Desenvolvimento para Dispositivos Móveis

Aula 12 - Acessando API

Material Didático do Instituto Metrópole Digital - IMD

Apresentação

Nesta aula, veremos como buscar dados de servidores para utilizá-los no nosso aplicativo.

Objetivos

- Conhecer como utilizar uma *API* para buscar dados
- Conhecer como utilizar o *fetch*
- Conhecer como utilizar o *FlatList* com dados de uma *API*

Buscando dados com o Fetch (Parte 1)

Link do video da aula: https://youtu.be/we7cBmemiEc

Para coletar dados, utilizaremos uma *API* que foi desenvolvida na disciplina de *Back-end*. Porém, poderia ser qualquer *API* ou ferramenta que exporte dados.

Iniciando novo projeto

Inicie um novo projeto através do comando abaixo:

```
expo init ProjectName
```

Buscando dados

Para buscar os dados na API, utilizaremos o *fetch*. Primeiramente, temos de criar um estado para armazenar as informações.

Veja um exemplo abaixo:

```
  const [posts, setPosts] = useState([])
```

Criado o estado, precisamos agora implementar uma função que carregue os dados da *API*. Veja essa implementação abaixo:

```
  const getPostsNaAPI = async () => {
    try {
      const response = await
fetch('https://dev-backend-imd.herokuapp.com/api/posts');
      const posts = await response.json()
```

```
      setPosts(posts.posts)
    } catch (error) {
      setPosts([])
      alert('Falha ao acessar servidor. Tente novamente mais
tarde!')
    }
  }
```

Após criada a função, precisamos chamá-la em algum momento para que os dados sejam carregados. Para isso utilizaremos o *useEffect*, que reage a mudanças de estados.

Veja o uso:

```
useEffect(() => {
    getPostsNaAPI()
  }, [])
```

Utilizando os dados no aplicativo

Agora que já temos os dados carregados e armazenados, podemos utilizá-los no aplicativo. Veja abaixo um exemplo de uso em um componente:

```
<View style={styles.container}>
      <Text>{posts.length > 0 ? posts[0].title : 'Não possui
postagens carregadas'}</Text>
      <StatusBar style="auto" />
</View>
```

Buscando dados com o Fetch (Parte 2)

Link do video da aula: https://youtu.be/XRz2mXC8P4M

A busca de dados nem sempre pode ser rápida, então é importante que o usuário tenha um retorno visual caso tenha de esperar o carregamento.

Criando o *Loading*

Para armazenar o estado do carregamento, utilizaremos novamente o *useState*. Veja a criação abaixo:

```
  const [isLoading, setLoading] = useState(true)
```

Após isso, iremos setar o *isLoading* para *true* no início do carregamento e para *false* assim que os dados estiverem carregados. Veja abaixo como deve ficar a função *getPostsNaAPI()*:

```
const getPostsNaAPI = async () => {
    try {
      setLoading(true)
      const response = await
fetch('https://dev-backend-imd.herokuapp.com/api/posts');
      const posts = await response.json()
      setPosts(posts.posts)
    } catch (error) {
      setPosts([])
      alert('Falha ao acessar servidor. Tente novamente mais
tarde!')
    } finally {
      setLoading(false)
    }
  }
```

Por fim, antes de exibir os dados em tela, iremos verificar se o carregamento finalizou ou não. Caso os dados tenham sido totalmente carregados, eles serão utilizados para exibição. Caso contrário, será exibido um componente de carregamento próprio do *react native*.

Veja um exemplo abaixo:

```
return (
    <View style={styles.container}>
      {isLoading
        ? <ActivityIndicator />
        : <Text style={{ fontSize: 20 }} > {posts.length > 0 ?
posts[0].titulo : 'Não possui postagens carregadas'}</Text>
      }
      <StatusBar style="auto" />
    </View>
);
```

FlatList com dados da API

Link do video da aula: https://youtu.be/3eMnJt6WnjU

Nesta aula, vamos combinar a busca de dados na *API* com um assunto visto em outras aulas: *FlatList*.

Componete *Item*

Para utilizar o *FlatList*, precisamos criar um componente que represente como deve ser apresentado um item da lista. Então, fora do escopo do componente App, implemente o componente abaixo:

```
const Item = (props) => {
  return (
    <View style={styles.item}>
      <Text style={styles.titulo}>{props.item.titulo}</Text>
    </View>
  )
}
```

FlatList

Após a implementação desse componente, já poderemos alterar o código para utilizar o *FlatList*.

Acompanhando o passo a passo da aula, realize as modificações de modo que seu código fique como abaixo:

```
import { StatusBar } from 'expo-status-bar';
import React, { useEffect, useState } from 'react';
import { ActivityIndicator, FlatList, SafeAreaView, StyleSheet,
Text, View } from 'react-native';

const Item = (props) => {
  return (
    <View style={styles.item}>
      <Text style={styles.titulo}>{props.item.titulo}</Text>
    </View>
  )
}

export default function App() {
```

```
  const getPostsNaAPI = async () => {
    try {
      setLoading(true)
      const response = await
fetch('https://dev-backend-imd.herokuapp.com/api/posts');
      const posts = await response.json()
      setPosts(posts.posts)
    } catch (error) {
      setPosts([])
      alert('Falha ao acessar servidor. Tente novamente mais
tarde!')
    } finally {
      setLoading(false)
    }
  }

  useEffect(() => {
    getPostsNaAPI()
  }, [])

  const [posts, setPosts] = useState([])
  const [isLoading, setLoading] = useState(true)

  return (
    <SafeAreaView style={styles.container}>
      {isLoading
        ? <ActivityIndicator />
        : <FlatList data={posts} renderItem={Item}
keyExtractor={item => item.id} />
      }
      <StatusBar style="auto" />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#ffD',
    alignItems: 'center',
    justifyContent: 'center',
  },
  item: {
    backgroundColor: '#0AE',
```

```
    padding: 20,
    margin: 8
  },
  titulo: {
    color: 'white'
  }
});
```

Resumo

Nesta aula, vimos como utilizar o *fetch* para buscar dados de uma API ou qualquer outra ferramenta que forneça dados. Ademais, vimos como utilizar esses dados combinados com o *FlatList*.