Hybrid Mobile App Development

Navegação

Navegação

Páginas

Até agora nós aprendemos diversos componentes e práticas de React Native. Mas algo básico em aplicativos, ainda não exploramos, que é a navegação.

Navegação

Ferramenta

Para a navegação, poderíamos utilizar uma estratégia similar ao HTML puro, mas já existem diversas bibliotecas que fornecem para nós uma alternativa mais simples e mais desenvolvida. No nosso caso utilizaremos o react-navigation

Navegação

React Navigation

```
npm install @react-navigation/native
```

```
npm install react-native-screens react-native-safe-area-context
```

```
npm install @react-navigation/native-stack
```

React Navigation

O que recebemos com o react-navigation:

- Navegação
- Stack navigation
- Nested navigation
- Header
- Tab navigation
- Drawer Navigation
- Modal screens

React Navigation

E como iniciar?

Precisamos declarar nossas routes, ou seja, os endereços e suas páginas respectivas. Essa etapa pode ser bem simples em apps menores, mas quando o app cresce em tamanho e complexidade, e temos nested navigation, pode se tornar complexa, por isso desde o início é preciso ter organização e cuidado nessa etapa

React Navigation

E como iniciar?

```
const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

React Navigation

Container

O Container é o nosso Provider. Ele deve encapsular todos os componentes da navegação. Ele é o responsável pelo estado da navegação. Raramente usaremos ele com alguma prop, ou precisaremos de sua referência. O caso mais comum será passar um initialState, para modificarmos a tela inicial do nosso app em casos especiais, como persistência de estado

React Navigation

Navigator

O Navigator é responsável por definir o tipo de navegação que teremos. Temos a opção de drawer, tab ou stack (o mais comum). Ele decide como nossas telas serão renderizadas. Podemos ter múltiplos Navigators, inclusive cascateados (nested navigation). Iremos adentrar mais afundo no comportamento deles neste caso.

React Navigation

Screen

Aqui é onde nós definimos a nossa tela de fato. É obrigatório passarmos um name (único) e um component, que é a página que iremos renderizar ao acessar essa rota específica.

React Navigation

Como navegar entre telas?

Nossa tela inicial será a primeira tela declarada, ou a tela passada na prop initialRouteName ao nosso navigator.

Cada tela declarada terá acesso a uma prop especial, o **navigation**, que nos permite fazer o controle do fluxo de telas

React Navigation

Agora podemos entender melhor

```
const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Provider e HOC Pattern

Como damos acesso à navegação a todas as nossas páginas?

Como vimos, temos acesso a uma prop especial. Mas como isso ocorre?
Aqui utilizamos um outro padrão de arquitetura, que é o provider. Ao encapsularmos nosso componente (que pode ser o App inteiro) com um Provider, ele terá acesso ao Context desse provider. E é dessa maneira que todas as telas tem acesso ao mesmo estado de navegação

Provider e HOC Pattern

Mas sempre é usado HOC?

Nesse caso, o navigation utiliza HOC para fornecer a todas as telas acesso ao navigation do seu Navigator, e somente dele (nós podemos ter múltiplos navigators).
Mas um outro padrão comum é a utilização de hooks ao invés de HOC. Nesse caso, teriamos acesso a um hook especial, que nos daria o navigation através de seu uso.
Provider é necessário, a HOC opcional.

Provider e HOC Pattern

Mas sempre é usado HOC?

Em casos que não utilizamos functional components, nós não temos acesso aos hooks, e nesse caso, um HOC é necessário.

Ex: A library react-intl fornece um objeto especial para traduzir as strings, que é o objeto Intl. Podemos usar o hook useInlt() para acessá-lo, ou então encapsular nosso componente com a HOC withIntl()

React Navigation

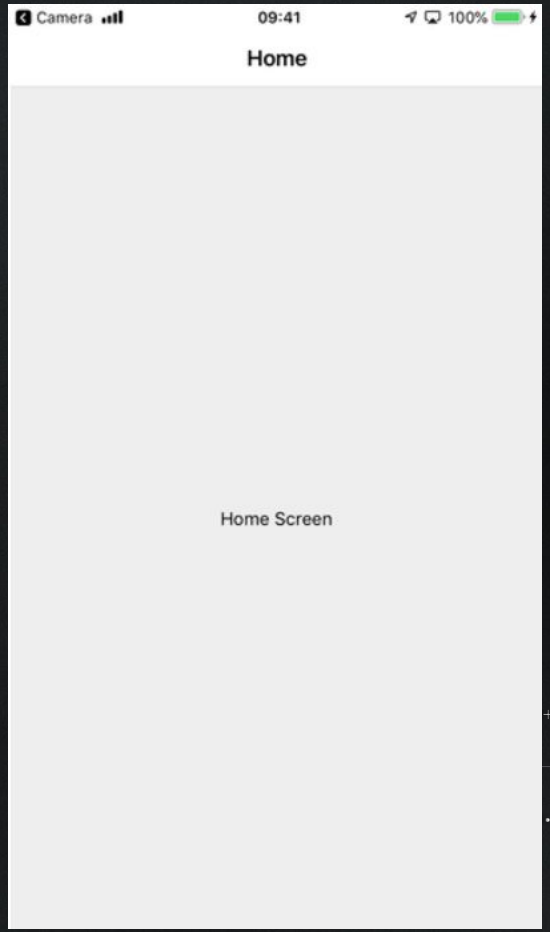
E como fica nosso exemplo?

```
const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

React Navigation

E como fica nosso exemplo?



React Navigation

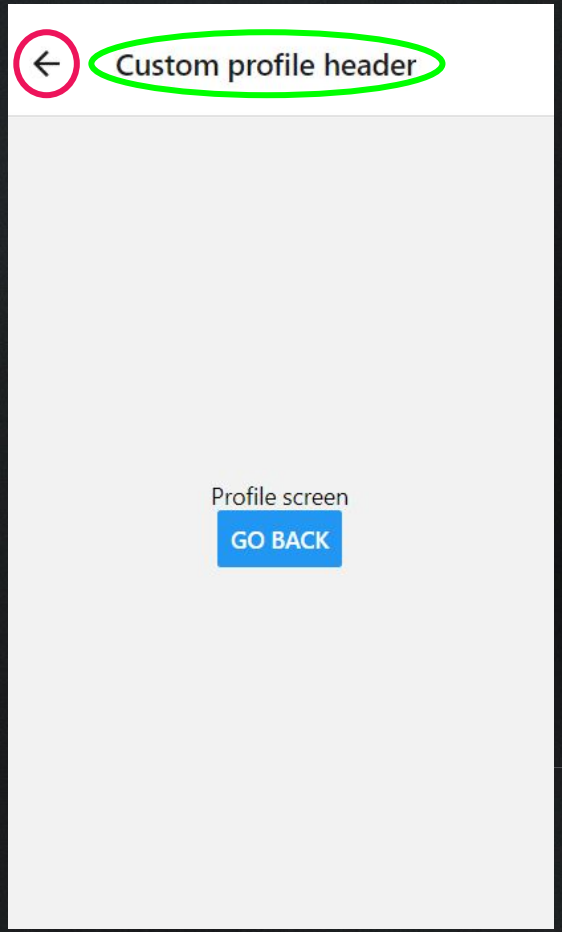
Header

Podemos ver que o React Navigation já nos fornece o header por padrão. Que bacana da parte dele!!
Porém é um header bem simples, um retângulo branco, com o nome da tela, e um backButton (quando não for a tela inicial)

React Navigation

E uma página que não é a inicial?

Vemos o back button e o title configurados para nós. O back button é o equivalente a chamar navigation.goBack() Por isso não temos ele acessível na nossa tela inicial



React Navigation

Navegação

Já vimos que o nosso Navigator nos permite acesso ao objeto especial **navigation**. Nele temos acesso a 3 principais métodos para a navegação:

- navigation.navigate(‘screen_name’)
- navigation.goBack()
- navigation.setParams({})

React Navigation

Navegação

```
function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}
```

React Navigation

Navegação

```
function DetailsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.navigate('Details')}
      />
      <Button
        title="Go Home"
        onPress={() => navigation.navigate('Home')}
      />
    </View>
  );
```

React Navigation

Navegação

```
const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

React Navigation

Header

O Header é fornecido de maneira simples para nós, mas podemos customizá-lo ou escondê-lo se assim quisermos
Para isso, usaremos uma Prop especial headerStyle, que pode ser usada na screen, ou no Navigator, afetando todas as Screens

React Navigation

Header

O Title e o BackButton podem ser substituídos por componentes personalizados, assim, nosso header é bem flexível
É possível também criar um botão adicional na direita, se assim quisermos

React Navigation

Header Title

Podemos mudar o Title, passando uma string, ou então, um componente para ser utilizado no lugar dele

React Navigation

Header Title utilizando uma string

```
<Stack.Screen
  name="Home"
  component={HomeScreen}
  options={{ title: 'My home' }}
/>
```

React Navigation

Header Title utilizando um componente

```
const HeaderTitle = () => {
    return (
        <View>
            <Text style={Style.text}>
                {"TITLE!!"}
            </Text>
        </View>
    )
}
```

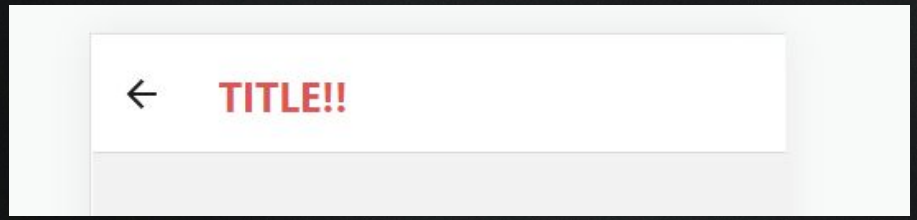
```
const Style = StyleSheet.create({
    text: {
        fontSize: 22,
        fontWeight: 'bold',
        color: "#d55",
    }
})

export default HeaderTitle;
```

React Navigation

Header Title utilizando um componente

```
<Stack.Screen
    name="Details" component={Details}
    options={{
        headerTitle: HeaderTitle,
    }}
/>
```



React Navigation

Podemos passar o route e navigation

```
const HeaderTitle = ({route}) => {
    return (
        <View>
            <Text style={Style.text}>
                {`!!${route.name}!!`}
            </Text>
        </View>
    )
}
```

```
<Stack.Screen
    name="Details" component={Details}
    options={({ route }) => ({
        headerTitle: () => <HeaderTitle route={route} />,
    })}

/>
```

React Navigation

E se quisermos nosso próprio header?

```
const TabHeader = ({route, navigation}) => {
    return (
        <View style={Style.headerContainer}>
            <Button title={"Home"} color={'#66d'} onPress={() => navigation.navigate('Home')}/>
            <Text style={Style.headerTitle}>
                {`${route.name}`}
            </Text>
            <Button title={"Profile"} color={'#66d'} onPress={() => navigation.navigate('Profile')}/>
        </View>
    )
}
```

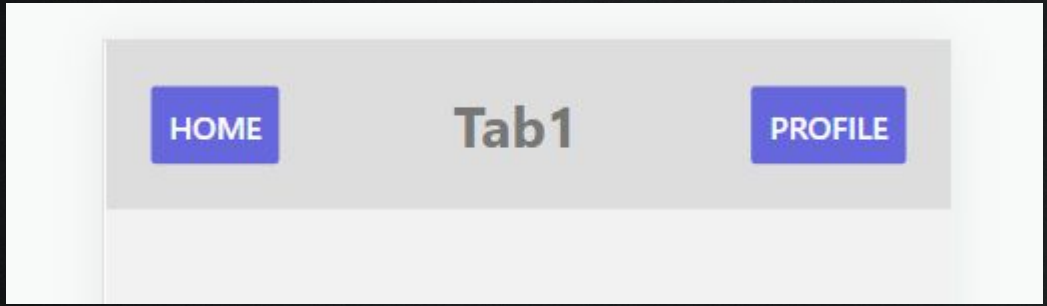
React Navigation

E se quisermos nosso próprio header?

```
const TabGroup = () => {
    return (
        <Tab.Navigator screenOptions={{ headerShown: true, header: TabHeader }}>
            <Tab.Screen name="Tab1" component={Tab1} />
            <Tab.Screen name="Tab2" component={Tab2} />
            <Tab.Screen name="Tab3" component={Tab3} />
        </Tab.Navigator>
    )
}
```

React Navigation

E se quisermos nosso próprio header?



React Navigation

E quais as opções para aplicar?

- Na Screen, usando a prop options
- No Navigator, usando screenOptions
- Num Group, usando screenOptions

React Navigation

Group no navigation

Para facilitar a aplicação dessas opções e personalizações, temos a opção de além de aplicar no Navigator, ou diretamente no screen, criar Groups específicos, e aplicar nesse group

React Navigation

Group no navigation

```
<Stack.Navigator>
  <Stack.Group
    screenOptions={{ headerStyle: { backgroundColor: 'papayawhip' } }}
  >
    <Stack.Screen name="Home" component={HomeScreen} />
    <Stack.Screen name="Profile" component={ProfileScreen} />
  </Stack.Group>
  <Stack.Group screenOptions={{ presentation: 'modal' }}>
    <Stack.Screen name="Search" component={SearchScreen} />
    <Stack.Screen name="Share" component={ShareScreen} />
  </Stack.Group>
</Stack.Navigator>
```

React Navigation

E para usar o navigation fora de uma screen?

Temos a opção de usar o hook useNavigation (lembra do provider pattern?)

```
function GoToButton({ screenName }) {
  const navigation = useNavigation();

  return (
    <Button
      title={`Go to ${screenName}`}
      onPress={() => navigation.navigate(screenName)}
    />
  );
}
```

Extra: PropTypes

prop-types

O prop-types é uma biblioteca que nos ajuda com a verificação e validação da tipagem das props dos componentes.

Extra: PropTypes

E por que validar os tipos das props?

Conforme nosso projeto cresce em tamanho e complexidade, os componentes ficam mais complicados de entender, e nem sempre é claro quais props e quais seus tipos nos componentes. A validação serve como uma documentação viva, além de permitir uma maneira clara de definir um valor default quando uma prop não for obrigatória.

Extra: PropTypes

Ex:

```
myComponent.propTypes = {
    isActive: PropTypes.bool,
    name: PropTypes.string,
    address: PropTypes.shape({
        street: PropTypes.string,
        number: PropTypes.number,
        complement: PropTypes.string,
        zipcode: PropTypes.number,
    }),
    userId: PropTypes.number.isRequired,
    callback: PropTypes.func,
    age: PropTypes.oneOf(['baby', 'child', 'adult', 'elderly']),
}
```

Dúvidas, anseios, desabafos?


FIAP