React Native

Bootcamp: Desenvolvedor de Mobile Apps

Marcelo Sampaio

2020

React Native

Bootcamp: Desenvolvedor de Mobile Apps

Marcelo Sampaio

Sumário

Capítulo 1. Ligando os pontos – Conceitos fundamentais

É impossível falar de React Native sem antes falar de React quando o grande slogan do do React Native desde seu surgimento é: "*Learn once, code everyhhere".* Na verdade, a curva de aprendizado de um programador React para o Native é bem suave.

Dessa forma, vamos trabalhar em três frentes nesse curso:

- Fazer você saber o máximo possível sobre React. Vamos tentar sair do básico para alguns tópicos um pouco mais avançados, como Redux e Middlewares. Atenção! Redux não será visto nesta apostila, ele será apresentado em uma das aulas interativas.
- Explicar o ambiente do React Native.
- Explicar as particularidades do mundo mobile e como o React se encaixa nele.

Nas próximas seções, vamos conceituar muitos pontos importantes e tentar juntá-los de forma a fazer sentido como uma coisa só.

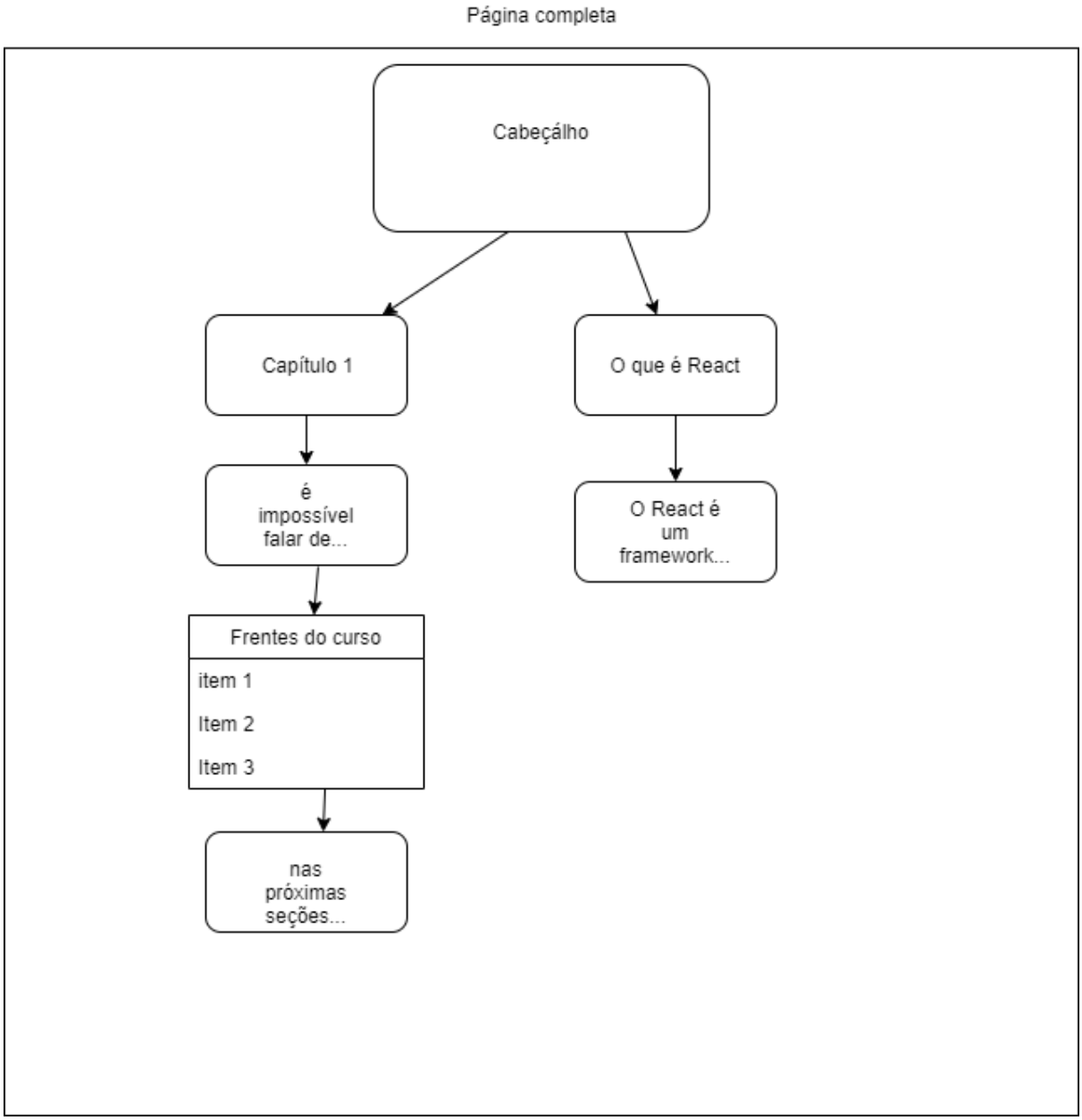
O que é o React e React Native

O React é um Framework que permite a criação de SPA (Single Page Applications). Já vimos um pouco no módulo de fundamentos o que é uma SPA, e vimos também que o maior trunfo do React é o Virtual DOM.

Relembrando, o DOM é o modelo em que um documento HTML é "traduzido" para o navegador. É um modelo documental em que vários objetos são montados de forma hierárquica.

Para entender melhor, imagine que essa página fosse ser representada de forma hierárquica, como demonstrado pela figura 1. Agora, imagine que eu queira alterar o item 3 desta página. Você teria que acessar a página e percorrer todo a hierarquia até chegar ao item que você quer alterar.

Figura 1 – Representação em documento da página anterior.



Agora imagine que você faça isso várias vezes por segundo para um modelo bem mais complexo (a árvore que nós criamos é muito simples, normalmente temos centenas de nós em uma árvore DOM típica).

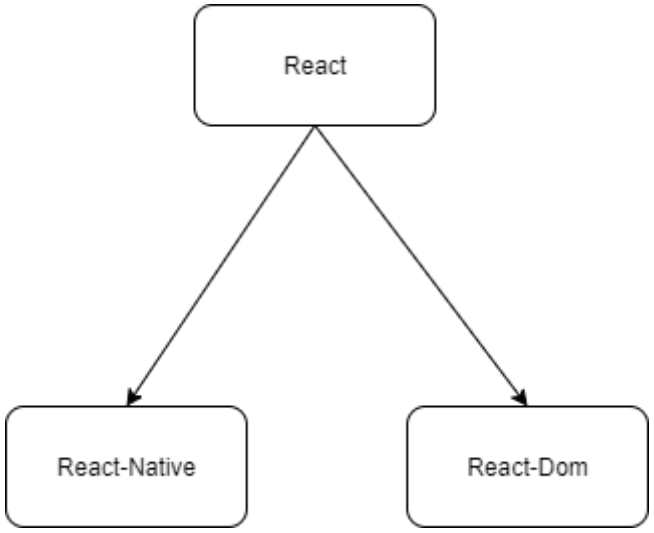
O que o Virtual DOM do React faz é criar um DOM virtual e alterar nele. Depois, um processo chamado Fiber faz a reconciliação e altera o DOM real.

Mas quais são as vantagens?

Primeiro, o React tem um apontador para cada item do DOM real. Assim, ele não precisa percorrer cada item da árvore para cada atualização. Segundo, ele pode realizar as alterações de forma batch e, caso perceba que uma nova alteração foi pedida, pode incluir nesse batch. Isso é um pouco do que o Fiber faz.

Durante o curso vamos aprender muito sobre o React, e ficará mais claro muito do que ele faz. Agora, no entanto, vamos passar para o React Native. Como ele se liga ao React? A figura abaixo ilustra esse processo.

Figura 2 – Representação em documento da página anterior.



O React tem uma biblioteca básica que realiza coisas comuns aos dois mundos (Web e Mobile). Essa biblioteca gerencia coisas como estado, hooks, componentes, Virtual DOM, etc. Não se preocupe, vamos falar sobre todos eles durante o curso. Já a biblioteca React-Dom faz a renderização no Browser. Repare que o virtual dom está na React, a React-Dom é a que realiza a atualização no DOM Real.

O que o React Native faz é carregar o JavaScript que escrevemos e executá-lo dentro de cada plataforma (IOS e Android), utilizando para isso uma camada que faz as chamadas nativas para cada plataforma.

A explicação foi extremamente simplista. Preferimos não ir muito mais a fundo por ser o primeiro contato com React Native. No entanto, ela com certeza é suficiente para os nossos propósitos.

Ambiente de desenvolvimento – Ferramentas

Durante as nossas aulas práticas, faremos a instalação de várias ferramentas que serão utilizadas no desenvolvimento nativo. Enquanto a sua instalação é bem simples, entender o que cada ferramenta fará por nós é interessante para que tenhamos uma boa visão do todo:

- **NodeJs**

O NodeJs é uma ferramenta que permite que código escrito em JavaScript rode diretamente no seu computador. No primeiro momento, o JavaScript foi criado para rodar apenas em navegadores da internet. Essa era a sua grande utilidade, tornar a Web um pouco mais dinâmica. O NodeJS aproveita-se de uma engine de JavaScript chamada V8. Essa engine foi criada para o Google Chrome, mas hoje é utilizado para diversos outros fins.

E queal é o papel do NodeJs no nosso projeto? Ocorre que tanto React quanto o React Native utilizam o Node como base para todo o ambiente de desenvolvimento.

- **NVM**

É uma espécie de gerenciador do NodeJs. O utilizaremos para confirmar que estamos utilizando a versão correta do NodeJs, além de baixarmos a versão mais atualizada do Node

- **Npm e Yarn**

São gerenciadores de pacotes utilizados pelo NodeJs. O gerenciador de pacotes padrão é o npm, mas o yarn começou a fazer bastante sucesso pela velocidade com que baixava os novos pacotes. A comunidade React utiliza bastante

o Yarn. Vamos ensinar o básico de cada um deles e você escolhe qual utilizar. Em alguns casos vamos preferir o Yarn, pois é a opção escolhida pelo ecossistema.

- **Android Studio**

É a ferramenta de desenvolvimento de código nativo do Android. Mas então, por que ele está aqui? Bom, nós precisamos de um emulador para rodar nosso código, certo? O Android Studio já vem com um gerenciador de imagens de Android com diversos modelos. Com o tempo, você verá que é possível instalar pela linha de comandos os emuladores, mas optamos pela forma mais fácil nesse caso.

- **Expo**

O Expo é um conjunto de ferramentas que ajudam em muito o desenvolvimento de aplicações React Native. Em especial, vamos utilizar três ferramentas:

- **Explo Client**, que é um cliente para o seu celular que permite que você teste a sua aplicação.
- **Expo Cli**: ferramenta de linha de comando que permite criar e iniciar uma nova aplicação em React Native.
- **Expo SDK**: serve para acessar API nativa dos dispositivos móveis de uma forma mais simples.

Capítulo 2. Noções de Programação em React

Vamos, nesse capítulo, conceituar o mais básico do React. Acredito que se você entender bem esses conceitos terá muita facilidade com o resto do curso. Se precisar, volte, leia novamente e até procure outras referências na internet (vamos citar alguns bons artigos na bibliografia ao final da apostila).

Componentes

A programação em React é totalmente baseada em componentes. A primeira coisa que você deve aprender é que cada parte da tela é um componente, e isso torna o framework bastante modularizável, permitindo alto aproveitamento de código. Vamos fazer um pequeno exercício: olhe para a tela abaixo e diga quantos componentes você imagina que ela possa ter. Não há resposta 100% correta nem errada. Eu já vi alunos com ideias sensacionais sobre esse exemplo. Na próxima página eu direi qual é a minha forma de pensar sobre ela:

Figura 3 –Tela de criação de um novo usuário.

Bom, pensando apenas nos componentes visuais, eu diria:

Tela: sim, a tela, de uma forma geral, é um componente. Ela é um tipo especial que normalmente chamamos de conteiner. No entanto, nada mais é do que um componente.

Título: se todas as nossas telas tiverem um título, talvez valesse a pena termos um componente assim.

Campo: se juntarmos a descrição e o campo em si (lugar onde digito um valor), podemos criar um componente Campo. Nessa tela utilizaríamos esse componente duas vezes, pois temos dois campos. O nosso campo pode ser mais sofisticado: ele pode, por exemplo, fazer a validação ao algo ser digitado.

Botão Salvar: também outro componente bastante importante em uma app mobile. Você pode não entender ainda como ele pode ser reutilizável, mas durante o curso entenderá melhor.

Form: sei que falamos de apenas componentes visuais, mas existe algo intrínseco nessa tela que é um formulário. Ele englobaria os dois campos e o botão. Poderia, por exemplo, ser o responsável por coordenar as ações de habilitar e desabilitar o botão de salvar conforme os campos estiverem preenchidos ou não.

Em alto nível (perdoem-me os programadores de React pela heresia), o código para essa tela (ou seja, o que estaria dentro do componente Tela) seria assim:

```
<>

<Titulo descricao={”Criar novo usuário”}/>
<Form>
   <Campo descricao={“Usuario”} tipo={Text} nome={usuario}/>
   <Campo descricao={“Senha”} tipo={Password} nome={senha}/>

      <Botao/>
</Form>
```

</>

Não se preocupe com o que está dentro de cada componente (descrição, tipo etc.). Vamos falar disso na próxima seção.

Por enquanto vamos entender alguns conceitos:

1. O formato do código é muito parecido com o de um HTML, certo? Na verdade estamos utilizando JSX. Entenda o JSX como um HTML que permite que escrevamos código JavaScript no meio dele. Ainda utilizaremos muito JSX, especialmente nas aulas práticas.

2. Da mesma forma que temos em HTML os componentes que são auto contidos (não tem nenhum filho) e os componentes que tem filhos, alguns dos nossos componentes tem filhos (Form) e outros não (Titulo, Campo). Repare que os que não tem filhos terminam com uma barra ao final dele.

3. Para terminar, o que significam esses símbolos <> e </>?

Bom, JSX significa JavaScript XML, e ele traz uma regra básica: só podemos ter um componente “Pai”. Como nesse caso temos título, form, campos e botão, precisamos de um pai único. O React criou, então, o conceito de fragmento. Os símbolos <> e </> são “apelidos” para <React.Fragment> e </React.Fragment>. Os fragmentos não fazem nada de importante. Eles apenas são responsáveis por nos ajudar a respeitar a regra de apenas um pai.

Propriedades

Você deve ter percebido que em todos os componentes temos algumas informações passadas para eles. Em React, nós chamamos isso de propriedades. Imagine as propriedades como parâmetros que passamos para uma função.

Um componente apenas **recebe** propriedades, ele **não pode retornar** dados. Isso é extremamente importante ao programar em React.

Outra característica da propriedade: ela é **somente leitura**. Essa característica é conhecida como **One Way Binding**. Ela é extremamente importante para ajudar a testar nossa aplicação, além de deixar nosso pensamento bastante compartimentado.

Se um componente pode apenas receber parâmetros, você não verá aquele emaranhado de informações sendo trocadas de todos os lados. O React nos obriga a programar mais limpo dessa forma. É muito comum no início achar que essa restrição nos atrapalha, mas acredite, ela é a solução da sua vida!

Gerenciamento de estado

Essa foi outra novidade trazida pelo React (e copiada por outros frameworks). Na verdade, é daqui que podemos utilizar o termo “React”.

O estado é algum valor (variável). Como toda variável, ela pode mudar durante a execução do nosso programa. No entanto, os estados são especiais, isso porque o React “reage” às mudanças nessas variáveis, gerando uma nova renderização do nosso componente.

Imagine que a nossa tela fosse modificada para incluir uma mensagem ao final dela. Ela poderia, por exemplo, mostrar se o usuário foi cadastrado corretamente ou mostrar que houve um erro ao cadastrar esse usuário.

Podemos fazer isso utilizando estados. Imagine que você tenha um estado chamado mensagem que, quando o componente é criado, está vazio.

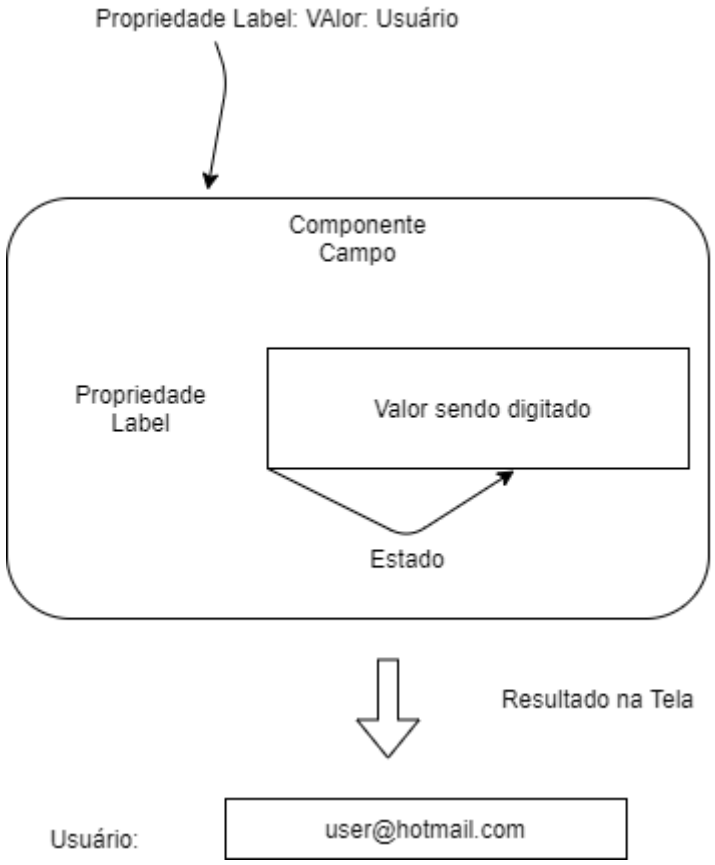
Logo que clicamos no botão salvar, chamamos nossa API. Com o resultado dessa chamada saberemos se foi bem-sucedida ou não. Daí, mudamos nosso estado mensagem para a mensagem que quisermos. Ao perceber que o estado mudou, o React realiza uma nova renderização do componente, mostrando nossa mensagem.

No entanto, precisamos entender uma coisa a respeito dos estados: nós não alteramos eles diretamente. Na verdade, utilizamos funções especiais para alterar o

estado. Quando falarmos sobre hooks explicaremos melhor como funcionam essas funções.

Dissemos que o React sempre “reage” às mudanças nos estados, mas ele também reage às mudanças nas propriedades. O diagrama abaixo explica isso melhor.

Figura 4 – Esquema de fluxo de informações em um componente.



Vamos entendê-lo melhor:

1. A parte debaixo do diagrama mostra o resultado na tela, ou seja, o que deveria ser mostrado com o nosso componente.
2. A parte de cima é o componente propriamente dito.
3. Repare que ele recebe como propriedade o label = “Usuário”, isso será mostrado no nosso componente.

4. Vamos criar um estado que chamaremos de valor. Essa variável será o valor mostrado no nosso edit (campo que digitaremos o texto). Inicialmente (quando o componente inicia), o valor estará vazio.

5. Conforme digitamos algum valor no campo, em algum evento relativo à mudança no campo que veremos mais à frente, modificamos o nosso estado "valor". O React, vendo que esse estado foi modificado, renderiza nosso componente novamente e vemos o novo valor do campo.

6. Se, por algum motivo, o valor da propriedade Label muda, nosso componente reage a isso e renderiza novamente.

Por fim, imagino que você deve estar se perguntando de que vale um componente com um valor que eu não tenho acesso. Na verdade, existe algumas formas de um componente se comunicar com o mundo ao seu redor. Veremos algumas durante esse curso, mas agora apresentarei a mais simples delas: **funções de callback**.

O que seria isso? Imagine que o nosso componente de campo receba como propriedade uma função. Essa função será mais ou menos assim (utilizando sintaxe de arrow functions):

```
const salvaUsuario = (valorCampo)=>{
// Fazer alguma coisa com o valor do campo
}
<Campo onType={ salvaUsuario}/>
```

Dentro do meu componente, em algum momento eu posso chamar essa função que recebi como propriedade.

Atenção! Um erro muito comum do programador ou da programadora iniciante:

Repare que a função, quando passada para a propriedade, **não** tem parênteses. Não estamos chamando a função, e simplesmente passando uma referência para ela.

Principais componentes do React Native

O React Native adiciona ao React dois tipos básicos de funcionalidades: visualização e acesso aos recursos do dispositivo.

Não seremos extensivos ao listar esses componentes, pois eles são vários. Iremos apenas listar o que consideramos os mais importantes. Essa lista foi difícil de fazer, a toda hora eu me lembrava de algum componente e resolvia mudá-la, provavelmente alguns faltaram. Para uma lista completa, entre no link: https://reactnative.dev/docs/components-and-apis.

Componentes de visualização:

- View: assemelha-se ao DIV do HTML, é o responsável por criar um bloco de visualização.
- Text: componente que mostra um texto. Atenção, diferente do HTML, não é possível deixar um texto solto sem que esteja na tag Text.
- Image: imagem em geral.
- TextInput: edit que permite incluir textos.
- StyleSheet: permite criar estilos parecidos (mas não iguais) ao CSS.
- Button: botão.
- FlatList: gera uma lista que pode ser vertical ou horizontal.
- Modal: cria conteúdo modal. Um modal é uma espécie de janela que aparece acima das outras, não permitindo que você acesse mais nada antes de sair dela.

Componentes de acesso ao dispositivo ou específicos para Android:

- expo permissions: esse, na verdade, não é um componente, e sim um módulo. Serve para gerenciar permissões de uma forma geral entre o app e o dispositivo. Nele, por exemplo, você pode saber se o seu app tem acesso à camera.

- ToastAndroid: toast é aquela mensagem que aparece e, logo depois, some. Ela normalmente é utilizada para informar algo simples, como por exemplo: “nova mensagem” ou “conectado com sucesso”.

Durante nossas aulas práticas conheceremos esses componentes e outros em mais detalhes.

Capítulo 3. Navegação no React Native

Um dos pontos fundamentais de quase todo app é poder navegar de uma tela para outra. No React Native, assim como no React, isso é feito com um conjunto de componentes externo. Aliás, isso é algo para irmos nos acostumando: quase tudo no mundo React são componentes externos ao framework. Isso também foi uma grande sacada da equipe desde o início, em vez de criar algo que seja bastante engessado, deram preferência a criar um framework que fazia o “simples” trabalho de renderizar e juntar componentes. O resto fica por conta da comunidade.

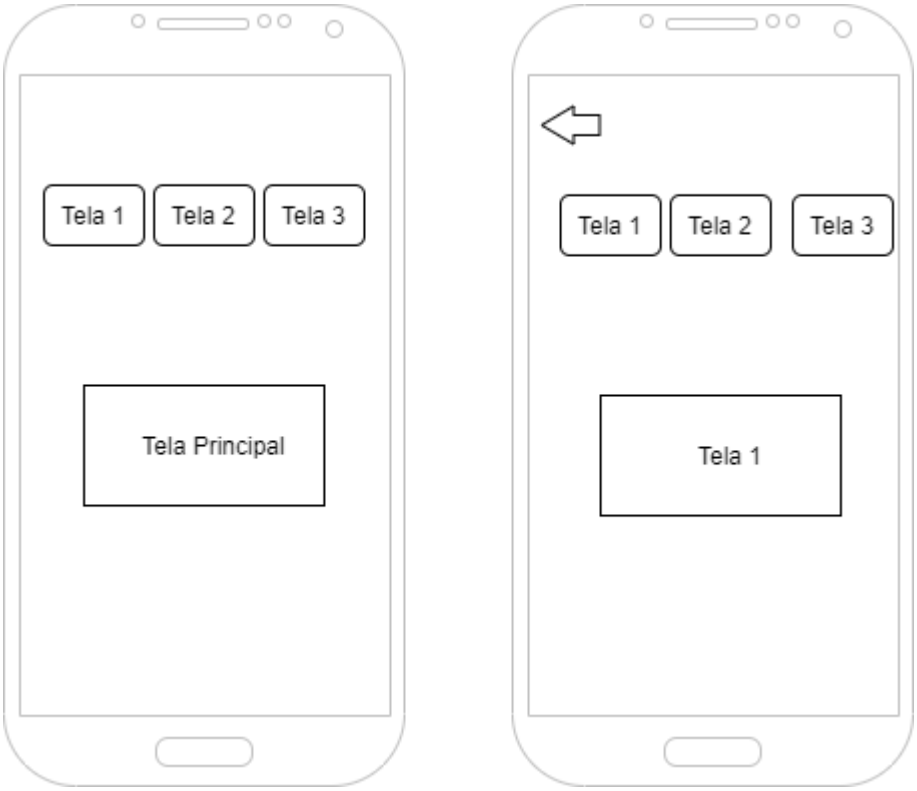
Como componente de navegação, a comunidade adotou o React Navigation. Acredito que seja possível explicar o React Routing explicando apenas os seguintes componentes:

- **NavigationContainer**: esse componente é o contêiner de tudo que precisa ser navegado. Ou seja, praticamente tudo dentro da sua aplicação estará “embaixo” dele. As diversas formas de navegação que veremos abaixo devem ser colocadas dentro do navigation conteiner, é ele que gerenciará o estado da sua navegação. Nele também é possível criar funções que são chamadas sempre que alguma navegação na sua aplicação foi chamada. Isso tem várias utilidades, por exemplo: você pode querer salvar no seu servidor quais as telas seu usuário vai mais (famoso analytics). Seu navigator ficaria mais ou menos assim:

```
<NavigationContainer
  onStateChange={(state) => /* aqui você envia para a sua API que uma tela foi chamada, por exemplo*/}
>
  {/* Aqui você coloca a chamada para todas as telas "navegaveis" do sistema */}
</NavigationContainer>
```

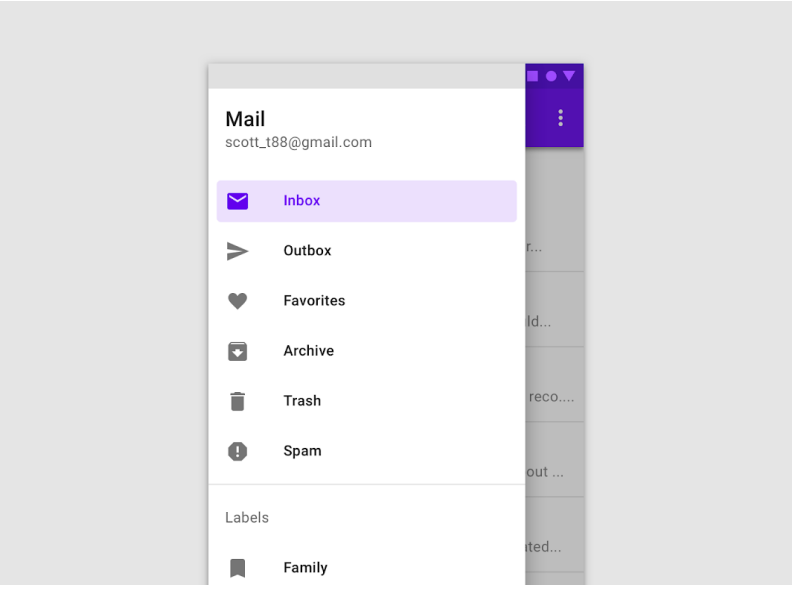
- **StackNavigator:** cria a navegação em forma de pilha. A figura abaixo tenta mostrar um croqui do que seria esse tipo de navegação.

Figura 5 – Croqui de Stack Navigator.



- **DrawerNavigator:** faz a navegação com aquele menu à esquerda que você puxa. A figura abaixo deve explicar melhor como essa navegação é feita.

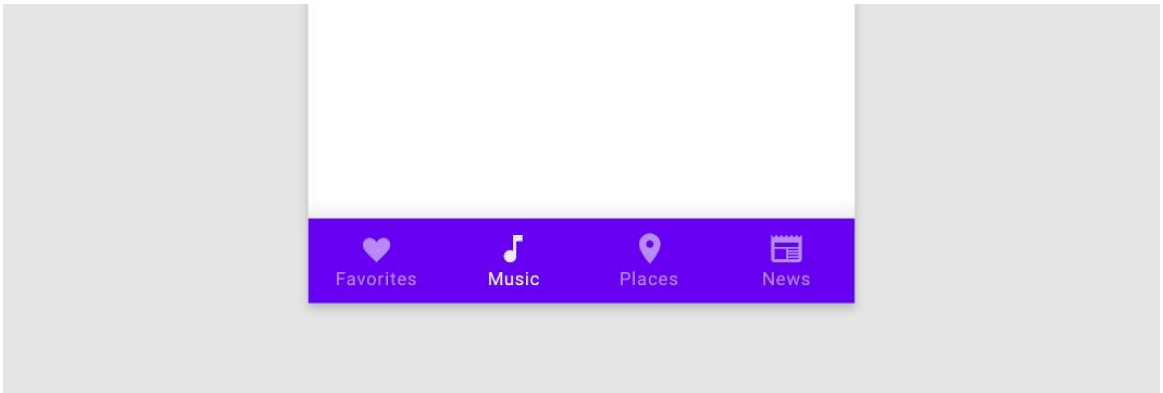
Figura 6 – Drawer Navigator.



Fonte: Site Material UI.

- **TabNavigator, MaterialTopTabNavigator e MaterialBottomTabNavigator:** são as navegações feitas por aquelas barrinhas embaixo da tela, com alguns botões que permitem a navegação. A MaterialBottomTabNavigator é a tab do material Design. Vamos falar de Material Design na nossa aula interativa, mas só para adiantar, é o padrão de design e experiência do usuário criado pelo Google, para Android e sites. Por essa última frase você consegue depreender que esse último componente é mais a cara do Android:

Figura 7 – Drawer Navigator.



Fonte: Site Material UI.

Capítulo 4. Estilo

Esse foi outro ponto em que a equipe do React Native acertou em cheio. Lembre-se, o lema do React Native é “Learn once, write anywhere”, ou seja, se você já está acostumado a programar em React para Web, programar em React Native é bastante parecido.

Antes de mais nada, o que é estilo? Estilo é a forma como sua app se comportará em relação ao espaçamento entre os componentes visuais, cores, bordas etc.

E como o React Native trata estilos? Basicamente dois conceitos são importantes aqui:

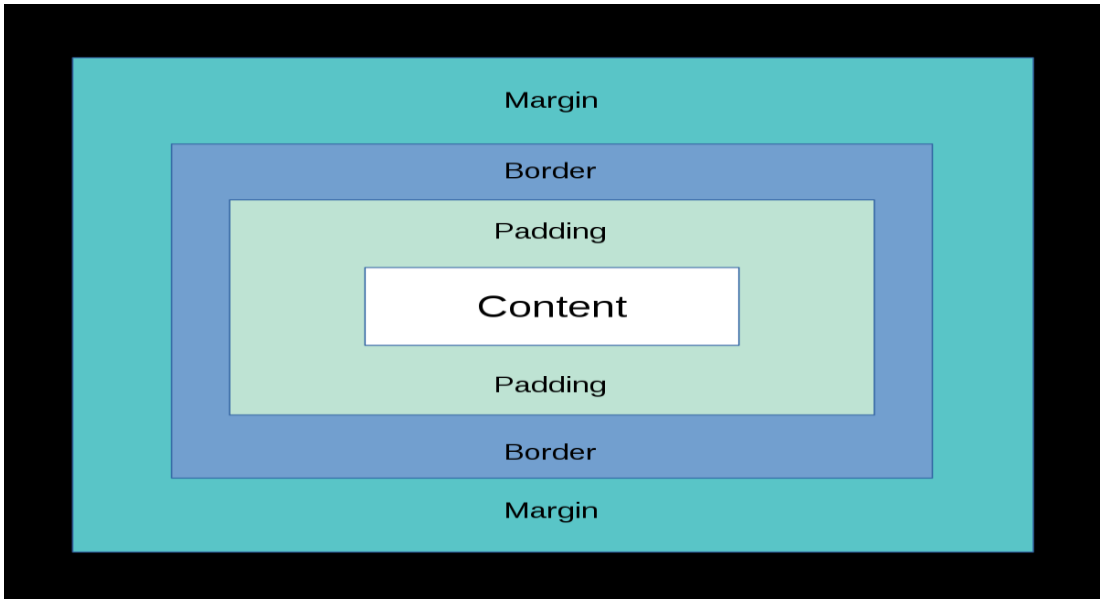
- Box Model.
- Flex Box.

E o que são esses dois?

Box Model e o modelo de separação entre componentes. Por exemplo, entre dois Text devemos ter separação, mas como ela é feita?

O Box Model diz que temos conteúdo, margin, padding e border. Eles se unem em caixas que ficam uma dentro da outra conforme mostramos na figura abaixo:

Figura 8 – Drawer Navigator.



Fonte: https://commons.wikimedia.org/wiki/File:Box-model.svg.

Em Content nós temos o conteúdo que vamos apresentar (no nosso exemplo, o texto). Os diversos textos são separados por padding, border e margin. A figura não mostra isso, mas podemos definir valores para margin esquerda, direita superior e inferior. O mesmo vale para os outros.

Mas qual a diferença entre eles? Bom, a única diferença importante de destacar é que a margem (margin) colapsa entre dois componentes que estão juntos.

Como um exemplo prático, imagine dois componentes que estão um ao lado do outro. A margem direita do primeiro componente é de 5 e a margem esquerda do segundo componente também é de 5. A margem total de separação entre esses dois componentes é de 5 pois eles irão “colapsar”, ou seja, juntar as margens. Já se incluirmos a esses dois componentes um padding de 5 (direito no primeiro e esquerdo no segundo), teremos uma margem total de 5 e um padding de 10, pois eles se somam e não ficam um sobre o outro.

O próximo passo para aprender a criar estilos é entender o flex layout.

Antes de começarmos, gostaria de que você tivesse em mente o problema que eles quiseram resolver quando criaram o flex layout.

Imagine que temos vários dispositivos móveis com tamanhos de tela diferentes. Não seria possível, por exemplo, dizer que o cabeçalho da sua app teria um espaço de 20 pixels. Isso seria completamente diferente em um celular com tela grande e um com tela pequena. A solução foi criar um modelo que permita que o tamanho e distância entre os componentes fosse realizado em proporções, e não em tamanhos absolutos.

Na verdade, não criaram nada do zero. O modelo foi copiado do flex layout do CSS. Vamos entender então o flex layout. A primeira coisa que precisamos escolher no nosso flex layout é a sua orientação ou o seu eixo principal: linha ou coluna. O padrão do React Native é coluna. Depois, precisamos escolher como será o afastamento entre os diversos componentes, ou seja, como eles serão distribuídos no espaço em que temos. Para esse afastamento, temos duas propriedades: justifyContent e alignItems. Elas vão funcionar de forma inversa:

Para a orientação row, temos:

- justifyContent no eixo principal e alignItems no outro eixo.

Para a orientação column, temos:

- alignItens para o eixo prinipal e justifyContent no outro eixo.

Agora, quais valores essas propriedades podem receber? Abaixo colocamos esses valores:

- JustifyContent:
  - Flex-start: os componentes ficarão sempre no início do eixo.
  - Flex-end: os componentes ficarão sempre no final do eixo.
  - Center: os componentes ficarão centralizados sem espaçamento.
  - Space-between: haverá espaçamento entre os componentes mas não nas bordas.

  - Space-around: haverá espaçamento entre os componentes e nas bordas.

  - Space-evenly: mesmo que space-around, mas os espaçamentos das bordas serão iguais aos espaçamentos dos componentes.

- AlignItems:

  - Flex-start, flex-end e center: iguais ao justifyContent.

  - Stretch: tomarão todo o espaço do seu eixo.

  - Baseline: ficarão no início da linha do texto.

Capítulo 5. Gerenciamento de estado global

Nós já sabemos como gerenciar estados dentro dos componentes React. No entanto, e se quisermos que dois componentes possam ter acesso a um estado único, global?

Vamos usar como exemplo a figura abaixo:

Figura 9 – Drawer Navigator.

Imagine que nós queiramos que o botão salvar apenas se habilite caso haja informação cadastrada para usuário e senha. Há duas formas de fazermos o gerenciamento deste estado. Na verdade, há várias, mas duas são mais comuns: Redux e Context API.

Para sabermos se devemos utilizar uma forma ou outra, devemos levar em consideração o tamanho da nossa App. Redux é um framework mais completo e complexo, sendo mais adequado para Apps maiores, enquanto o Contex Api é a ferramenta ideal para apps menonores. Conheceremos o Redux em uma das nossas aulas interativas, mas nessa apostila vamos nos focar na Context API.

O funcionamento da Context API é bem simples. Ele segue a ideia de produtor e consumidor. Basicamente, você deverá envolver toda parte da app que utilizará o contexto em uma tag, que chamamos de provider (produtor). Seria algo mais ou menos assim:

```
<ContextoGlobal.provider value={estado}>
   ... toda a minha app
</ContextoGlobal.provider>
```

Dessa forma, em cada componente da sua app (que estiver dentro do ContextoGlobal), será possível chamar o hook useContext e ter acesso ao contexto global. E como fazemos para salvar uma informação a esse contexto? Utilizamos outro hook, o useReducer.

Através do useReducer, é possível criar um estado da aplicação e usar uma função redutora. Essa função é chamada de redutora pois recebe o estado e uma ação a ser executada e modifica o valor do estado original.

Você pode utilizar o useReducer em conjunto com o provider para gerenciar o estado global. Por exemplo, se queremos gerenciar um estado da nossa tela, dizendo se há ou não valor nos campos de usuário e senha, podemos fazer o seguinte:

- Teremos em nosso estado global (nesse caso ele é global à nossa tela apenas) dois booleanos, dizendo se há usuário e se há senha.
- Quando houver modificação nos campos chamados, nossa função redutora passando a ação (por exemplo, se foi digitado algum valor no campo usuário, passamos a ação “usuárioDigitado”; se for apagado o usuário, passamos a ação “usuárioApagado”).
- Dentro da nossa função redutora, fazermos a lógica para ligar (true) ou desligar (false) nossas variáveis.

Sei que parece bastante abstrata a ideia ainda. No entanto, pode ficar tranquilo, quando fizermos nosso primeiro código de exemplo ficará bastante claro o que será feito e como funcionará.

Referências

EXPO IO. *Home*. 2020. Disponível em <https://expo.io>. Acesso em: 11 set. 2020.

FACEBOOK INC. *React Native*. 2020. Disponível em: <https://reactnative.dev/>. Acesso em: 11 set. 2020.

FACEBOOK INC. *React*. 2020. Disponível em: <https://reactjs.org/>. Acesso em: 11 set. 2020.

REACT NAVIGATION. *Home*. Disponível em: <https://reactnavigation.org/>. Acesso em: 11 set. 2020.

Marcas registradas

IOS é uma marca registrada de Apple.

Android é uma Marca Registrada de Google LLC.

Expo e Expo.io são uma Marca Registrada de Expo inc.

NodeJs é uma marca registrada de OpenJS.

React e React Native são marcas registradas de Facebook.

Para saber como utilizar as marcas registradas, entre em:

- https://developer.android.com/distribute/marketing-tools/brand-guidelines
- https://expo.io/terms
- https://nodejs.org/en/about/trademark/
- https://opensource.facebook.com/legal/privacy
- https://www.apple.com/legal/intellectual-property/guidelinesfor3rdparties.html