CENTRO UNIVERSITÁRIO DE PATOS DE MINAS – UNIPAM
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
TURMA: 7º PERÍODO – 01/2025
DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO AVANÇADOS II
PROFESSOR RAFAEL MARINHO E SILVA

NAVEGAÇÃO EM APLICAÇÕES MOBILE

COMO EXECUTAR OS CÓDIGOS NO CODESPACE DO GITHUB

- Vá para o GitHub e crie um novo repositório.
  - Dê um nome ao repositório, por exemplo,"***navegacao_react_native***".
  - Abra o repositório criado, e clique no botão Code e selecione *Codespaces*.
  - Crie um novo Codespace ou abra um já criado.
- Para usar os recursos de navegação, crie um novo projeto React Native executando no terminal, os seguintes comandos:
  - Instale as dependências principais no Codespace com o seguinte comando no terminal para garantir que o ambiente tenha Node.js e npm instalados:

```
sudo apt update
sudo apt install -y nodejs npm
```

  - Para criar um novo projeto Expo, siga os comandos descritos em cada tipo de navegação.

NAVEGAÇÃO EM APLICAÇÕES MOBILE

A navegação em aplicações mobile é um aspecto crucial para definir a experiência do usuário e a estrutura do aplicativo. A escolha da forma de navegação depende do design e dos requisitos funcionais do aplicativo. Em projetos mais complexos, é comum combinar diferentes tipos de navegação para criar uma experiência de usuário coesa e intuitiva. Cada forma de navegação apresenta características e casos de uso específicos, e a seguir são descritos os principais conceitos e componentes das formas abordadas:

1. NAVEGAÇÃO GAVETA (DRAWER NAVIGATION)

- **Estrutura:** Utiliza um menu lateral que desliza a partir da borda da tela. Esse menu pode agrupar diversas opções e acessar diferentes seções do aplicativo.
- **Componente Principal:** O *Drawer.Navigator* do React Navigation, que contém múltiplas telas (screens) acessíveis via o menu.
- **Quando Usar:** Ideal para aplicações que possuem várias funcionalidades ou seções e onde o acesso a essas opções não precisa ser tão frequente quanto os itens principais. É comum em apps que precisam de um menu global para acessar configurações, perfil, e outras funcionalidades secundárias.
- **Vantagens:** Fornece uma navegação limpa e organizada, liberando espaço na interface principal.
- **Desvantagens:** Pode esconder funcionalidades importantes se o usuário não estiver habituado a abrir o menu; a descoberta de conteúdo pode ser um desafio.

- Comandos para criar projetos com navegação com gaveta:

```
npx create-expo-app navgaveta --template blank
```

```
cd navgaveta
```

- Para configurar a navegação com gaveta, instale as seguintes bibliotecas:

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

```
npm install react-native-screens react-native-safe-area-context react-native-gesture-handler
react-native-reanimated react-native-vector-icons
```

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

- Instale os pacotes expo necessários:

```
npx expo install react-native-gesture-handler react-native-reanimated react-native-screens
react-native-safe-area-context react-native-vector-icons
```

- No diretório raiz do seu projeto, crie o arquivo chamado babel.config.js, e adicione o seguinte conteúdo ao arquivo:

```
module.exports = function(api) {
    api.cache(true);
    return {
        presets: ['babel-preset-expo'],
        plugins: ['react-native-reanimated/plugin'],
    };
};
```

- Estruture a navegação no arquivo App.js. Substitua o conteúdo de App.js por:

```
import React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createDrawerNavigator } from '@react-navigation/drawer';

const Drawer = createDrawerNavigator();

function HomeScreen() {
    return (
        <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
            <Text>Home Screen</Text>
        </View>
    );
}

function ProfileScreen() {
    return (
        <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
            <Text>Profile Screen</Text>
        </View>
    );
}
```

```
export default function App() {
    return (
        <NavigationContainer>
            <Drawer.Navigator>
                <Drawer.Screen name="Home" component={HomeScreen} />
                <Drawer.Screen name="Profile" component={ProfileScreen} />
            </Drawer.Navigator>
        </NavigationContainer>
    );
}
```

- Execute o projeto Expo:

```
npx expo start --tunnel
```

2. NAVEGAÇÃO PILHA (STACK NAVIGATION)

- **Estrutura:** Organiza as telas em uma "pilha", onde cada nova tela é empilhada sobre a anterior. A navegação entre telas é similar a um fluxo linear, onde o usuário pode avançar ou voltar facilmente.
- **Componente Principal:** O ***Stack.Navigator*** que gerencia a transição entre as telas, permitindo animações de empilhamento e retorno.
- **Quando Usar:** Recomendado para fluxos sequenciais ou hierárquicos, como processos de login, cadastros, ou qualquer situação em que o usuário precise avançar por uma sequência de etapas e ter a opção de voltar à tela anterior.
- **Vantagens:** Fornece um gerenciamento de histórico natural, facilitando o retorno à tela anterior e a execução de transições animadas.
- **Desvantagens:** Pode não ser o ideal para navegações paralelas ou quando o aplicativo possui seções independentes que não se relacionam em uma hierarquia linear.
- Comandos para criar projetos com navegação baseada em pilha:

```
npx create-expo-app navpilha --template blank
```

```
cd navpilha
```

- Para configurar a navegação baseada em pilha, instale as seguintes bibliotecas:

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

```
npm install react-native-screens react-native-safe-area-context react-native-gesture-handler
react-native-reanimated react-native-vector-icons
```

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

- Instale os pacotes expo necessários:

```
npx expo install react-native-gesture-handler react-native-reanimated react-native-screens
react-native-safe-area-context react-native-vector-icons
```

- No diretório raiz do seu projeto, crie o arquivo chamado babel.config.js, e adicione o seguinte conteúdo ao arquivo:

```
module.exports = function(api) {
    api.cache(true);
    return {
        presets: ['babel-preset-expo'],
        plugins: ['react-native-reanimated/plugin'],
    };
};
```

- Estruture a navegação no arquivo App.js. Substitua o conteúdo de App.js por:

```
import React from 'react';
import { View, Text, Button } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

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

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

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

- Execute o projeto Expo:

```
npx expo start --tunnel
```

3. NAVEGAÇÃO POR ABAS (BOTTOM TAB NAVIGATION)

- **Estrutura:** Apresenta uma barra de abas na parte inferior da tela, onde cada aba representa uma seção ou funcionalidade principal do aplicativo.
- **Componente Principal:** O ***BottomTab.Navigator***, que permite trocar de tela com um simples toque, oferecendo uma navegação rápida e intuitiva.
- **Quando Usar:** Indicada para aplicativos que possuem seções principais e independentes, como um app de rede social, onde as abas podem representar feed, mensagens, notificações e perfil. É muito útil quando se deseja facilitar a troca de contexto entre funcionalidades centrais.
- **Vantagens:** A interface é de fácil acesso e muito utilizada, oferecendo uma navegação consistente e clara.
- **Desvantagens:** Pode limitar o número de seções que podem ser exibidas simultaneamente, já que a área de abas é limitada.
- Comandos para criar projetos com navegação baseada abas:

```
npx create-expo-app navabas --template blank
```

```
cd navabas
```

- Para configurar a navegação baseada em abas, instale as seguintes bibliotecas:

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

```
npm install react-native-screens react-native-safe-area-context react-native-gesture-handler
react-native-reanimated react-native-vector-icons
```

```
npm install @react-navigation/bottom-tabs
```

- Instale os pacotes expo necessários:

```
npx expo install react-native-gesture-handler react-native-reanimated react-native-screens
react-native-safe-area-context react-native-vector-icons
```

- No diretório raiz do seu projeto, crie o arquivo chamado babel.config.js, e adicione o seguinte conteúdo ao arquivo:

```
module.exports = function(api) {
    api.cache(true);
    return {
        presets: ['babel-preset-expo'],
        plugins: ['react-native-reanimated/plugin'],
    };
};
```

- Estruture a navegação no arquivo App.js. Substitua o conteúdo de App.js por:

```
import React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
```

```
import { Ionicons } from '@expo/vector-icons';

const Tab = createBottomTabNavigator();

function HomeScreen() {
    return (
        <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
            <Text>Home Screen</Text>
        </View>
    );
}

function SettingsScreen() {
    return (
        <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
            <Text>Settings Screen</Text>
        </View>
    );
}

export default function App() {
    return (
        <NavigationContainer>
            <Tab.Navigator
                screenOptions={({ route }) => ({
                    tabBarIcon: ({ focused, color, size }) => {
                        let iconName;

                        if (route.name === 'Home') {
                            iconName = focused
                                ? 'home'
                                : 'home-outline';
                        } else if (route.name === 'Settings') {
                            iconName = focused
                                ? 'settings'
                                : 'settings-outline';
                        }

                        return <Ionicons name={iconName} size={size} color={color} />;
                    },
                    tabBarActiveTintColor: 'tomato',
                    tabBarInactiveTintColor: 'gray',
                })}
            >
                <Tab.Screen name="Home" component={HomeScreen} />
                <Tab.Screen name="Settings" component={SettingsScreen} />
            </Tab.Navigator>
        </NavigationContainer>
    );
}
```

- Execute o projeto Expo:

```
npx expo start --tunnel
```

4. PASSAGEM DE PARÂMETROS ENTRE TELAS

- **Estrutura:** Consiste em enviar informações de uma tela para outra por meio de parâmetros na navegação, utilizando a propriedade route.params.
- **Componentes Envolvidos:** Utiliza os mesmos navegadores (como o Stack Navigator), mas foca na comunicação entre as telas. O mecanismo de passagem de parâmetros é uma funcionalidade integrada ao React Navigation.
- **Quando Usar:** Essencial em fluxos onde os dados inseridos ou selecionados em uma tela precisam ser exibidos ou processados em outra, como formulários, detalhes de itens ou qualquer cenário que exija a transferência de contexto.
- **Vantagens:** Facilita a passagem de dados sem a necessidade de gerenciamento global (como o Context API), mantendo o fluxo de dados simples e direto.
- **Desvantagens:** Pode ficar complexo em aplicativos muito grandes, onde o gerenciamento de múltiplos parâmetros pode requerer soluções mais robustas, como gerenciamento de estado centralizado.
- Comandos para criar projetos com navegação com passagem de parâmetros entre telas:

```
npx create-expo-app navparams --template blank
```

```
cd navparams
```

- Para configurar a navegação com passagem de parâmetros entre telas, instale as seguintes bibliotecas:

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

```
npm install react-native-screens react-native-safe-area-context react-native-gesture-handler
react-native-reanimated react-native-vector-icons
```

- Instale os pacotes expo necessários:

```
npx expo install react-native-gesture-handler react-native-reanimated react-native-screens
react-native-safe-area-context react-native-vector-icons
```

- No diretório raiz do seu projeto, crie o arquivo chamado babel.config.js, e adicione o seguinte conteúdo ao arquivo:

```
module.exports = function(api) {
    api.cache(true);
    return {
        presets: ['babel-preset-expo'],
        plugins: ['react-native-reanimated/plugin'],
    };
};
```

- Crie uma pasta chamada ***components***. Agora crie os arquivos HomeScreen.js e DetailsScreen.js
- Arquivo HomeScreen.js: Este arquivo permite ao usuário inserir dados e navegar para outra tela

```
import React, { useState } from 'react';
import { View, Button, TextInput, StyleSheet } from 'react-native';
```

```
const HomeScreen = ({ navigation }) => {
    const [input, setInput] = useState('');

    return (
        <View style={styles.container}>
            <TextInput
                style={styles.input}
                placeholder="Digite algo"
                value={input}
                onChangeText={setInput}
            />
            <Button title="Enviar" onPress={() => navigation.navigate('Details', { input })} />
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
    },
    input: {
        height: 40,
        borderColor: 'gray',
        borderWidth: 1,
        marginBottom: 20,
        width: '80%',
        paddingHorizontal: 10,
    },
});

export default HomeScreen;
```

- Arquivo DetailsScreen.js: Exibe o parâmetro enviado da tela anterior

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const DetailsScreen = ({ route }) => {
    const { input } = route.params;

    return (
        <View style={styles.container}>
            <Text>Você digitou: {input}</Text>
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
    },
});

export default DetailsScreen;
```

- Estruture a navegação no arquivo App.js. Substitua o conteúdo de App.js por:

```
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationContainer } from '@react-navigation/native';
import HomeScreen from './components/HomeScreen';
import DetailsScreen from './components/DetailsScreen';

const Stack = createStackNavigator();

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

- Execute o projeto Expo:

```
npx expo start --tunnel
```

5. NAVEGAÇÃO COM AUTENTICAÇÃO CONDICIONAL

- **Estrutura:** Implementa um fluxo condicional que alterna entre telas de autenticação (como uma tela de login) e as telas principais do aplicativo, com base no estado de autenticação do usuário.
- **Componentes Principais:** Combina o uso de navegadores (Stack ou Drawer) com lógica condicional para renderizar diferentes fluxos de navegação. Geralmente, utiliza-se um estado global (via hooks ou Context API) para determinar se o usuário está autenticado.
- **Quando Usar:** Fundamental para aplicativos que possuem áreas protegidas e precisam restringir o acesso até que o usuário se autentique. Ideal para apps que possuem dados sensíveis ou funcionalidades exclusivas para usuários logados.
- **Vantagens:** Oferece uma camada extra de segurança e personalização, permitindo que o usuário acesse somente as funcionalidades adequadas ao seu status.
- **Desvantagens:** Requer um gerenciamento cuidadoso do estado de autenticação e pode aumentar a complexidade do fluxo de navegação.
- Comandos para criar projetos com navegação condicional e fluxo de autenticação:

```
npx create-expo-app navauth --template blank
```

```
cd navauth
```

- Funcionalidades do Projeto
  - ➔ Autenticação Simulada: O usuário faz login com as credenciais fixas (user e pass).
  - ➔ Redirecionamento Condicional:
    - ★ Usuário autenticado → Tela principal com navegação por gaveta.
    - ★ Usuário não autenticado → Tela de login.
- Para configurar a navegação condicional e fluxo de autenticação, instale as seguintes bibliotecas:

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

```
npm install @react-navigation/drawer react-native-screens react-native-safe-area-context
react-native-gesture-handler react-native-reanimated react-native-vector-icons
```

- Instale os pacotes expo necessários:

```
npx expo install react-native-gesture-handler react-native-reanimated react-native-screens
react-native-safe-area-context react-native-vector-icons
```

- No diretório raiz do seu projeto, crie o arquivo chamado babel.config.js, e adicione o seguinte conteúdo ao arquivo:

```
module.exports = function(api) {
    api.cache(true);
    return {
        presets: ['babel-preset-expo'],
        plugins: ['react-native-reanimated/plugin'],
    };
};
```

- Crie uma pasta chamada ***components***. Agora crie os arquivos **LoginScreen.js**, **HomeScreen.js**, **ProfileScreen.js** e **AppNavigator.js**.
- Arquivo AppNavigator.js: Implementa a navegação principal com um Drawer Navigator.

```
import React from 'react';
import { createDrawerNavigator } from '@react-navigation/drawer';
import HomeScreen from './HomeScreen';
import ProfileScreen from './ProfileScreen';

const Drawer = createDrawerNavigator();

export default function AppNavigator({ setIsAuthenticated }) {
    return (
        <Drawer.Navigator>
            <Drawer.Screen name="Home">
                {() => <HomeScreen setIsAuthenticated={setIsAuthenticated} />}
            </Drawer.Screen>
            <Drawer.Screen name="Profile">
                {() => <ProfileScreen setIsAuthenticated={setIsAuthenticated} />}
            </Drawer.Screen>
        </Drawer.Navigator>
    );
}
```

- Arquivo LoginScreen.js: Simula uma autenticação que atualiza o estado global.

```
import React, { useState } from 'react';
import { View, TextInput, Button, StyleSheet, Text } from 'react-native';

const LoginScreen = ({ setIsAuthenticated }) => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');

    const handleLogin = () => {
        if (username === 'user' && password === 'pass') {
            setIsAuthenticated(true);
        } else {
            alert('Credenciais inválidas!');
        }
```

```
    };

    return (
        <View style={styles.container}>
            <Text style={styles.title}>Login</Text>
            <TextInput
                style={styles.input}
                placeholder="Usuário"
                value={username}
                onChangeText={setUsername}
            />
            <TextInput
                style={styles.input}
                placeholder="Senha"
                value={password}
                secureTextEntry
                onChangeText={setPassword}
            />
            <Button title="Entrar" onPress={handleLogin} />
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
    },
    title: {
        fontSize: 24,
        marginBottom: 20,
    },
    input: {
        height: 40,
        borderColor: 'gray',
        borderWidth: 1,
        marginBottom: 20,
        width: '80%',
        paddingHorizontal: 10,
    },
});

export default LoginScreen;
```

- Arquivo HomeScreen.js

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const HomeScreen = () => {
    return (
        <View style={styles.container}>
            <Text>Bem-vindo à Home!</Text>
        </View>
    );
};
```

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

export default HomeScreen;
```

- Arquivo ProfileScreen.js

```
import React from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';

const ProfileScreen = ({ setIsAuthenticated }) => {
    return (
        <View style={styles.container}>
            <Text>Bem-vindo ao Perfil!</Text>
            <Button title="Sair" onPress={() => setIsAuthenticated(false)} />
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
    },
});

export default ProfileScreen;
```

- Estruture a navegação no arquivo App.js. Substitua o conteúdo de App.js por:

```
import React, { useState } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import AppNavigator from './components/AppNavigator';
import LoginScreen from './components/LoginScreen';

export default function App() {
    const [isAuthenticated, setIsAuthenticated] = useState(false);

    return (
        <NavigationContainer>
            {isAuthenticated ? (
                <AppNavigator setIsAuthenticated={setIsAuthenticated} />
            ) : (
                <LoginScreen setIsAuthenticated={setIsAuthenticated} />
            )}
        </NavigationContainer>
    );
}
```

- Execute o projeto Expo:

```
npx expo start --tunnel
```