- O **`StatusBar`** (componente do React Native – importado em **`App.js`**) serve para ***controlar a barra de status do aplicativo***.
  ➜ É a **zona** – geralmente presente na **parte superior da tela** – que exibe a **hora atual**, **Wi-Fi** e **informações de rede celular**, **nível de bateria** e/ou **outros ícones de status**.
  ➜ Mais em: documentação da StatusBar.
- O **`ActivityIndicator`** é um **componente nativo do React Native**, responsável pelo efeito de ***loading***.
- O **`activeOpacity`** é uma propriedade que ***aumenta/diminui a opacidade do botão***.
- O **`KeyboardAvoidingView`** garante que, ao clicarmos no **`input`**, a nossa ***interface*** seja **jogada *para cima***, fazendo com que tenhamos **acesso aos demais componentes mesmo com a presença do teclado**.
  ➜ Para garantir isso no ***iOS***, devemos **importar**, **do React Native**, o **componente** ***`Platform`***.
  ➡ Usaremos a propriedade **`behavior`**, que receberá uma ***condição a partir do SO usado***. Para ativar passamos ***`enabled`***.
- O **`headerBackTitleVisible`** só é **aplicado no *iOS***.
  ➜ Ao **lado da flecha de voltar**, existe um **título** chamado de **backTitle**, que significa "**voltar**" (ou "**back**", se o SO estiver em inglês).
  ➡ Com **`headerBackTitleVisible`**, este título é ***removido***.
- O **`context`** nos fornece uma maneira de **passar dados pela árvore de componentes sem ter que passar `props` manualmente**.
  ➜ Naturalmente, os dados são passados de uma forma ***top-down*** – ou seja, **de pai para filho** –, mas este uso fica inviável para certos tipos de props – principalmente quando exigidos por muitos componentes em uma aplicação.
  ➡ Em outras palavras: nos ajuda a ***acessar dados globalmente em uma aplicação sem ter que passar uma props explicitamente***.
- O **`createContext`** serve para ***criar um contexto***.
  ➜ Para o **`createContext`** precisamos passar um **valor inicial**.
- O **Provider** é quem vai ***prover todas as informações para todas as telas***.
  ➜ As telas ficarão **dentro do Provider**, tendo, desta forma, acesso à todas as informações.
  ➜ Para o **`<AuthContext.Provider></AuthContext.Provider>`**, precisamos passar a propriedade ***`value`***, que é o **valor cujo qual queremos que os children** (toda a nossa interface) **tenham acesso**.
  ➜ Podemos passar os **filhos (*children*) ao Provider**.
  ```
  export function AuthProvider({children}) {
    ...
  }
  ```
- O React Native nos permite fazer apenas **requisições que são *HTTPS***.
  ➜ Quando não é HTTPS – como é o meu caso (estou fazendo uma requisição do localhost) – usamos o ***IP da máquina***, principalmente quando usamos o localhost.
  ➡ Para saber o **IP da máquina**: executar o comando **`ipconfig`** no terminal e pegar o **IPv4**.
- O **`useIsFocused`** é usado para **garantir** que estamos com o **foco na *tela atual***.
- O **`TouchableWithoutFeedback` fecha o teclado** ao clicarmos **fora do input**.
- O **parseInt()** converte uma ***string*** em um **número *inteiro***, enquanto o **parseFloat()** converte uma ***string*** em um **número *float***.
- O **`Alert`** (importado do ***React Native***) cria uma **caixa de diálogo de alerta** com um ***título***, uma ***mensagem*** e, opcionalmente, ***botões***.
- Na função de registro ("***`src/pages/NewRegister/index.js`***"), ***não*** precisamos passar o token porque o **`useEffect`** (presente em "***`src/contexts/auth.js`***") já nos **fornece o token ao carregar o app**.
- O **`numberOfLines`** garante que o **texto** fique em uma **determinada quantidade de linhas**.
- O **`DrawerItem`** funciona como se fosse uma **página/item específico**.
- O **`LocaleConfig`** serve para **configurar o calendário**. Podemos configurar no padrão brasileiro, por exemplo.