Mobile App Development

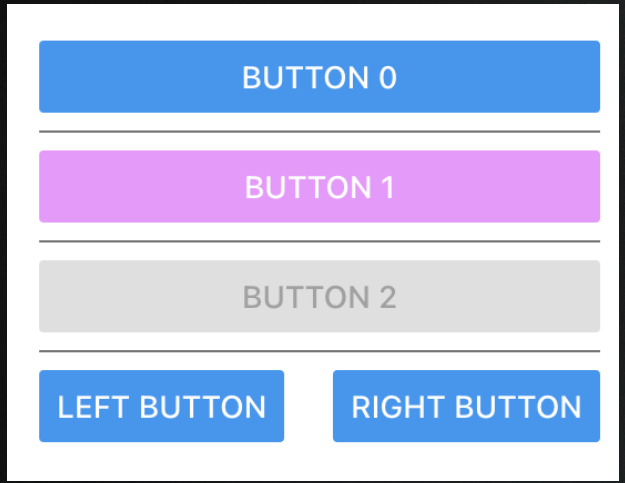
Dynamic Render, Button, mas agora melhor :)

Revisando

Button

Aprendemos a usar o componente Button, que é muito simples e rápido de implementar. Mas já vimos que ele vem com diversas limitações quanto ao estilo, e não queremos que nossos aplicativos tenham sempre os mesmos botões, certo?

Button



"Você só sabe fazer botão igual? Vou levar o app pro meu sobrinho, ele fez um curso de excel e vai saber fazer melhor"

- Cliente chata após ver seus botões todos iguais

Revisando

“Chegará o momento em que o Button não mais será suficiente para vossa necessidade, neste dia, vós deveis buscar um componente customizável”

- Prof. Mateus (Prof. = Profeta)

Pressable

Solução: Pressable

Pressable é um container (como nossa View), que possui callbacks de toque para transformá-lo em um botão, caso seja necessário

```
<Pressable onPress={onPressFunction}>
  <Text>I'm pressable!</Text>
</Pressable>
```

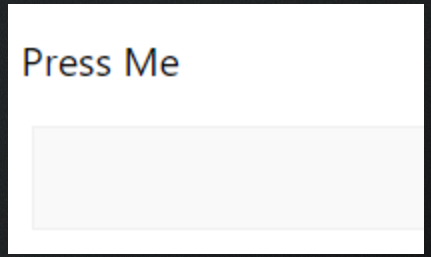
Pressable

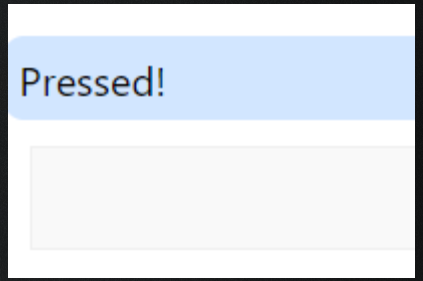
E como estilizá-lo?

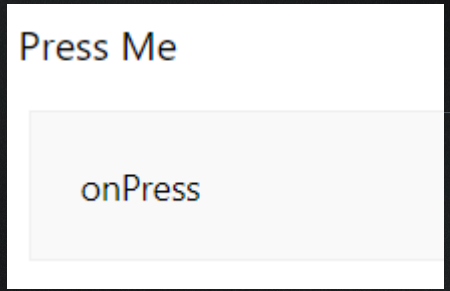
Os componentes filho desse componente recebem uma prop especial chamada pressed, que é responsável por avisar se ele está pressionado ou não. Usando ela, podemos criar um feedback visual ao usuário, quando ele é apertado

Pressable

```
<View style={styles.container}>
  <Pressable
    onPress={() => {
      setTimesPressed(current => current + 1);
    }}
    style={({pressed}) => [
      {
        backgroundColor: pressed ? 'rgb(210, 230, 255)' : 'white',
      },
      styles.wrapperCustom,
    ]}>
    {({pressed}) => (
      <Text style={styles.text}>
        {pressed ? 'Pressed!' : 'Press Me'}
      </Text>
    )}
  </Pressable>
```







Pressable

props importantes:

- disable
- onPress
- onLongPress
- style
- children

Higher-Order Component

Parentesis importante:

Para entender melhor como funciona o pressable, precisamos entender o que é um Higher-Order Component (HOC para os íntimos)
Muitas vezes, queremos adicionar um comportamento nos componentes filhos de um pai, e é nesse caso que usamos o HOC

Higher-Order Component

Problema:

Imagina que temos um Context que é utilizado em basicamente todos os nossos componentes. Talvez um Context com informações do nosso usuário… Não queremos replicar a chamada do Context em todos os locais. E mais ainda, queremos a facilidade de, caso necessário, a mudança seja feita em um local só.

Higher-Order Component

Solução:

Criar um Higher-Order Component que irá “injetar” essa informação nos nossos componentes.

Higher-Order Component

Solução:

```
const withUserContext = (Component) => (props) => {
    const userContext = React.useContext(UserContext);
    return (
        <Component { ...props, useContext }/>
    );
}

const myComponent = () => {
    <Text>
        Eu sou um componente sem context
    </Text>
};

export default withUserContext(myComponent);
```

Instanciação Dinâmica

Como melhorar?

Na última aula, vimos que podemos fazer a instanciação dinâmica de componentes usando Array.map(). Mas, será que podemos utilizar uma técnica melhor que essa para alguns casos?

Instanciação Dinâmica

Problema

Você recebeu um projeto de fazer um aplicativo para o cardápio da pizzaria “Ki Fome Rapaz”. Ela é famosa por ter a maior variedade de sabores da região.
Atualmente ela possui 256 sabores, e quer um aplicativo semelhante ao ifood, onde apareça todos os sabores e o cliente possa clicar para escolher

Instanciação Dinâmica

Problema

Além disso, o gerente da pizzaria é um visionário, e acredita que a foto da pizza não é chamativa, e exige que ao invés de foto, um gif curto, mas de alta qualidade seja usado.

Qual problema enfrentaremos no nosso app, ao utilizarmos a estratégia do Array.map()?

Instanciação Dinâmica

Problema

Teremos 256 componentes, com gif, o que os torna pesados, instanciados ao mesmo tempo, porém só uma parte deles será visualizado… A clientela da “Ki Fome Rapaz” não é famosa por ter uma clientela munida de Iphones 14 pro Max, e ao testar em um smartphone mais humilde, você percebeu que seu aplicativo trava toda hora.

Instanciação Dinâmica

Solução:

Ainda bem que, por ser um(a) aluno(a) modelo, você sempre presta atenção nas aulas de Hybrid, e resolve seguir a dica de vasculhar os componentes do react-native

E se depara com o FlatList (ou o SectionList)

FlatList

O que nos oferece:

É um componente que nos permite criar inúmeros outros componentes através de um config ou lista de dados, semelhante ao que o Array,map() nos permite.

Mas qual a vantagem?

FlatList

Vantagens:

Para poucos componentes, onde todos aparecem na tela ao mesmo tempo, ou quase todos, é bem aceitável usarmos o Array.map().
Quando esse número cresce muito, precisamos pensar em performance e facilidade

FlatList

Vantagens:

FlatList é baseado em 2 componentes que já nos ajudam a entender as vantagens.
VirtualizedList e **ScrollView**

FlatList

Virtualização:

No FlatList, componentes que estão fora da tela são virtualizados, e possuem um lazyLoad. Isso é, eles só são de fato carregados quando necessários.
Existe uma “janela” maior que a tela de fato, onde, a partir daquele local, o componente passa a ser instanciado

FlatList

Virtualização:

Além disso, podemos definir facilmente alguns componentes para melhorar nossa interface:

-

Dúvidas, anseios, desabafos?


FIAP