Principais hooks do React Native

O React possui uma série de ***hooks*** que podem ser utilizados para controlar o estado, ciclo de vida e efeitos colaterais dos componentes.

pedro.mateus@unicesumar.edu.br

- ***useState***: Hook que permite adicionar estado a um componente.

Retorna um array com dois elementos, o primeiro é o estado atual e o segundo é uma função que permite atualizar esse estado.

Toda vez que esse estado for atualizado, o componente é renderizado novamente.

```
const [count, setCount] = React.useState(0)
```

Exemplo de uso do useState
Documentação do React sobre useState

- ***useEffect***: Hook que permite realizar efeitos colaterais em componentes.

Aceita uma função que será executada toda vez que o componente for renderizado, e um array de dependências, que determina quando a função será executada novamente. Pode ser utilizado para buscas em APIs, atualizar a DOM em resposta a eventos e também adicionar ou remover eventos.

```
React.useEffect(() => {
  console.log('Componente renderizado')
}, [])
```

Exemplo de uso do useEffect

Documentação do React sobre useEffect

- ***useContext***: Hook que permite acessar um contexto, dentro de componentes. Permite que se subscreva a mudanças e atualize o contexto.

```
// Cria context
const Context = React.createContext(null)

// Engloba a context em um componente pai
<Context.Provider value={value}>
  <Componente />
</Context.Provider>

// Todo componente filho pode acessar o contexto
const value = React.useContext(Context)
```

Exemplo de uso do useContext

Documentação do React sobre useContext

Post do Kent C. Dodds sobre boas práticas com useContext

- ***useReducer***: Hook que permite adicionar e controlar um estado complexo a um componente funcional.

Centraliza a lógica em uma função específica e pura, que recebe o estado atual e uma ação, e retorna o novo estado.

```
const initialState = { count: 0 }
const reducer = (state, action) => {
  // Lógica para atualizar o estado
}

// Utiliza o useReducer em um componente
const [state, dispatch] = React.useReducer(reducer, initialState)
```

Exemplo de uso do useReducer

Documentação do React sobre useReducer

- ***useCallback***: Hook que permite memorizar funções.

Retorna uma versão memorizada da função, que só muda se uma das dependências mudar.

```
const memoizedCallback = React.useCallback(() => {
  doSomething(a, b)
}, [a, b])
```

Exemplo de uso do useCallback

Documentação do React sobre useCallback

- ***useMemo***: Hook que permite memorizar valores.

Muito semelhante ao useCallback. Retorna um valor memorizado, que só muda se uma das dependências mudar.

```
const memoizedValue = React.useMemo(() => computeExpensiveValue(a, b), [a, b])
```

Exemplo de uso do useMemo

Documentação do React sobre useMemo

Estilização

O React Native utiliza um subconjunto de propriedades CSS para estilização.

Os estilos são criados utilizando JavaScript, através de objetos que contém propriedades e valores.

Os nomes das propriedades são escritos em **camelCase** e os valores são escritos como strings.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
})

export default function App() {
  return (
    <View style={styles.container}>
      <Text style={{ color: 'red' }}>Hello World!</Text>
    </View>
  )
}
```

Documentação do React Native sobre estilização

> Modelo de layout flexbox

Flexbox é um modelo de layout que permite o design de layout responsivo e previsível, sem depender de tamanhos específicos de tela.

O React Native implementa o layout Flexbox de uma maneira mais simplificada.

Algumas das principais propriedades são:

1. `flex`:

A propriedade flex determina como o componente vai crescer ou encolher para preencher o espaço disponível do container.
Aceita um valor numérico que representa o fator flex.

Por exemplo, um componente com **flex: 2** vai ocupar o dobro do espaço de um componente com **flex: 1**.

2. `flexDirection`:

A propriedade **flexDirection** determina a direção principal do layout, se é uma coluna ou uma linha. Aceita os valores **row**, **row-reverse**, **column** e **column-reverse**.

Por exemplo, **row** coloca os itens da esquerda para a direita e **column** coloca os itens de cima para baixo.

3. `justifyContent`:

A propriedade **justifyContent** alinha os itens no eixo principal. Aceita os valores **flex-start**, **center**, **flex-end**, **space-around**, **space-between** e **space-evenly**.

Por exemplo, **justifyContent: flex-start** alinha os itens no início do container e **flex-end** alinha os itens no final do container.

4. `alignItems`:

A propriedade **alignItems** alinha os itens no eixo secundário. Aceita os valores **flex-start**, **center**, **flex-end**, **stretch** e **baseline**.

Por exemplo, **alignItems: center** alinha os itens no contro do eixo secundário do container.

5. `alignSelf`:

A propriedade **alignSelf** sobrescreve a propriedade **alignItems** do container para um único item. Aceita os valores **auto**, **flex-start**, **center**, **flex-end**, **stretch** e **baseline**.

Por exemplo, **alignSelf: flex-end** alinha um item no final do container.

Documentaçao do React Native sobre Flexbox
CSS Tricks Guia completo de Flexbox

Conectando com APIS (Network)

A maneira mais simples de realizar requisições HTTP em React Native é utilizando a API ***fetch***.

A função ***fetch*** é nativa do JavaScript, e é utilizada para realizar requisições HTTP, como GET, POST, PUT, DELETE, etc.

Por ser uma operação assíncrona, a função ***fetch*** retorna uma ***Promise***, que pode ser tratada com ***then*** e ***catch***.

```
fetch('https://api.github.com/users/satinp')
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error(error))
```

É importante tratar os erros, pois a função ***fetch*** pode retornar um erro caso a requisição falhe.

Também é possível utilizar a função ***async/await*** para realizar requisições HTTP, que é uma maneira mais limpa e legível de lidar com operações assíncronas.

```
const fetchData = async () => {
  try {
    const response = await fetch('https://api.github.com/users/satinp')
    const data = await response.json()
    console.log(data)
  } catch (error) {
    console.error(error)
  }
}
```