cada icono del paquete de vectores libres para ser disponible a nosotros fácilmente. Esto incluye fontawesome y más
* [selector de imágenes](https://www.npmjs.com/package/react-native-image-picker)[24](#page58) - para agilizar el actionsheet imagen Cosecha través tanto para iOS y Android, utilizando la interfaz de usuario nativa siempre que sea posible, y esto es todo transparentes para nosotros

Además, utilizamos varios módulos de la comunidad, que nos permiten llegar a cabo en cada plataforma (iOS, Android) servicios del sistema operativo de código nativo Reaccionar Javascript contribuyó:

* [contactos](https://www.npmjs.com/package/react-native-contacts)[25](#page58) - para agarrar todos los contactos del dispositivo de manera estructurada
* [FS](https://www.npmjs.com/package/react-native-fs)26 - para obtener acceso a las operaciones importantes del sistema de archivos, así como metadatos (Documentos ubicación de carpeta para nuestra aplicación).

Internamente estamos haciendo un mini tubería de flujo con una tienda y un emisor de sucesos pero no prestar demasiada atención a eso. Parece que hay mucha discusión sobre qué marco Flux es la mejor, pero es demasiado de una discusión dogmática para poder ser contenida en este libro. En los proyectos de la vida real es posible que desee elegir[alt.js](http://alt.js.org/)[27](#page58) por su enfoque de todo incluido o [Redux](https://github.com/rackt/redux)[28](#page58) porque Es un marco que le permite construir su propio marco.

Tampoco se menciona explícitamente este - se puede utilizar casi todos los marcos React.js Flux con Reaccionar nativo, lo cual es un logro de la comunidad en el camino de fuzzing completamente la línea entre estos dos.

1. <https://www.npmjs.com/package/react-native-swipeout>
2. <https://www.npmjs.com/package/react-native-listitem>
3. <https://www.npmjs.com/package/react-native-vector-icons>
4. <https://www.npmjs.com/package/react-native-image-picker>
5. <https://www.npmjs.com/package/react-native-contacts>
6. <https://www.npmjs.com/package/react-native-fs>
7. <http://alt.js.org/>
8. <https://github.com/rackt/redux>

|  |  |
| --- | --- |
| La aplicación Contactos | 54 |

**Potencial (o: Preparación)**

La aplicación que estamos construyendo es completamente funcional, y no es un confeccionado o aplicación de demostración inútil como una aplicación de tiempo o aún por otra aplicación de lista de tareas. Usted puede tomar esto como un gran punto de partida para cualquier pueblo aplicaciones basadas en: Meetup, charlas, eventos, redes sociales y así sucesivamente. Algunos ejemplos:

* Mediante la adición de un calendario a la vista de grupo, esto se convierte en una aplicación eventos. Añadir una opción de notificación, así y ya está de oro
* Mediante la conversión de la vista de grupo a una lista de chat y conectándolo a un backend reactiva como [firebase](http://firebase.com)29, Que acaba de hacer una aplicación de chat
* Si se mantiene la vista de grupo como está pero que sea más accionables, como hacer un toque en el avatar de un contacto con él o ella llame, o grabar una acción en un servidor que todo el mundo está conectado a, que acaba de hacer una variante de una red social.

Podría ayudar a averiguar si desea realizar cualquiera de estos mientras va sobre el libro o sólo tiene que seguir el libro tal cual. También puede ser que desee para que sea un paso dos, al terminar el libro tal cual y luego la piratería en su variante de la aplicación y tratar el libro como un manual de escritorio de recoger de vez en cuando.

**Resumen**

Espero que tenga una idea de la potencia de React nativo. Nos acabamos de cubrir una aplicación que fue construida para dos plataformas. A diferencia de los cursos típicos de nivel principiante o libros, cubrimos una aplicación del mundo real que se puede convertir en aplicaciones reales con bastante facilidad y servir a un propósito significativo (no hay más aplicaciones tiempo!). Esto es para mostrar que, finalmente, - un nativo, performante, aplicación multiplataforma realmente puede ser construido en el año 2016 - usando Reaccionar nativo.

Lo que no se obtiene de este capítulo hasta ahora, es que la experiencia de construir una aplicación tal fue fácil e incluso divertido. Estoy seguro de que llegaremos a sentir que a medida que avanzamos!

1. <http://firebase.com>

**Tutorial**

En este capítulo, vamos a estar empezando nuestra ruptura de la aplicación Contactos. Tendrá la oportunidad de entender la fórmula que la mayoría de las aplicaciones están adoptando, las decisiones técnicas, los porqués y los cómos de la construcción de un mundo real Reaccionar aplicación nativa. entonces vamos a diseccionar cada pantalla y entender cómo reaccionar y Reaccionar juego nativo juntos, así como supuestamente problemas distintos del desarrollo, tales como el estilo y la gestión de recursos.

**Estilo walkthough**

Vamos a establecer algunas expectativas y objetivos de la próxima tutorial. En su forma actual en el momento de la escritura, hay un montón de maneras de ir cuando usted desea aprender cómo construir una aplicación. Usted puede escoger un Udemy o curso cabeza de huevo o un tutorial en línea, o seguir un folleto que te lleva paso a paso, línea por línea por un proyecto vacío a una aplicación totalmente funcional.

Este libro tiene un enfoque más difícil, específicamente nos estamos tomando el enfoque de la disección - usted ha visto una aplicación totalmente funcional, y vamos a desmontarlo, entender los porqués y los cómo y extraer el conocimiento destilada que necesita con el fin para hacer su propia aplicación; eso es porque sé que va a seguir leyendo, y seguir aprendiendo a través de recursos como Udemy y Cerebrito largo de su carrera - que nunca se deja de aprender qué?

Este enfoque también se ocupa de la volatilidad del mundo Javascript, el mundo reaccionar y el proyecto React nativo (que todavía es nuevo). En otras palabras, si escribía un libro de tutorial, es probable que dentro de los 6 meses usted perdería todo valor por su dinero, ya que para entonces ya sea cambiada Javascript (ES6, ES7), Reaccionar cambiado, o reaccionar nativo cambió. En lugar de ello, he elegido a centrarse en los principios y las pautas básicas que llevarán a cabo a través de la prueba del tiempo, y le mostrará formas antiguas y nuevas maneras de hacer las cosas (Javascript, ES6, etc.) en caso de que toparse con edad o una nueva Reaccionar código nativo en la web. Esta es una guía pragmática, una guía de conocimientos destilada.

Con eso fuera del camino, vamos a empezar!

**Las opciones técnicas**

Como recordatorio, la aplicación Contactos es una aplicación sobre sus personas. Se trata de una aplicación de gestión de contactos en los esteroides que se pueden tomar y construir sus propias aplicaciones sobre. Actualmente se mantiene una lista de grupos de personas, sus contactos y varios

55

|  |  |
| --- | --- |
| Tutorial | 56 |

características caramelo del ojo para promover fácil de búsqueda y personalización. Si lo desea, puede saltarse el capítulo de la aplicación Contactos para obtener una visión general en toda regla de la aplicación, así como sus bloques de productos y la construcción de infraestructura.

Por ahora el mundo es conocido por el Javascript [su churn](http://www.breck-mckye.com/blog/2014/12/the-state-of-javascript-in-2015/)[30](#page61), Y debido a eso, siento que los primeros artículos que necesitamos para conseguir de la tabla son las elecciones que hacemos a la base de código que vamos a diseccionar, así que vamos a hacerlo ahora.

**javascript**

Por nuestra base de código Javascript, utilizamos una mezcla de ambos ES5 (probablemente el Javascript todo el mundo está familiarizado con), y ES6 (la sexta edición, introducción de muchos estado del arte de construir en la lengua) de manera intercambiable. Esto es para apoyar el viejo y el nuevo. Muchas veces yo estaba ya sea buscando en un viejo Reaccionar código de estilo que tenía que soportar, pero parecía extraño para mí porque se me ocurrió ya olvidar la vieja manera de hacer las cosas, y luego otra vez miré a un nuevo Reaccionar código y la misma cosa sucedido porque todavía no estaba familiarizado con otra nueva forma de hacer las cosas que se introdujo en los apenas dos semanas que estaba haciendo back-end o código móvil nativa.

Muchos dicen que 2014-15 fueron los años en que Javascript se convirtió en una jungla; Me gustaría que le muestre las dos formas de hacer las cosas en Reaccionar para que pueda orientarse bien cuando uno siente que está varado en esa selva.

**Reaccionar**

La buena noticia con Reaccionar nativo es que se obtiene un cuadro bastante cerrada-React. En mi opinión, teniendo todo el peso de Javascript y reaccionar elecciones y decisiones, además de desarrollo móvil es demasiado difícil de soportar, que si se tratara de esa manera sería un importante desvío.

Con Reaccionar nativo a obtener los componentes básicos que usted puede construir un mundo entero con y que son bastante al horno para usted, y usted tiene un estilo que se realiza en línea. No hay hora de tocar el violín con webpack (esa es la realidad en el momento de la escritura) para hacer un complejo de tuberías de construcción que extrae CSS en sus propios archivos, se fusiona sprites y así sucesivamente. Más encima, se obtiene un demonio que vigila sus archivos y paquetes todo perfectamente ya - así que eso es también algo que no es necesario configurar. Además hay una estrecha integración con herramientas como[Fluir](http://flowtype.org/)[31](#page61) así que aunque Reaccionar nativo como un proyecto de software ve React - el marco - como aguas arriba, que la disposición entera se siente muy moderna y actualizada.

**Flujo**

La elección y discutiendo sobre qué marco Flux es mejor está de moda en estos días. Para mantener nuestro enfoque vamos a definir Flux como una forma estructurada de hacer

1. <http://www.breck-mckye.com/blog/2014/12/the-state-of-javascript-in-2015/>
2. <http://flowtype.org/>

|  |  |
| --- | --- |
| Tutorial | 57 |

arquitectura de la aplicación basada en eventos, y en mis ojos es una evolución de la gestión de eventos basado en mensajes y estoy totalmente de acuerdo [con este artículo](https://bitquabit.com/post/the-more-things-change/)[32](#page62).

Para mantener nuestro enfoque, que he elegido para construir una versión delgada de una tubería de flujo con un simple emisor de sucesos y una tienda. Si desea utilizar un marco, se puede extraer[alt.js](http://alt.js.org/)[33](#page62) o [Redux](https://github.com/rackt/redux)[34](#page62). Entonces, ¿cómo elegir para sus propios proyectos futuros? Depende. Tengo la opinión de que si se trata de una pequeña aplicación suficiente, sólo tiene que ir con lo que cada vez le ahorra más trabajo (alt.js), y si es una aplicación que te gustaría mantener durante mucho tiempo y que consulte a un considerable gran hoja de ruta para, elegir algo sencillo y pequeño que se puede construir sobre más tarde (Redux).

**Estructura de carpetas**

Vamos a utilizar la siguiente estructura de carpetas para nuestro proyecto. En mi opinión, la estructura de carpetas de un proyecto debe tener la forma principalmente por el equipo que trabaja con él, y sólo entonces, se adapta a la tecnología de pila.

* puntos de vista/
* ayudantes /
* dominar/
* index.js
* styles.js
* other.js

7 componentes /

8 servicios/

9 [cualquiera de las carpetas orientado a flujo, tales como acciones, tiendas, etc.]

En nuestra aplicación, he optado por centrarse en React nativo y no Flux, y es por eso que no verá la mega estructura típica de las carpetas que se acciona mediante un proyecto Flux. En nuestro proyecto que estamos sosteniendo en una tiendaservicios/ y hacer acciones simples objetos, y que se traduce en un solo archivo y un marco como alt.jspara ajustarse a Flux. Yay por simplicidad!

Para el puntos de vista/ parte, estamos llevando a cabo el componente principal en una index.js archivo de modo que cuando se requiere vistas / master se consigue tirado de forma automática, y luego en nuestra index.js presentar cuando simplemente requerimos ./stylesque conseguirá fue a buscar a nivel local de la carpeta de ese punto de vista. En otras palabras, estamos manteniendo los estilos bien ocultas a lo largo del componente o vista que lo utiliza (hablaremos de estilos y por qué son en realidad presentemos una Javascript tarde).

Otras maneras de hacer esto? Se podría ocupar un lugar central para estilos, en un solo archivo. Se podría sustituir esta estructura de carpetas con uno procedente de la vainilla Flux, o puede utilizar una estructura que está llevando a React.js si eres un desarrollador React.js ya - que sería una gran ventaja.

1. <https://bitquabit.com/post/the-more-things-change/>
2. <http://alt.js.org/>
3. <https://github.com/rackt/redux>

|  |  |
| --- | --- |
| Tutorial | 58 |

**Pruebas**

La combinación de reaccionar y móvil para la prueba es increíble. En mi opinión Reaccionar es casi comprobable por diseño. Su naturaleza funcional permite que su prueba sea limpia y libre de configuraciones de efectos secundarios, y la forma de grano fino de la construcción de cosas en componentes es compatible con la unidad de pruebas muy bien. Que cubre todo el código Java o Objective-C que puede o no puede haber escrito a lo largo de los años, si eres un desarrollador móvil, y que llegó a ser comprobable, ya que se sometió a un proceso de putrefacción, y obtuve etiquetados como “no comprobable , vamos a llegar a algún día”.

Más adelante, hay pruebas de integración, o la automatización de pruebas. El mundo móvil está prosperando con herramientas de automatización de pruebas, incluso más que lo que la web exhibe en sus épocas de oro de la prueba de automatización con herramientas como el selenio. Apple hizo su propio marco de pruebas de interfaz de usuario en la empresa, Google lanzó su propio bien, y ambos son bastante buenos, y herramientas como[Appium](http://appium.io)[35](#page63) ese el objetivo de proporcionar un único código base de pruebas para ambas plataformas están llegando a un alto nivel de calidad - y estamos aquí para disfrutar de todo.

Hay tanto que decir que no es simplemente demasiado como para cubrir este tema y realmente debe hacerlo en un libro de su propia. Es por esto que en este libro que he elegido no tomar partido, y por desgracia no a estar cubriendo eso.

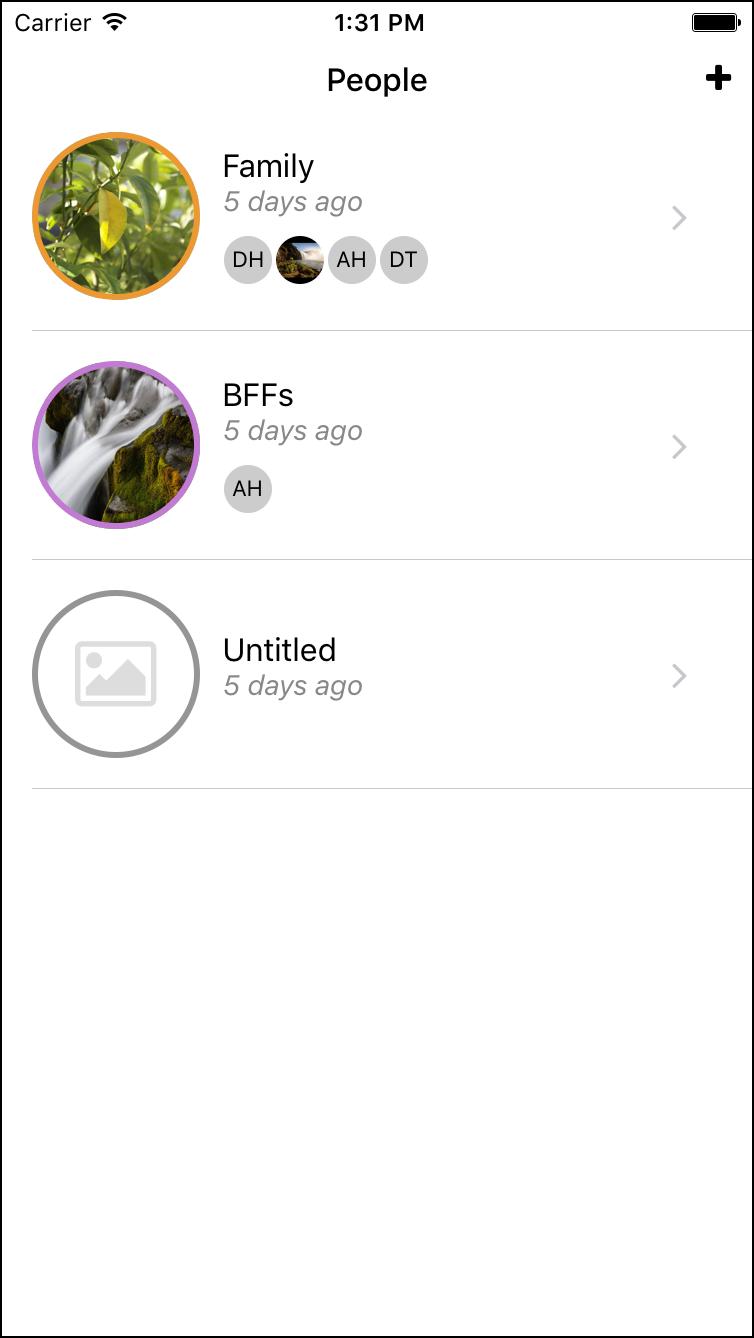
**La disección de nuestra primera pantalla**

Vamos a tomar nuestra primera cabeza de pantalla activada, diseccionar y cubrir todo el material necesario para construir una cosa así.

Así es como se ve:

1. <http://appium.io>

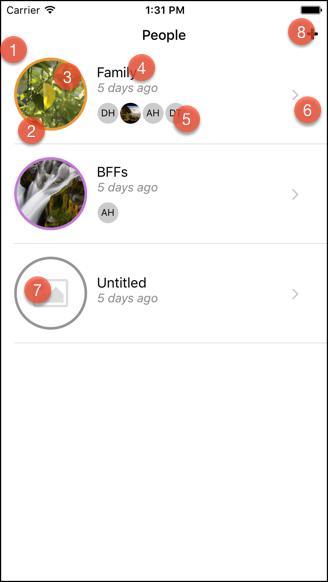
|  |  |
| --- | --- |
| Tutorial | 59 |



**Pantalla principal de la aplicación**

Antes de continuar, vamos a ver lo que estamos tratando con:

|  |  |
| --- | --- |
| Tutorial | 60 |



**Pantalla principal de la aplicación anotado**

1. los [Vista de la lista](https://facebook.github.io/react-native/docs/listview.html)[36](#page65) Reaccionar componente nativo viene con. Vamos a cubrir esto más a fondo en breve.
2. Nuestra celda de lista. Cada gran aplicación de vista de lista le permite hacer su propia plantilla de células (más sobre esto más adelante) y el Reaccionar un nativo no es diferente, ya que utiliza las implementaciones de iOS y Android que ya hacen esto. Una vez que se hace clic, la célula nos llevará a la vista de detalle y se poblarlo basado en el elemento seleccionado.
3. El avatar grupo. La imagen se tomó desde el rollo de la cámara o el almacenamiento de imágenes en el teléfono del usuario.
4. El título del grupo, y la marca de tiempo. Ambos son recogidos a partir de la estructura JSON representa un grupo directamente, y usamosmoment.js para conseguir “hace tiempo” un ser humano legible rinde por el momento.
5. La lista de contactos. Estos se representan en directo por grupo, a partir de los contactos establecidos dentro de cada grupo. Hacemos uso de la misma repetimos
6. <https://facebook.github.io/react-native/docs/listview.html>

|  |  |
| --- | --- |
| Tutorial | 61 |

componente para contactos avatares, así como avatar grupo aquí y en otros lugares. Utilizamos Flex para asegurarse de que todo expone correctamente.

1. Este diseño elemento Chevron y en general es una pista de diseño por defecto de iOS' de elementos de la lista. Recogemos el símbolo Chevron y otros símbolos a través de salida de la aplicación del componente comunitario por vectores iconos (más sobre esto más adelante), por lo que en realidad es un simple<Icono> elemento de aplicación.
2. La interfaz de usuario sabe cómo representar marcadores de posición cuando no existe ninguna imagen. Utilizamos Reaccionamos de capacidad para calcular el estado y escupir un árbol interfaz de usuario diferente cuando los datos se ve diferente.
3. Este botón de navegación va a crear un nuevo grupo y nos llevará a la pantalla de detalles en un estado de enrutamiento “Nuevo”. Vamos a cubrir enrutamiento también.

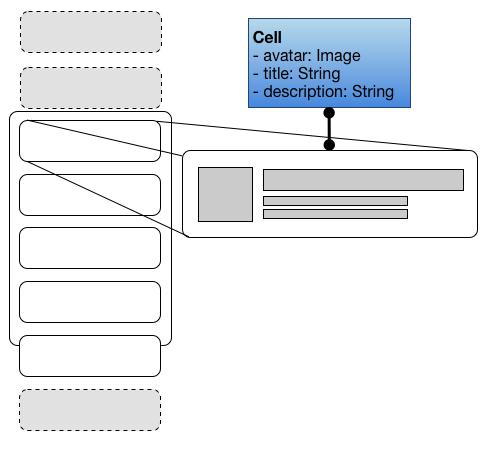
Antes de continuar, debemos cubrir la vista de lista, el patrón y lo que significa en el móvil, y cómo se aplica en React nativo.

**Lista móviles Vistas**

La vista de lista es el caballo de batalla de la aplicación móvil. Independientemente de React nativo, si se toma un momento para mirar realmente a las aplicaciones en su teléfono, usted encontrará que muchos de ellos son simplemente pimped-up vistas de lista (en IOS es elUITableView y en Android el RecyclerView): Gmail es simplemente la gestión de listas (bandeja de entrada, basura, filtros personalizados), Twitter es la gestión de una lista interminable, e incluso la configuración es realmente una estática UITableView en IOS.

Cada iOS o Android libro sobre el desarrollo móvil cubre su propia lista Ver variante y hoy, cuando ambas plataformas maduraron - existen los mismos principios. Me las he arreglado para destilar estos principios ya que son muy importante saber ahora que está al frente de ambas plataformas, como veremos en la siguiente imagen y discusión.

|  |  |
| --- | --- |
| Tutorial | 62 |



**Mecánica de una vista de lista**

Por lo que una vista de lista es un contenedor de artículos, a menudo acompañado con una funcionalidad envoltura de desplazamiento. Es responsable de los siguientes:

* Disposición - haciendo que cada elemento de procesamiento, medición y ajuste con el resto en un formato de página seleccionado (lista, rejilla o lo que sea)
* Rendering - cada célula es un componente independiente, que es una cosa muy buena. Usted puede construir cada celda por su propio derecho, como si nunca tiene que ir en una vista de lista con el tiempo, y luego enchufarlo como mejor le parezca. Ayuda a la experiencia de desarrollo, y la capacidad de prueba del composability. En Reaccionar nativo tal célula es simplemente un punto de vista nativo Reaccionar - nada nuevo que aprender! Si viene desde una plataforma nativa entonces esto puede sonar una campana - en IOS se puede utilizar la celda por defecto que viene conUITableView, O su propio diseño xib componente que se quiere en plataforma tu UITableViewluego. En Android diseñaría un nuevo archivo de diseño de XML para la célula e inflar que por célula.

|  |  |
| --- | --- |
| Tutorial | 63 |

* Reciclaje - en iOS y recientemente (depende de cómo se defina “reciente”) en Android, vistas de lista sólo se están llevando a cabo una cantidad fija de células “vivas” proporcional al área visible de la vista de lista en la pantalla, y que va a engañar en el pensamiento existe una mayor cantidad de células mediante el reciclaje de las células fuera de la pantalla y tirando de ellos de nuevo en circulación mientras se desplaza. Es por esto que en el dibujo algunas células son de color blanco y algunos son de color gris.
* Ciclo de vida y eventos - se repartirán eventos para cada célula en turno. Con la forma nativa Reaccionar configurar eventos como lo haces con ellos para cualquier Reaccionar componente nativo, y los eventos del ciclo de vida sigue siendo la misma que con cualquier componente

**-** de nuevo, nada nuevo que aprender!

* Datos y adaptadores - ya que cada vista de la lista es el responsable de las preocupaciones que acabamos de mencionar, es crucial que se sabe cómo los datos se parece - el número de elementos, su contenido para diferir el render por célula, y cuando los cambios de contenido. Tal punto de vista lista no tiene por qué importa si los datos proviene de un servicio Web o la base de datos local. Aquí es donde el concepto de adaptadores entran en juego; tanto como adaptadores de Android, la vista de lista Reaccionar nativo tiene su propio.

**La pantalla de Grupos**

Ahora podemos continuar mirando a la pantalla principal de los grupos. Vamos a volcar el código aquí y analizar el caso poco a poco.

* 'Use strict';

2

3 **var** Reaccionar = exigir('Reaccionar nativo')

4 **var** {

* Vista de la lista,

6 } = Reaccionar

7

8 **var** Celda = exigir('./celda')

9

1. **var** suscribibles=exigir('Suscribible')
2. **var** R=React.createClass

12

1. **var** Dominar=R ({
2. mixins: [Subscribable.Mixin],
3. almacenar: **función**() {
4. **devolver este**.props.store
5. },
6. getInitialState: **función**() {

|  |  |
| --- | --- |
| Tutorial | 64 |

1. **var** ds= **nuevo** ListView.DataSource ({rowHasChanged:
2. (R1, r2) => {

|  |  |
| --- | --- |
| 23 | *// piratear, necesitará inmutabilidad en la tienda para que esto se detecta* |
| 24 | **return true** |

1. }
2. })
3. **var** lista= **esta**.lista de la tienda()
4. **regreso** {
5. fuente de datos: ds.cloneWithRows (lista),
6. }
7. },

32

1. componentDidMount: **función**() {
2. **esta**.addListenerOn (**esta**.store (). eventos,'cambio', **esta**.onStoreChanged)
3. },

36

1. onStoreChanged: **función**() {
2. **var** lista= **esta**.lista de la tienda()
3. console.log ("Maestro: nueva lista, elementos:", List.length)
4. **esta**.setState ({dataSource: **esta**(.state.dataSource.cloneWithRows
5. lista
6. )})
7. },

44

1. didSelectRow:**función**(fila){
2. **esta**.props.navigator.push ({
3. carné de identidad: 'detalle',
4. título: row.title,
5. accesorios: { ít: fila,

|  |  |
| --- | --- |
| 50 | almacenar: **esta**.almacenar(), |
| 51 | navEvents: **esta**.props.navEvents}, |

1. })
2. console.log ("MASTER: seleccionado", Row.title)
3. },

55

1. didDeleteRow:**función**(fila){
2. **esta**.store (). eliminar (fila)
3. },

59

1. renderRow:**función**(fila){
2. **regreso** (
3. <clave de la célula={} Row.id

|  |  |
| --- | --- |
| 63 | ít={fila} |
| 64 | onDelete={()=>**esta**.didDeleteRow (fila)} |
| sesenta y cinco | onPress={()=> **esta**.didSelectRow (fila)} |
| 66 | /> |

|  |  |
| --- | --- |
| Tutorial | sesenta y cinco |

1. )
2. },
3. hacer: **función**() {
4. **regreso** (
5. <estilo ListView={{acolchado superior: 50, flexionar: 1}}

|  |  |
| --- | --- |
| 73 | fuente de datos={**esta**.state.dataSource} |
| 74 | renderRow={**esta**.renderRow} |
| 75 | /> |

1. )
2. }
3. })
4. module.exports = Dominar

Eso es un montón de código, pero sorprendentemente no demasiado. Es sólo la lógica y no mucho código de interfaz de usuario, o así llamada la “[Componente inteligente](https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.mh91y38yq)[37](#page70)”, Y me gusta llamarlo‘View Controller’, por lo que ya personas familiarizadas con MVC y otros marcos tendrán algo que conectarse. Como tal punto de vista, nos aseguramos de que la propiedad real de los datos se lleva a cabo aquí (en contraposición al interior de las células):

* Navegando a un elemento
* Supresión de un elemento
* sincronización de datos y notificaciones

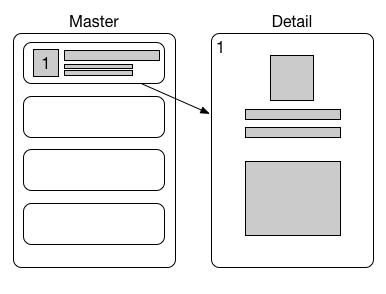
También he llamado a esta “Maestro” intencionadamente y no “GroupsView”, y yo he llamado la celda grupo acaba de “Cell” y no “GroupCell” o “GroupItem”. Esto es para impulsar la idea de que no estamos haciendo nada especial aquí, este es el mismo patrón de nuevo - Maestro-Detalle. Deje de lado la pista a ese momento, y volver al listado de código después.

**Maestro-Detalle**

Este patrón existido desde el [albores de las edades](https://en.wikipedia.org/wiki/Master%E2%80%93detail_interface)[38](#page70), Para mainframes, aplicaciones de escritorio y aplicaciones móviles ahora. Aquí es un dibujo abstracto de lo que significa.

1. <https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.mh91y38yq>
2. <https://en.wikipedia.org/wiki/Master%E2%80%93detail_interface>

|  |  |
| --- | --- |
| Tutorial | 66 |

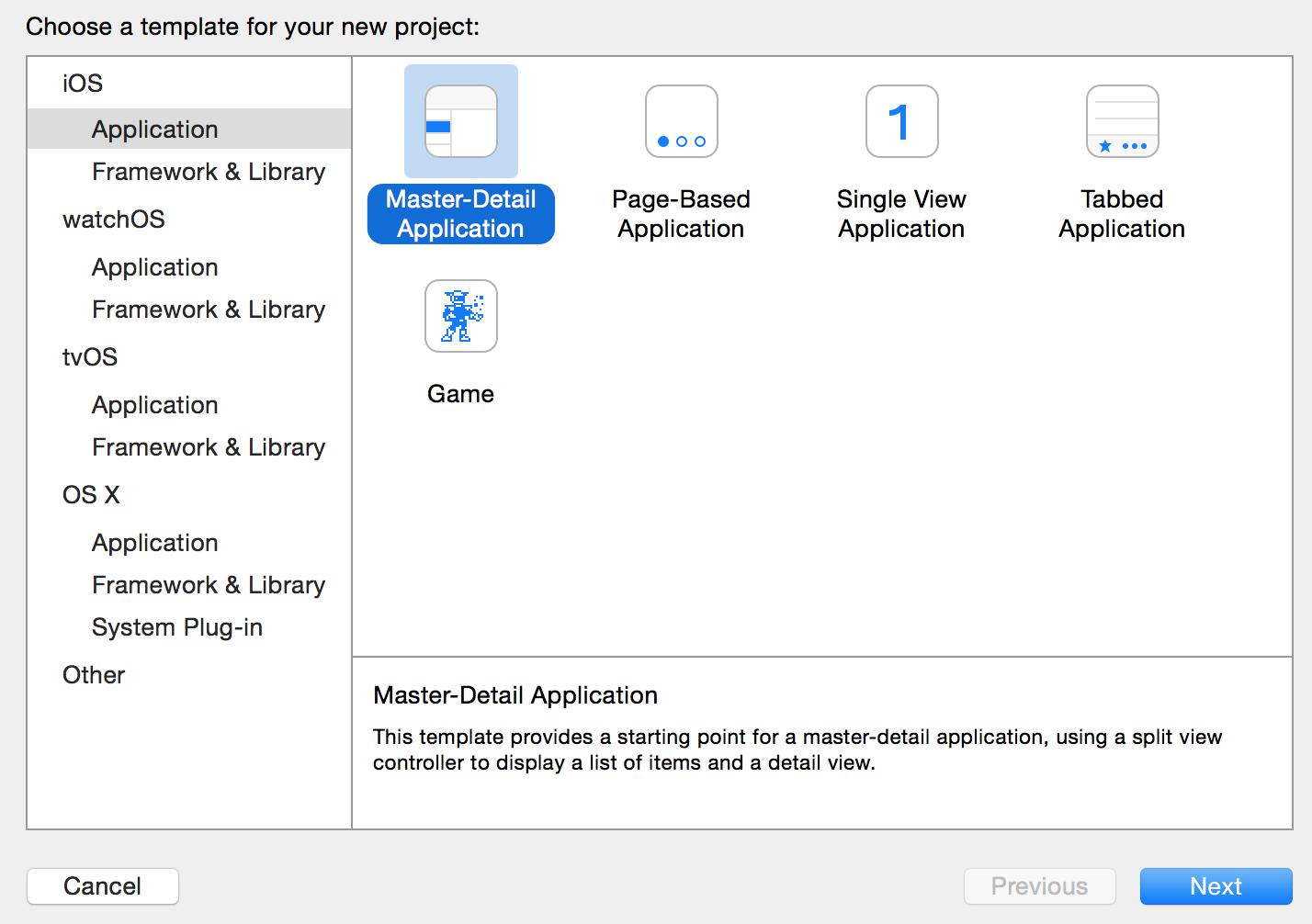


**Típica Maestro-Detalle**

Básicamente siempre debemos tener una lista de cosas - la vista Patrón. Y en React Nativo este será nuestroVista de la listacomponente. Entonces, para cada elemento, una vez seleccionado, prestamos una vista detallada. Para nuestra aplicación, y en React nativo, esto será una visión completa que pasará a, entregándoselo el elemento de datos de la vista de lista en sus apoyos, a través de nuestra pila de enrutamiento (más sobre esto más adelante).

En iOS, puede generar un maestro detalle de aplicaciones basadas en vivo con bastante rapidez:

|  |  |
| --- | --- |
| Tutorial | 67 |



**Plantilla Maestro-Detalle con Xcode**

Seguir adelante y probarlo, a continuación, ejecute la aplicación que se generó a jugar con él. Es esencial que desarrolle la intuición “nativo” para saber cómo se hacen las cosas más en cada plataforma a pesar de que está centrado en hacer una aplicación multiplataforma.

Para envolverlo, en nuestra aplicación estamos demostrando estos conceptos:

* Maestro - la pantalla de grupos, por lo maestro / index.js
* Cell - el componente de la célula maestra, en maestro / cell.js
* Detalle - la pantalla de edición de grupo, por lo detalle / index.js

Para el resto de la base de código que vamos a hacer de distancia de esta manera patrón-ish de nombrar las cosas y los componentes de llamadas en su nombre semántica. Ahora estamos listos para continuar mirando la pantalla de grupos, o en la vista principal.

**ListView y Nuestra principal Vista**

Dado que la mayor parte de nuestro punto de vista es Maestro Vista de la lista aparejo y concurso completo, vamos a echar un vistazo a lo que se necesita para hacer una lista de los objetos prestados y reactiva dentro de un nativo Reaccionar Vista de la lista. Como siempre, echar un vistazo a la[documentos oficiales ListView](https://facebook.github.io/react-native/docs/listview.html)[39](#page72) para una gran visión de conjunto de la Vista de la lista componente.

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

|  |  |
| --- | --- |
| Tutorial | 68 |

Pongamos aparte nuestra vista Patrón.

* 'Use strict';

2

3 **var** Reaccionar = exigir('Reaccionar nativo')

4 **var** {

* Vista de la lista,

6 } = Reaccionar

7

8 **var** Celda = exigir('./celda')

9

1. **var** suscribibles=exigir('Suscribible')
2. **var** R=React.createClass

12

1. **var** Dominar=R ({
2. mixins: [Subscribable.Mixin],
3. almacenar: **función**() {
4. **devolver este**.props.store
5. },

Así que esto es como empieza, estamos usando el “viejo estilo” Reaccionar convenciones donde React.createClasspara crear un nuevo componente. El “nuevo estilo” es el que alentado por ES6, donde extendemosReact.Component y hacer una adecuada ES6 clase - vamos a llegar a eso también. Una de las razones para utilizar el “viejo estilo” es ser capaz de utilizar mixins existentes fácilmente; no quiere decir que no es posible con código basado ES6, es sólo mucho más fácil y coexiste con código que ves en la Web a partir del momento de la escritura.

El siguiente es nuestra tienda, lo que vamos a llegar a más tarde, y el suscribibles mixin, vamos a llegar a los que más tarde también.

Pasando a la vista de lista.

* getInitialState: **función**() {
* **var** ds= **nuevo** ListView.DataSource ({rowHasChanged:
* (R1, r2) => {

|  |  |
| --- | --- |
| 4 | *// piratear, necesitará inmutabilidad en la tienda para que esto se detecta* |
| 5 | **return true** |

* }
* })
* **var** lista= **esta**.lista de la tienda()

9**regreso** {

1. fuente de datos: ds.cloneWithRows (lista),
2. }
3. },

|  |  |
| --- | --- |
| Tutorial | 69 |

Reaccionar como un componente adecuado, es aconsejable proporcionar una getInitialStateY eso es un gran lugar para establecer nuestra Vista de la listaestado. Si recuerdan nuestra discusión sobre las vistas de lista, recordará que una vista de lista esperar algún tipo de fuente de datos abstracta; este será el siguiente:

* **var** ds= **nuevo** ListView.DataSource ({rowHasChanged:
* (R1, r2) => {

|  |  |
| --- | --- |
| 3 | *// piratear, necesitará inmutabilidad en la tienda para que esto se detecta* |
| 4 | **return true** |

* }

6})

Tu Fuente de datos es lo que lleva a una colección llanura de artículos y lo hace en una adecuada Vista de la listafuente de datos. Aquí estamos inicializarlo con una función de cambio de la cual es responsable de ayudar alVista de la listapara detectar cambios en ella. Sin embargo, hay que tener cuidado aquí - si usted no está haciendo el trabajo adecuado objeto inmutable, no esperar que funcione ya que va a cambiar el contenido de los existentes por el cual elVista de la listaya es titular. Por lo tanto, teniendo en cuenta dos filas,r1 y r2 que se supone que calcular si son iguales o no, y sobre todo que la voluntad significar comparar las referencias de instancia. De lo contrario, significaría comparar los ID de objetos o contenidos, y así sucesivamente - pero eso es algo que no va a salir de, y que tendrá que pagar una multa para la comparación.

La mejor forma posible hacer esto de la manera Reaccionar es utilizar la inmutabilidad y un marco de recogida inmutable como [Immutable.js](https://facebook.github.io/immutable-js/)[40](#page74). colecciones inmutables, también llamadas estructuras de datos persistentes, componer muy bien con Reaccionar, y se puede leer más acerca[aquí](https://en.wikipedia.org/wiki/Persistent_data_structure)[41](#page74) y si es atraído a trabajos académicos existe esta [trabajo seminal sobre el tema](http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf)[42](#page74).

Para concluir, cuando se utiliza herramientas como Immutable.js, es suficiente decir simplemente r1 r2 === y reaccionar nativo será capaz de recoger los cambios de manera muy eficiente. De lo contrario, si su arquitectura de la aplicación no se basa en la inmutabilidad puede ver que Vista de la lista no sería capaz de actualizarse a sí mismo, en cuyo caso me gustaría tratar de forzar cambios y return true en lugar de tratar de crear manualmente el concepto de colecciones inmutables sin colecciones inmutables adecuadas (como la creación de un nuevo objeto de forma manual cuando uno cambia), siempre y cuando se siente seguro de que no hay ningún problema de rendimiento.

Para este libro, tenemos la aplicación colecciones inmutable en una rama separada del código; Sin embargo, para mantener la simplicidad del debate que necesitaremos para mantener lejos de que a medida que queremos centrarnos en las técnicas básicas.

Pasando con el listado de código, vemos esto:

1. <https://facebook.github.io/immutable-js/>
2. <https://en.wikipedia.org/wiki/Persistent_data_structure>
3. <http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf>

|  |  |
| --- | --- |
| Tutorial | 70 |

* didSelectRow:**función**(fila){
* **esta**.props.navigator.push ({
* carné de identidad: 'detalle',
* título: row.title,

5accesorios: { ít: fila,

|  |  |
| --- | --- |
| 6 | almacenar: **esta**.almacenar(), |
| 7 | navEvents: **esta**.props.navEvents}, |

* })
* console.log ("MASTER: seleccionado", Row.title)

10},

11

12didDeleteRow:**función**(fila){

1. **esta**.store (). eliminar (fila)
2. },

15

1. renderRow:**función**(fila){
2. **regreso** (
3. <clave de la célula={} Row.id

|  |  |
| --- | --- |
| 19 | ít={fila} |
| 20 | onDelete={()=>**esta**.didDeleteRow (fila)} |
| 21 | onPress={()=> **esta**.didSelectRow (fila)} |
| 22 | /> |

1. )
2. },
3. hacer: **función**() {
4. **regreso** (
5. <estilo ListView={{acolchado superior: 50, flexionar: 1}}

|  |  |
| --- | --- |
| 29 | fuente de datos={**esta**.state.dataSource} |
| 30 | renderRow={**esta**.renderRow} |
| 31 | /> |

1. )
2. }
3. })

Estamos saltando la funcionalidad de sincronización tienda por ahora, hasta que cubrimos la propia tienda. Vamos a empezar desde arriba abajo. En primer lugar, nuestrahacer función es simplemente una declarativa Vista de la listarender, sin ninguna lógica o ninguna composición UI complejo en absoluto; y que vale para un patrón de Vista de Smart bastante bien. Estamos haciendo los estilos en línea con fines de demostración - el propósito de estos estilos es empujar a la vista de lista hacia abajo por debajo de la barra de navegación, y para asegurarse de que la vista de contenedor se flexiona al máximo lo largo de la ventana de observación (vamos a llegar a un estilo más tarde, pero por ahora se puede Google “FlexBox” para tener una idea del modelo de diseño Reaccionar usos nativos).

Estamos suministrando el fuente de datoscocinamos a través de la hélice bien llamado, y que es simplemente la fuente de datos que se encuentra en nuestro estado. Adicionalmente

|  |  |
| --- | --- |
| Tutorial | 71 |

que tenga que darle una renderRow apoyo, que es sólo una función que devuelve la vista nativo que se supone que representa nuestro elemento de la lista, en nuestro caso React - la Celda.

Moviéndose a renderRow, Simplemente estamos volviendo nuestra Celda componente con unos cuantos accesorios importantes:

* llave - Esta propiedad es necesaria para la recogida de los componentes. Muestra Reaccionar nativo sabe cómo hacer un seguimiento de nuestros puntos de vista. Normalmente se configurarlo como 1: 1 asignación de la identidad del objeto que presta. En nuestro caso se trata simplemente derow.id
* onDelete - esta es nuestra propia prop personalizado. Señala que el usuario tiene la intención derechos de privacidad |
* onPress - esta es nuestra propia prop personalizado. Señala que el usuario desea interactuar con este artículo
* ít - así es como esta célula se pone el modelo de datos que tiene que hacer, nuestra ít.

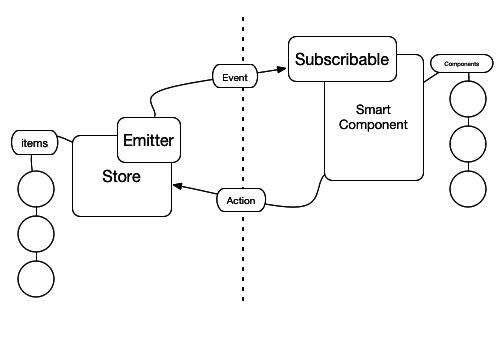
Tenga en cuenta, que la forma en que cada método de interacción “sabe” lo que el tema se tiene que manejar, es que simplemente estrecha sobre ella con una función lambda! Estas funciones de interacción simplemente delegan a todos y cada parte interesada. En el caso de borrado - la tienda recibe una notificación al respecto, y en el caso de la navegación, así, el navegador recibe una notificación al respecto. Vamos a llegar a ambos un poco más tarde.

Esto completa nuestra visión general de la vista Patrón. Pero tenemos un par de agujeros: nuestra tienda Flux y navegante. Vamos a cubrir estos ahora, y después podemos seguir adelante con el resto de los puntos de vista.

**La tienda**

Nuestra tienda es simplemente un lugar donde se llevan a cabo y la forma de nuestros elementos de datos. Además recibe acciones (en forma de métodos) y envía de vuelta eventos. La forma en que se relaciona entre con su entorno es simple pero potente:

|  |  |
| --- | --- |
| Tutorial | 72 |



**Nuestro Pipeline Flux-como**

Vamos a echar un vistazo cómo hacemos una tienda tal por dejar caer la ficha de Play Store y la clase anuncio esté completamente anotada.

* 'Use strict';

2

3 **importar** emisor de 'EventEmitter'

4 **importar** \_ de 'Lodash'

5 **importar** UUID 'Uuid-js'

6

7 **clase** grupo {

8**estático** withTitle (título) {

9**return new** Grupo (uuid.create (1) .toString (), título, **nuevo** Fecha)

1. }
2. **estático** fromJSON (en bruto) {
3. **var** marido=JSON.parse (en bruto)
4. h.date = **nuevo** Fecha(H.date)
5. **regreso** \_.unir(**nuevo** Grupo (), h)
6. }
7. constructor (ID, título, fecha) {
8. **esta**.carné de identidad=carné de identidad

|  |  |
| --- | --- |
| Tutorial | 73 |

1. **esta**.título=título
2. **esta**.fecha=fecha
3. **esta**.etiqueta="Tag-ninguno"
4. **esta**.contactos={}
5. }
6. }

Así que ahora estamos utilizando ES6, con el nuevo importarsintaxis y adecuados clases, métodos estáticos y un constructor. También podríamos haber utilizado[param por defectoetros](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/default_parameters)[43](#page78), pero vamos a mantener las cosas así por razones de simplicidad.

los Grupo clase es bastante simple, que ejerce en nuestros datos ya diferencia de la celebración de un simple diccionario Javascript para datos, tenemos una opción para adjuntar cualquier lógica nos gustaría directamente en el Grupoclase si pertenece allí, lo que es programación orientada a objetos clásico. También nos aseguramos de que cada grupo tiene una identificación global única, o un UUID, con la norma[UUID V1](https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_1_.28MAC_address_.26_date-time.29)[44](#page78), Lo cual es suficiente para nuestro caso.

También tomamos una referencia a la EventEmitter suministrada por Reaccionar nativo en sí, no hay necesidad de conseguir uno de un módulo comunidad externa.

Pasando a la tienda.

* **clase** tienda {
* constructor(){
* **esta**.artículos=[]
* **esta**.eventos= **nuevo** Emisor ()

5}

Hemos creado nuestros artículos y una instancia de un emisor. Esto ya indica que probablemente vamos a necesitar sólo una única instancia de tienda a estar viajando a lo largo de nuestros objetos vivos, ya que queremos que todos se registra al mismo emisor de sucesos, porque de lo contrario se anularía el propósito de un emisor de sucesos despacho central.

* lista de carga){
* **esta**.artículos=\_.map (lista, (en bruto)=>{

3**regreso** Group.fromJSON (en bruto)

* })
* **esta**.ordenar()
* **esta**.publicar("cambio", {Acción:"carga"})

7}

Esta es la forma en que Deserialize nuestra tienda, un JSON prima se suministra externamente cuando se inicia el app arriba y configura cada componente central como el

1. <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/default_parameters>
2. <https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_1_.28MAC_address_.26_date-time.29>

|  |  |
| --- | --- |
| Tutorial | 74 |

almacenar. Por lo tanto, simplemente carga que JSON como una matriz, y dejar que la clase de grupo decidir cómo deserializar objetos en vinagre para vivir queridos. Finalmente clasificamos por si acaso nos presentaron datos no ordenados, y publicar un evento de cambio. Si usted está mirando\_.mapa y rascándose la cabeza sobre el \_símbolo. Bueno, eso es una variable regular, la biblioteca lodash (y antes de ella, la biblioteca subrayado) eran muy populares cuando la biblioteca Javascript Backbone hizo su debut (el amanecer de una sola página Web Apps) y su propósito era llevar el estilo de manejo colección funcional a Javascript. Por lo que encontrará muchos operadores de recolección funcionales, tales como mapas, cada uno, seleccionar, rechazar y así sucesivamente.

* lista(){
* **devolver este**.artículos
* }

4

* encontrar Identificación){
* **regreso** \_.encontrar(**esta**.artículos, **esta**.\_byId (id))
* }

8

9**nuevo**() {

1. **regreso** Group.withTitle ('Intitulado')
2. }

12

13

1. actualización (elemento) {
2. \_.retirar(**esta**.artículos, **esta**.\_byId (item.id))
3. **esta**.items.push (punto)
4. **esta**.ordenar()
5. **esta**.publicar("cambio", {Acción:"actualizar", tema:ít} )
6. }

20

21

1. remover el artículo){
2. console.log (`Extracción **ps**ít**}** (**ps**Identificación del artículo**}**) de **psesta**.artículos**}** (**ps**\_.map (TH\
3. is.items, (i)=> i.id)**}**) `)
4. **var** remoto=\_.retirar(**esta**.artículos, **esta**.\_byId (item.id))
5. **esta**.publicar("cambio", {Acción:"retirar", tema:ít})
6. }

A continuación se muestran las operaciones típicas CRUD. Manipulamos nuestra internaartículos lista como una respuesta para cualquiera de los actualizar o retiraracciones, ya que éstos están haciendo las escrituras, que también publican los eventos correspondientes. Además también ofrecemos la consulta conlista y encontrar, Donde ya puedo decir que usted reacciona con sólo necesitan lista- Reaccionar sabrá cómo averiguar cuál de los artículos han cambiado para volver a hacer, que es impresionante. Esta es la razón por marcos como Redux no se molestan con los eventos de cambio o el acceso de grano fino de grano fino

|  |  |
| --- | --- |
| Tutorial | 75 |

a otro, simplemente se entregan a todo el estado tienda y se dejan reaccionar haga el trabajo pesado!

* ordenar(){
* **esta**.items.sort ((a, b)=>{
* **regreso** -1 \*(b.date-una cita)
* })
* }

6

7(publicación de eventos, ... args) {

* **esta**.events.emit (evento, ... args)

9}

10

1. \_byId (id) {
2. **función de retorno**(candidato){
3. **regreso** Identificación del candidato==carné de identidad
4. }
5. }
6. }

17

1. module.exports = Almacenar

Restantes son algunas funciones de mantenimiento de separación, la edición, y una función de buscador interno. Tenga en cuenta que exportamos nuestro módulo de la forma tradicional y no conpor defecto de exportación ya que al momento de escribir estas líneas Reaccionar nativo no admite que todavía.

**Secuencia de arranque y de navegación**

Analicemos nuestro punto de entrada, en este caso index.ios.js (Que es similar a la nuestra .androideuno). En la cabecera del archivo, es nuestro código de programa previo. Aquí vamos a necesitar para inicializar nuestros principales tiendas, navegación (que solo recibió uno de cada uno a través de toda la aplicación) y cualquier servicio de limpieza que se necesita para el arranque de la aplicación. Como preparación para el cableado de las rutas de navegación importamos todos los puntos de vista relevantes también.

|  |  |
| --- | --- |
| Tutorial | 76 |

* 'Use strict';

2

3 **importar** Reaccionar, {

* AppRegistry,
* hoja de estilo,
* Texto,
* Ver,
* Navegador,
* TouchableOpacity, 10 } de 'Reaccionar nativo' 11

12 **importar** maestro de './views/master'

13 **importar** detalle de './views/detail'

14 **importar** Gente de './views/people'

15 **importar** tienda de './services/store.js

dieciséis **importar** almacenamiento de './services/storage.js

17 **importar** icono de 'Reaccionar nativo-vector-iconos / FontAwesome' 18 **importar** emisor de 'EventEmitter'

19 **importar** estilos de './styles

Importar las partes de interfaz de usuario que utilizaremos más adelante en nuestro mapeo de navegación, y cada uno de los bloques de construcción de nuestra aplicación - el Maestro, el detalle y vistas gente inteligente, así como nuestra almacenamiento, estilos, y el emisor evento de navegación.

* tienda var = nueva tienda ()
* almacenamiento var = nuevo almacenamiento ()

3 storage.load ((err, artículos) => {

* si (err) {
* console.log ( "ALMACENAMIENTO: error de carga de almacenamiento", err)
* regreso
* }
* store.load (artículos) 9 })

10

1. storage.syncWhenStoreChanges (la tienda)

Tienda es nuestro almacén de objetos en vivo que nuestra aplicación interactúa con, y Almacenamientoes un componente de servicio que escucha a nuestra tienda, persistirlo cuando detecta cambios de forma automática por nosotros (vamos a revisar este simple componente por separado). Este es un buen beneficio de un despacho evento central - la tienda no sabe cómo conservar en sí, ni tampoco necesita preocuparse por eso. Al final, terminamos con nuestro ejemplo, tienda,almacenar, Que vamos a enchufe felizmente en nuestras rutas que hacemos un mapa de nuestra pila de navegación.

Esta es la misma instancia tienda que nuestra vista Patrón obtendrá, y suscribirá con el suscribibles mixin (Prometí Ataremos que uno arriba eventual).

|  |  |
| --- | --- |
| Tutorial | 77 |

El siguiente es un tema bastante pesado, pero su quizás la más crucial para nuestra aplicación. He dedicado un capítulo completo para un análisis en profundidad de enrutamiento Reaccionar nativo y la pila de navegación, así que espero que disfrutará de ese tema por sí solo.

Para nuestra aplicación, esta es la forma en la pila de rutas y navegación parezca, poco a poco diseccionado.

* **var** navEvents= **nuevo** Emisor ()

2 **var** \_navegador

Tendremos que ambos éstos. navEventses un emisor de sucesos de despacho que se encarga de emitir eventos de navegación. Por ejemplo, si alguien tocó el botón 'Guardar' y ese botón existe en la barra de navegación, no hay garantía de una visión lo sabrá porque por diseño, son nuestros barra de navegación, mapas y rutas separadas de cualquier vista de la preocupación específica. Así que esta es la forma recomendada de hacer las cosas - a mantener un emisor de sucesos dedicar para comunicar eventos abajo la barra de navegación. Cuando llegamos a laDetalle punto de vista, donde se puede editar un grupo, vamos a ver cómo interactúa con el navEventsemisor cuando se quiere salvarse a sí misma. Consejo: arquitectónico, también se puede vincular esta comunicación con otra entidad central para su aplicación, se puede adivinar cuál es? (Pista -. Su tienda Bueno, no realmente una pista :-)

Mantenemos una \_navegadorvariable global ya que como ocurre (pero que no se ve aquí todavía), Android puede navegar de manera que son diferentes a IOS - a través del botón “Volver” de hardware (o software). Se trata de cómo podemos tomar nuestro navegador en vivo y el alambre a través de nuestra lógica de gestión con el botón de regreso; esto se describe con más detalle en el capítulo de enrutamiento y navegación.

El siguiente es nuestra asignación de ruta de navegación y la navegación. En primer lugar, ¿cuál es el sentido de todo esto? (Se puede vislumbrar en el listado de código de abajo y volver aquí).

* Navegador - este es un componente reaccionar. Es el principal punto de entrada para el Reaccionar cosas nativo necesita saber con el fin de rendir un bar totalmente funcional y de navegación interactiva: estilos, ruta inicial, la representación ruta, componente de la barra de navegación, y dentro de ella un asignador de ruta
* Una ruta - es un simple diccionario. Lo mejor sería seguir una convención y darle forma, como a continuación, con una ruta dedicadacarné de identidad, título y accesorios lo que vamos a pasar a nuestros puntos de vista directamente (por patrones más interesantes, saltar al capítulo de enrutamiento y navegación)
* Una ruta inicial - simplemente la primera ruta que la aplicación va a emitir. Esto a su vez invocar larenderScene la función que a su vez instruir Reaccionar nativo qué componente de vista real para hacer
* renderScene - una función que mapea una ruta en un nativo Reaccionar real

componente, lleno configurado y listo para funcionar. Aquí suelen tener identificadores de ruta comodominar, detalle y tal vez más URI similar a como app: // usuarios / 1/2 / carros / artículos - que hace las reglas.

|  |  |
| --- | --- |
| Tutorial | 78 |

• barra de navegación - con el NavegadorPuede proporcionar cualquier componente de la barra de navegación para hacer que la interfaz de usuario real de la barra de navegación, con interacciones incluidos. Aquí nos estamos tomando la barra de navegación por defecto, siempre que:Navigator.NavigationBar y la aplicación de nuestra propia routeMapper por lo que lo hará reflejar las rutas que se muestran actualmente de manera significativa (más sobre esto en el capítulo de enrutamiento y navegación).

* routeMapper - una función especial que toma una ruta y refleja las partes que son típicos de una barra de navegación: el botón izquierdo, el título, y el botón derecho.
* **clase** Navegación **se extiende** React.Component {
* hacer(){
* **regreso**(
* <Navegador
* estilo={} Styles.container
* initialRoute={{carné de identidad:'dominar', título: "Gente", puntales:{}}} navEvents
* renderScene={**esta**.navigatorRenderScene}
* barra de navegación={
* <Navigator.NavigationBar

|  |  |
| --- | --- |
| 10 | routeMapper={} NavigationBarRouteMapper |
| 11 | estilo={} Styles.navBar |

1. />
2. }
3. />
4. )
5. }

Siguiendo, así es como nuestra renderScene la función se parece.

* navigatorRenderScene (ruta, navegador) {

2\_navegador = navegador

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

4**caso** 'dominar':

* **regreso** <maestro navegador={navegador}

|  |  |
| --- | --- |
| 6 | almacenar={almacenar} |
| 7 | {...} route.props |
| 8 | /> |

* **caso** 'detalle':

1. **regreso** <detalle del navegador={navegador}

|  |  |
| --- | --- |
| 11 | almacenar={almacenar} |
| 12 | {...} route.props |
| 13 | /> |

1. **caso** 'gente':
2. **regreso** <Las personas navegador={navegador}

|  |  |  |
| --- | --- | --- |
|  | Tutorial | 79 |
| dieciséis |  | almacenar={almacenar} |
| 17 |  | {...} route.props |
| 18 |  | /> |
| 19 | } |  |
| 20 | } |  |
| 21 | } |  |

El enrutador Reaccionar nativo y la renderScenefunción le permite separar por completo las preocupaciones en su aplicación. losDominar componentes tiene ni idea de quién es el Detallecomponentes y así sucesivamente. Todo el recorrido se realiza con un resumenrutaobjeto, y esta es la parte de la pila que convierte esa abstracción en detalle de implementación real. Tenga en cuenta que estamos pasando el navegador y tienda para los puntos de vista que lo necesitan - típicamente todas las vistas. Esto se DRY'd fácilmente con un poco de abstracción mayor comodidad se puede hacer, pero estoy manteniéndolo adicional pie de la letra lo que ayudaría a impulsar la idea en cuestión. Un buen uso de la nueva[operador de difusión](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Spread_operator)[45](#page84) es pasar por los apoyos que recibimos a través de la ruta.

Finalmente aquí está la pieza de lógica que hace que la barra de navegación render sí correctamente como un reflejo de la ruta actual:

* var NavigationBarRouteMapper = {
* LeftButton: function (ruta, navegador, índice, navState) {

3si (índice === 0) {

4return null

* }
* var previousRoute = navState.routeStack [index - 1]
* regreso (

8

* **<TouchableOpacity**

1. onPress ={() =**>** navigator.pop ()}
2. style = {} styles.navBarLeftButton>
3. **<texto** style ={[Styles.navBarText,styles.navBarButtonText]}**>**
4. **<Icono** name ="Chevron-izquierda"

14 size ={18} /**>**

1. {} PreviousRoute.title
2. **</ Texto>**
3. **</ TouchableOpacity>**
4. )
5. },

20

1. RightButton: function (ruta, navegador, índice, navState) {
2. interruptor (route.id) {
3. caso 'maestro':
4. regreso (
5. <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Spread_operator>

|  |  |
| --- | --- |
| Tutorial | 80 |
| 25 | **<TouchableOpacity** |
| 26 | onPress ={() =**>** navigator.push ({id: 'detalle', título: "Nuevo", puntales \ |

1. : {NavEvents}})}

|  |  |
| --- | --- |
| 28 | style = {} styles.navBarRightButton> |
| 29 | **<Icono** style ={[Styles.navBarText,styles.navBarButtonText]} |
| 30 | name ="más" |
| 31 | size ={18} /**>** |
| 32 | **</ TouchableOpacity>** |

1. )
2. caso 'detalle':
3. regreso (

|  |  |
| --- | --- |
| 36 | **<TouchableOpacity** |
| 37 | onPress ={() =**>** route.props.navEvents.emit ( "guardar")} |
| 38 | style = {} styles.navBarRightButton> |
| 39 | **<texto** style ={[Styles.navBarText,styles.navBarButtonText]}**>** |
| 40 | Salvar |
| 41 | **</ Texto>** |
| 42 | **</ TouchableOpacity>** |

1. )
2. }
3. },
4. Título: function (ruta, navegador, índice, navState) {
5. regreso **<texto** style ={[Styles.navBarText, styles.navBarTitleText]}**>**{Route.t \
6. TÍTULO}**</ Texto>**
7. },
8. }

Esto parece un poco complicado, ya que estamos escupiendo trozos de interfaz de usuario reales para el botón de la derecha y el título (que podría ser fácilmente extraído a cabo, por supuesto). Pero si usted escudriña verá que lo que realmente es simple; el botón izquierdo normalmente solo necesita pasar un elemento de la pila de desplazamiento, y que es lo que describe el código, el título tiene que tirar de latítulo clave de nuestra ruta (por suerte tenemos que listo para su uso justo ahí!), y el botón derecho tenemos que hacer el trabajo en función de la ruta carné de identidad:

* dominar - Si estamos en el *Dominar* vista, queremos navegar a la *Detalle* ver para hacer un nuevo grupo. Así que este es un botón “Nuevo” tipo y comportamiento - navegar a vista de detalle.
* detalle - Si estamos en el *Detalle* vista, es probable que quieren guardarlo. Asi que esto significa emite un evento “salvar” a la navEventsemisor de despacho. Sabemos que la propia Vista detallada está escuchando en ese evento, por lo que a su vez activará una real SAVE (veremos que más tarde, cuando diseccionar la vista Detalle)

Esto completa el cuadro de los boostrap, navegación, y ver su Maestro partes de la aplicación. En este punto se ceba nuestra opinión y nuestro Maestro pila de navegación nos llevará a cualquier otro punto de vista que queremos.

|  |  |
| --- | --- |
| Tutorial | 81 |

Ahora, todo lo que queda es tomar un vistazo a la llamada Vista detallada, o en la pantalla de edición de grupo, y la pantalla de Contactos. Esto significa que vamos a discutir la interfaz de usuario, el estilo, eventos, y el uso de componentes tercera parte y la integración.

**Styling**

Antes de continuar, vamos a echar un vistazo a cómo el estilo se hace con Reaccionar nativo. Creo que será una grata sorpresa.

Reaccionar estilo nativo se realiza con Javascript, pero se siente como CSS. Si se han fijado la comunidad Reaccionar en algún momento en el tiempo comenzó a moverse hacia la tendencia de los estilos en línea, y recuerdo[esta cubierta](https://speakerdeck.com/vjeux/react-css-in-js)[46](#page86) para ser un abridor de ojos - me ha gustado el proceso enseñado tanto que se mantuvo persistente durante un tiempo, y yo estaba feliz de ver los resultados en React nativo cuando finalmente salió.

Con el fin de crear una hoja de estilo, o una descripción del estilo utilizable, echar un vistazo a la siguiente:

* **var** S=StyleSheet.create

2 module.exports = S ({

* mensaje de texto: {
* tamaño de fuente: 17,
* fontWeight: '500',

6relleno: 15,

* marginTop: 50,

8marginLeft: 15,

9},

1. botón: {
2. color de fondo: 'blanco',
3. relleno: 15,
4. borderBottomWidth: StyleSheet.hairlineWidth,
5. borderBottomColor: '#CDCDCD',
6. },
7. :
8. :
9. .

Me gusta la taquigrafía React.create y StyleSheet.create funciona como R y Srespectivamente. Pero a nuestro punto, la función de crear se pasa un diccionario que tiene el aspecto de una hoja de estilo CSS ordinaria, donde los guiones se sustituyen con carcasa de camello. Más encima, Reaccionar nativo tiene un conjunto mucho más pequeño de propiedades que se pueden jugar con una pila que con CSS regular en un navegador moderno, y por una buena razón - que no tiene que apoyar los modelos de diseño rotas o décadas de accidentes debidos al navegador guerras.

Para averiguar qué propiedades que se pueden utilizar, revisar la [documentos oficiales estilo](https://facebook.github.io/react-native/docs/style.html#supported-properties)[47](#page86).

1. <https://speakerdeck.com/vjeux/react-css-in-js>
2. <https://facebook.github.io/react-native/docs/style.html#supported-properties>

|  |  |
| --- | --- |
| Tutorial | 82 |

Después de haber hecho su hoja de estilo, puede exportarlo como con la lista anterior, y utilizarlo como una variable de este modo:

* <Ver estilo={} Styles.myCustomStyle />

Dónde estilo es un apoyo especial Reaccionar nativo utiliza para aplicar estilos, y existe en todos los Reaccionar componente nativo que soporta el estilo - que debe seguir esta convención, así como en sus propios componentes personalizados.

Además, puede inline sus estilos de este modo:

* <Ver estilo={{relleno: 10}} />

Y, por supuesto, se puede pasar una variable en su lugar:

* *// algún lugar alrededor de su código:*

2 **var** ViewStyle = { relleno: 10 }

3 <Ver estilo={} ViewStyle />

Aquí, usted está proporcionando un diccionario prima, así que por qué incluso utilizar StyleSheet.create? Bueno, ya que creará una tabla inmutable e internados que es más ligero para transmitir renders. Para más información sobre esto, echa un vistazo a la[estilo oficial](https://facebook.github.io/react-native/docs/style.html) [docs](https://facebook.github.io/react-native/docs/style.html)[48](#page87).

La propiedad en cascada de CSS no se ha ido bien, puede utilizar un efecto de cascada limitada con Reaccionar nativo así y hace que para una construcción muy útil. Simplemente suministrar una variedad de estilos en lugar de un objeto, en línea o como una variable:

1 <Ver estilo={[{relleno: 10}, Styles.myCustomStyle, {color: "rojo"}]} />

El orden de aplicación de los estilos (en caso de conflicto de défini-ciones) se deja a la derecha. Por lo tanto, mantener sus estilos de base en la esquina izquierda de la matriz.

Otra sorpresa agradable es - es su hoja de estilos Javascript. Esto significa que puede hacer cualquier cosa que un preprocesador CSS como Sass o menos pueden hacerlo, sin la totalidad de esa tecnología. Se pueden definir y utilizar variables para indicar los colores primarios y secundarios de aplicaciones, un tema para toda la aplicación, la altura de cálculo, el ancho y la posición sobre la marcha, hacer una fábrica de generadores de estilos e incluso componer hojas de estilo en formas más interesantes, como objetos de Javascript de civil , si te gustaria.

El modelo de diseño que reaccionan instrumentos nativos es Flexbox, mientras que podemos masticar a través de las páginas que muestran diferentes diseños y cómo llegar a ser,

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

|  |  |
| --- | --- |
| Tutorial | 83 |

Creo que es una pérdida de tiempo ya que Flexbox está tan bien explicado en otro lugar por ahora. Para hacerse una idea de ella, echa un vistazo a esta web basado[parque infantil Flexbox](http://the-echoplex.net/flexyboxes/)[49](#page88), Es una manera rápida e interactiva para empezar. A continuación, se puede extraer las propiedades específicas de FlexBox Reaccionar nativo en la[página docs](https://facebook.github.io/react-native/docs/flexbox.html)[50](#page88) en vez de mí sólo copiar y pegar aquí y aumentando el número de páginas del libro sin razón :-).

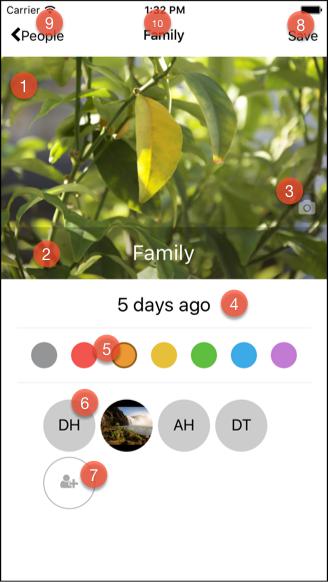
**La imagen detallada**

O: la pantalla de edición de grupo. En esta parte vamos a inmersión profunda en la interfaz de usuario orientada más temas tales como la forma de componer puntos de vista, los diferentes componentes de interfaz de usuario Reaccionar nativos, componentes y un estilo de 3 ª parte.

Vamos a empezar con una anotación de ver una muestra de población:

1. <http://the-echoplex.net/flexyboxes/>
2. <https://facebook.github.io/react-native/docs/flexbox.html>

|  |  |
| --- | --- |
| Tutorial | 84 |



**Pantalla detallada anotado**

1. Un soporte de imagen de la portada. Se trata de un nativo Reaccionar regularesImagen componente.
2. Un cuadro de texto editable. Se trata de un nativo ReaccionarTexto componente de estilo para un aspecto limpio.
3. UN TouchableOpacity componente tiene una visión dentro de él y funciona como

un botón. Aquí estamos incrustar unaIcono componente de la -reaccionar-vector-iconos nativos paquete.

1. Una marca de tiempo en vivo, traducido a una moda “hace tiempo” con la biblioteca Javascript moment.js familiares. Esta biblioteca se utiliza comúnmente en la Web, pero funciona sin problemas en React nativo, como otras librerías Javascript lisos haría.
2. La sección de las etiquetas es una colección generada de puntos de vista que incrustar en un soporte de lado. Estos son una serie de vistas envueltos conTouchableOpacity para la interactividad.
3. La sección de contactos se genera, así, a partir de la descripción de contactos que actualmente están asociadas con este grupo. Utilizamos nuestra propiaAvatar componente, que tiene como protagonistas en otros puntos de vista también.