TechPoint - Loja de eletrônicos.

Um projeto feito em React Native.

Desenvolvemos um projeto realizado em React Native, na qual tivemos como tema principal, uma loja de eletrônicos.

Em nossa aplicação, é possível se cadastrar e logar, realizar a simulação de um pedido, confirmar o seu pedido, excluir o seu pedido, editar os dados do seu perfil e sair de sua conta. Com tudo isso, utilizamos tecnologias como: Hooks, Componentes, UseState e UseEffect (gerenciamento nas mudanças de estado), Navigation, bibliotecas nativa do React Native (View, Text, Image e entre outros), AsyncStorage (armazenar e buscar os dados do usuário) e Estilização pelo StyleSheet.
Abaixo está disponível um print das telas e um pouco sobre a tecnologia que utilizamos para criar a página.

LOGIN E CADASTRO:

Nesta tela foi utilizada apenas itens nativos para construção da tela. Segue o código principal da tela:

```
<View style={styles.container}>

    <Text style={styles.title}>Uma mistura de todos os seus eletrônicos favoritos!</Text>

    <TouchableOpacity

     style={styles.button}

     onPress={() => navigation.navigate('Login')}
```

```
  >
    <Text style={styles.buttonText}>Entrar</Text>
  </TouchableOpacity>
  <TouchableOpacity
    style={styles.buttonOutline}
    onPress={() => navigation.navigate('Register')}
  >
    <Text style={styles.buttonOutlineText}>Cadastrar</Text>
  </TouchableOpacity>
</View>
```

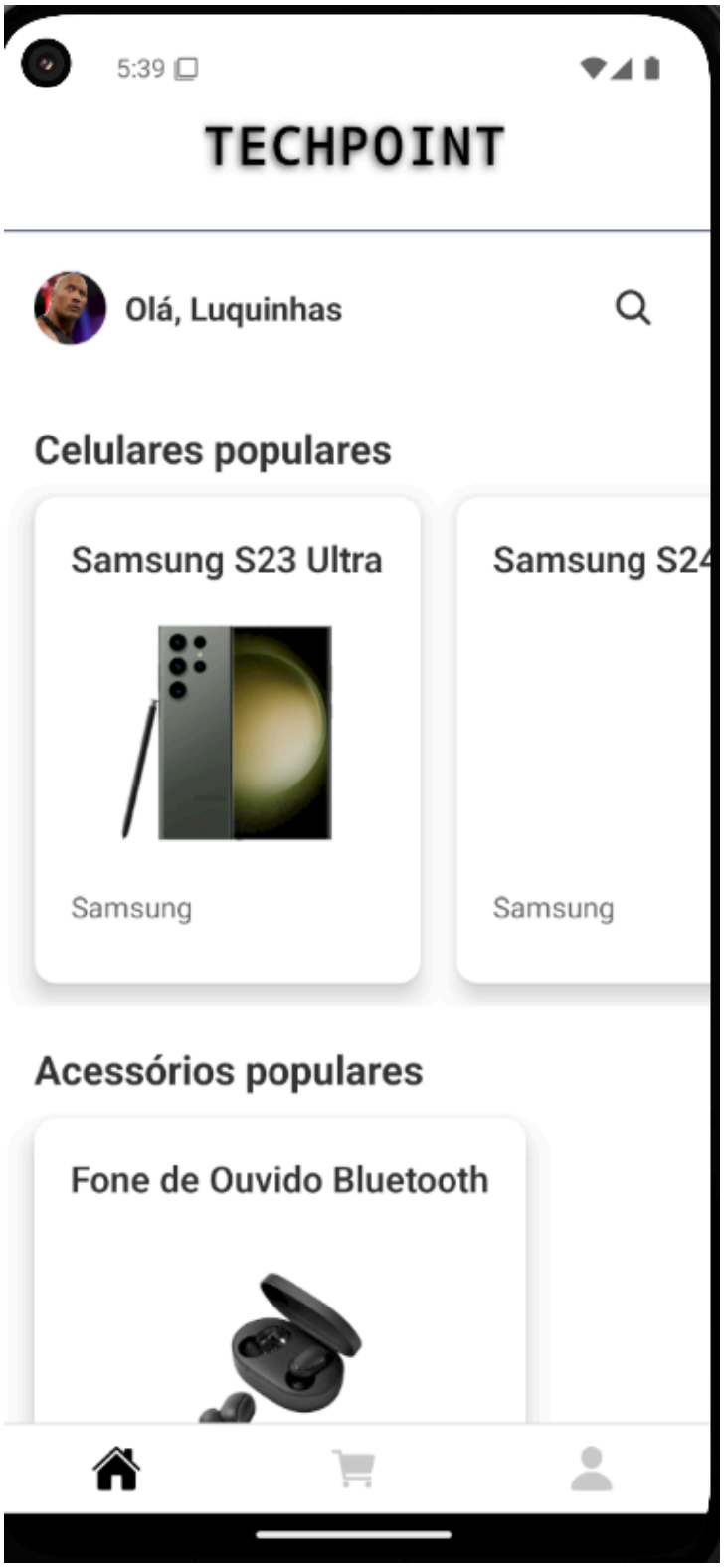
CADASTRO:

Além do UseState criado e do codigo para estruturação visual da tela, criamos essa estrutura de backend. Utilizamos a biblioteca de “axios” para criar a requisição htpp POST, utilizamos também Navigation para navegar entre as rotas após se cadastrar, sendo redirecionado para a página de Home da aplicação. Segue o código:

```
const handleRegister = async () => {

if (password !== confirmPassword) {

  Alert.alert('Erro', 'As senhas não coincidem!');
```

```
    return;
  }

  if (!name || !phone || !email || !password || !profileImage) {
    Alert.alert('Erro', 'Por favor, preencha todos os campos.');
    return;
  }

  try {
    const response = await axios.post('http://10.0.2.2:3000/users', {
      name,
      phone,
      email,
      password,
      profileImage,
    });

    console.log('Usuário registrado:', response.data);
    navigation.navigate('Login');
  } catch (error) {
    console.error('Erro ao registrar usuário:', error);
    Alert.alert('Erro', 'Ocorreu um erro ao tentar registrar o usuário.');
  }
};
```

HOME:

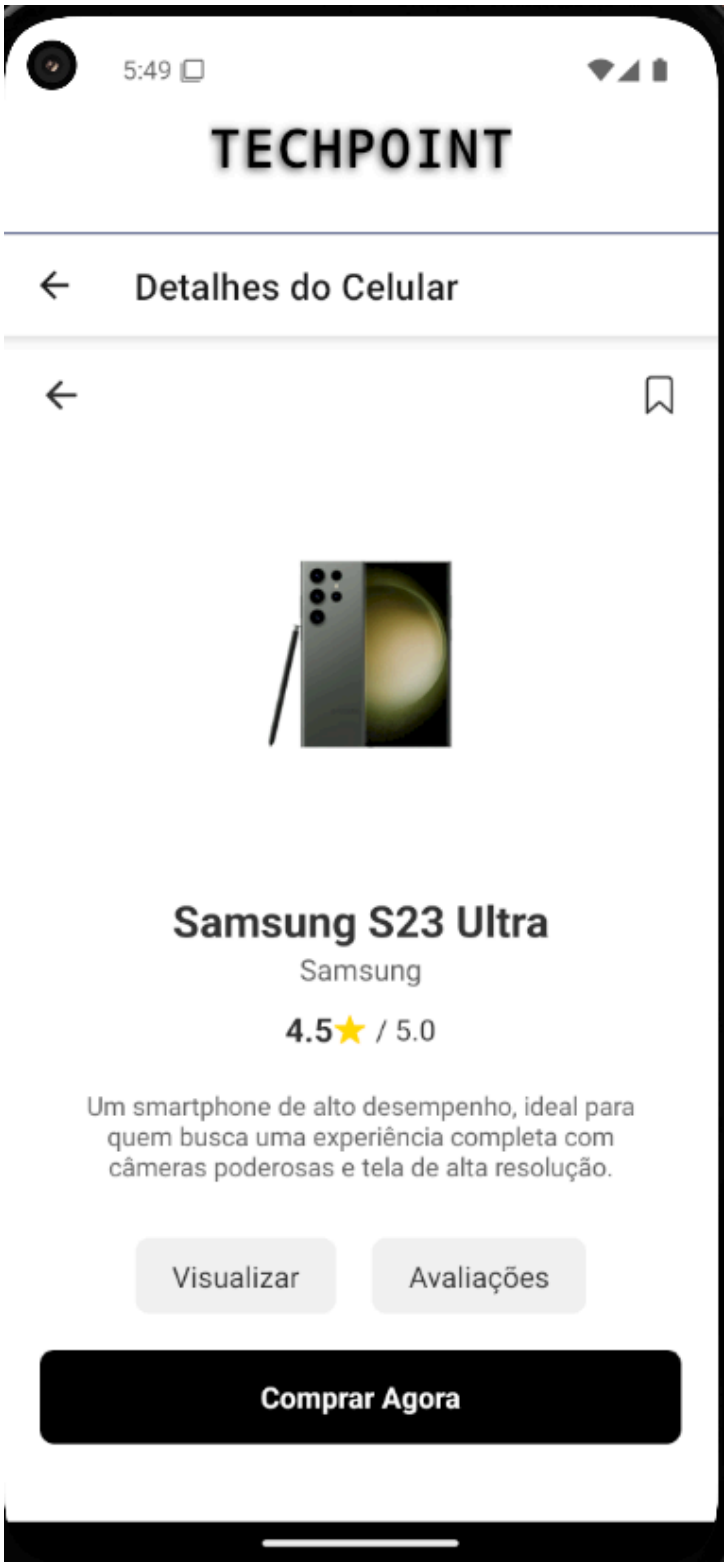


Nesta tela foram utilizadas além da biblioteca e componentes nativos do React Native, Navigation e array para cadastro dos Itens, utilizamos useState e useCallback que são os Hooks do React. Utilizamos também AsyncStorage que é usado para armazenar e recuperar dados localmente no dispositivo, permitindo persistência de dados do usuário, como o nome e a imagem de perfil.
Segue parte do código da página de Home:

```
const navigation = useNavigation();
```

```
const [user, setUser] = useState(null);

// Carregar os dados do usuário sempre que a tela for focada
useFocusEffect(
  React.useCallback(() => {
    const loadUserData = async () => {
      try {
        const name = await AsyncStorage.getItem('userName');
        const profileImage = await AsyncStorage.getItem('userProfileImage');
        setUser({
          name: name || 'Usuário',
          profileImage: profileImage || 'https://cdn-icons-png.flaticon.com/512/1077/1077063.png
'
        });
      } catch (error) {
        console.error('Erro ao carregar dados do usuário:', error);
      }
    };

    loadUserData();
  }, [])
);

const editProfile = () => {
  navigation.navigate('Perfil', { user });
};
```

DETALHES DO PRODUTOS:

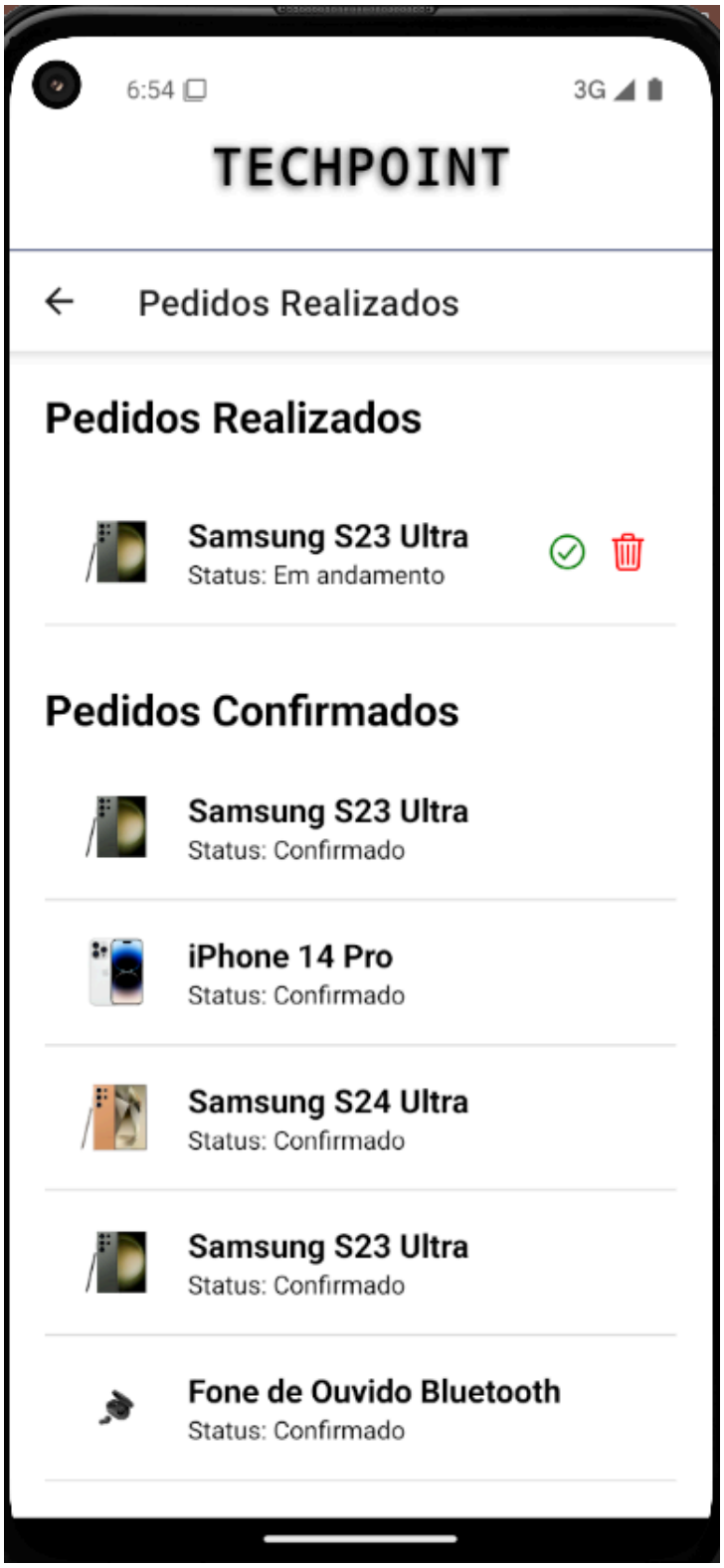


Esse código usa React Native para criar a interface e navegação entre telas, além de React Navigation para facilitar o retorno após a ação de fazer um pedido. O envio dos dados do pedido para um backend local é feito com axios, que, junto com async/await, torna as requisições mais simples e organizadas. A URL 10.0.2.2 conecta o emulador Android ao backend local, salvando os dados sem precisar de uma infraestrutura complexa. Assim, o usuário consegue fazer pedidos e receber feedback imediato no app. Segue parte do código:

```
const { title, imageUri, content } = route.params;
```

```
const handleBuy = async () => {
  try {
    const url = 'http://10.0.2.2:3000/pedidos'; // Ajuste conforme necessário

    const response = await axios.post(url, {
      title,
      imageUri,
      content,
      status: 'Em andamento', // Status inicial do pedido
    });

    if (response.status === 201) {
      alert('Pedido realizado com sucesso!');
    } else {
      alert('Erro ao realizar o pedido.');
    }

    navigation.goBack(); // Volta para a página anterior
  } catch (error) {
    console.error('Erro ao enviar o pedido:', error);
    alert('Erro ao realizar o pedido. Verifique sua conexão com a internet e tente novamente.');
  }
};
```

PEDIDOS REALIZADOS:

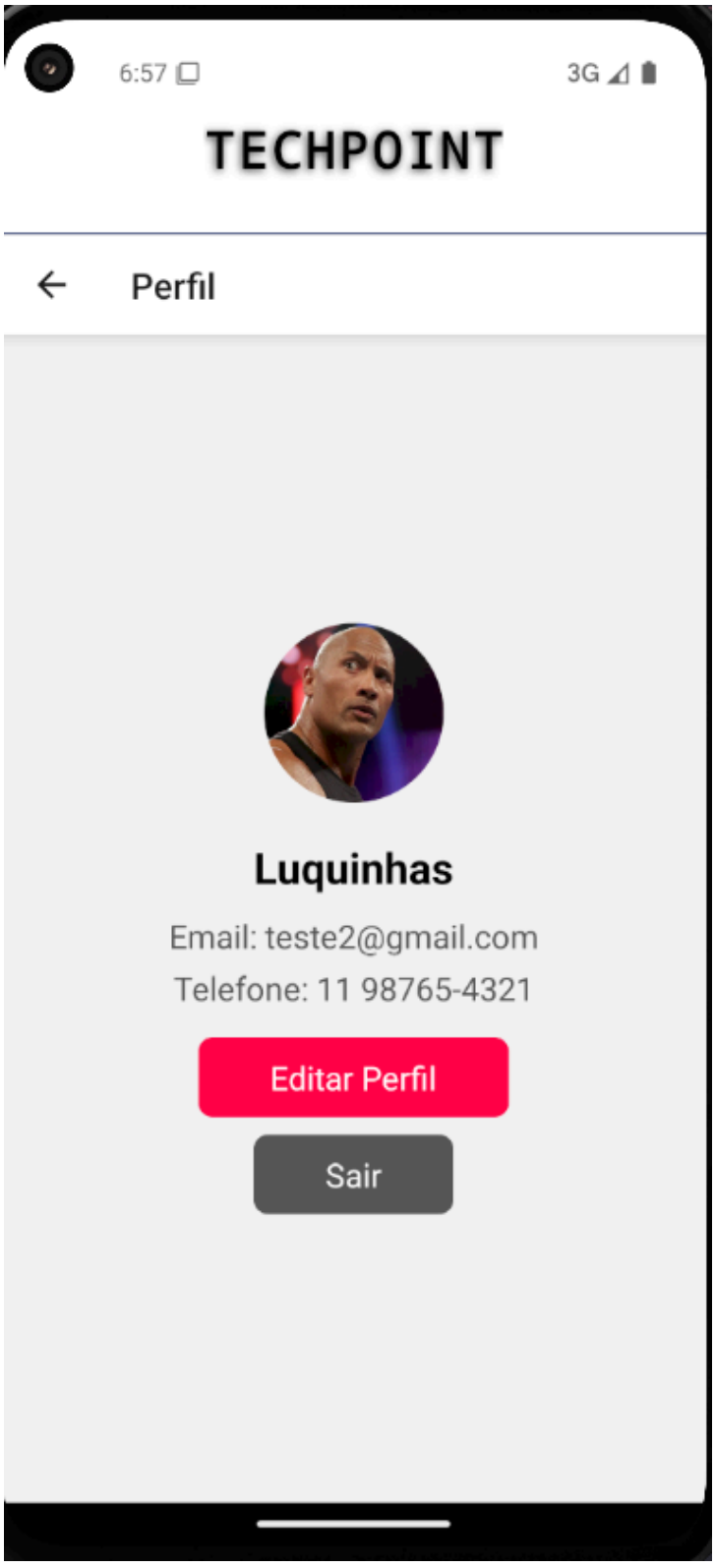


Esse código usa React Native para criar uma interface responsiva, enquanto o hook useEffect executa uma função ao carregar a tela, buscando e exibindo dados com axios. A comunicação com o backend local é feita através do endereço 10.0.2.2, permitindo ao emulador Android acessar o servidor local onde os pedidos são gerenciados. Com axios, o código pode buscar pedidos, confirmar e excluir, oferecendo um feedback imediato ao usuário com Alert.alert para informar sobre o sucesso ou falha das ações. Isso torna o aplicativo mais dinâmico e interativo, mantendo o usuário atualizado sobre suas ações.

```
                                                    useEffect(() => {
  const fetchPedidos = async () => {
    try {
      const response = await axios.get('http://10.0.2.2:3000/pedidos');
      setPedidos(response.data);

      const responseConfirmados = await axios.get('http://10.0.2.2:3000/pedidosConfirmados');
      setPedidosConfirmados(responseConfirmados.data);
    } catch (error) {
      console.error('Erro ao buscar pedidos:', error);
    }
  };

  fetchPedidos();
}, []);

const deletePedido = async (id) => {
  try {
    await axios.delete(`http://10.0.2.2:3000/pedidos/${id}`);
    setPedidos(pedidos.filter(pedido => pedido.id !== id));
    Alert.alert("Sucesso", "Pedido excluído com sucesso!");
  } catch (error) {
    console.error('Erro ao excluir pedido:', error);
    Alert.alert("Erro", "Não foi possível excluir o pedido.");
  }
};

const confirmarPedido = async (pedido) => {
```

```
  try {
    await axios.delete(`http://10.0.2.2:3000/pedidos/${pedido.id}`);
    await axios.post('http://10.0.2.2:3000/pedidosConfirmados', { ...pedido, status: 'Confirmado' });

    setPedidos(pedidos.filter(p => p.id !== pedido.id));
    setPedidosConfirmados([...pedidosConfirmados, { ...pedido, status: 'Confirmado' }]);
    Alert.alert("Confirmado", "Pedido confirmado com sucesso!");
  } catch (error) {
    console.error('Erro ao confirmar pedido:', error);
    Alert.alert("Erro", "Não foi possível confirmar o pedido.");
  }
};
```

PERFIL:



Esse código usa React Native para construir uma interface que permite carregar, editar e salvar dados do usuário de forma dinâmica. O useEffect é utilizado para buscar os dados do usuário assim que a tela é carregada, enquanto AsyncStorage armazena e recupera informações localmente no dispositivo, garantindo que os dados do usuário persistam entre sessões. A biblioteca axios facilita o envio de dados atualizados ao backend local, acessível no emulador Android pelo endereço 10.0.2.2. Com o uso de Alert.alert, o app notifica o usuário sobre o sucesso ou erro

das operações, tornando o processo mais interativo e claro. A função de logout usa AsyncStorage.clear() para limpar os dados e navigation.reset para redirecionar o usuário para a tela inicial, proporcionando uma experiência de navegação fluida e controlada.

```
const [userData, setUserData] = useState({
  name: '',
  email: '',
  phone: '',
  profileImage: '',
});

const [isEditing, setIsEditing] = useState(false);

useEffect(() => {
  const loadUserData = async () => {
    try {
      const name = await AsyncStorage.getItem('userName');
      const email = await AsyncStorage.getItem('userEmail');
      const phone = await AsyncStorage.getItem('userPhone');
      const profileImage = await AsyncStorage.getItem('userProfileImage');

      setUserData({
        name: name || 'Nome do usuário',
        email: email || 'E-mail não disponível',
        phone: phone || 'Telefone não disponível',
        profileImage: profileImage || 'https://www.cnnbrasil.com.br/wp-content/uploads/sites/12/2023/10/shrek-e1696623069422.jpeg',
      });
    } catch (error) {
```

```
      console.error('Erro ao carregar dados do usuário:', error);
    }
  };

  loadUserData();
}, []);

const handleSaveChanges = async () => {
  try {
    await AsyncStorage.setItem('userName', userData.name);
    await AsyncStorage.setItem('userPhone', userData.phone);
    await AsyncStorage.setItem('userProfileImage', userData.profileImage);

    await axios.put('http://10.0.2.2:3000/users/43f7', {
      name: userData.name,
      phone: userData.phone,
      profileImage: userData.profileImage,
    });

    Alert.alert('Sucesso', 'Dados atualizados com sucesso!');
    setIsEditing(false);
    navigation.goBack(); // Retorna para a tela anterior (Home)
  } catch (error) {
    console.error('Erro ao salvar dados do usuário:', error);
    Alert.alert('Erro', 'Não foi possível salvar as alterações.');
  }
};
```

```
const handleLogout = async () => {
  await AsyncStorage.clear(); // Limpa todos os dados do usuário
  navigation.reset({ index: 0, routes: [{ name: 'Welcome' }] }); // Redireciona para a página de boas-vindas
};
```

Conclusão:

Este é o nosso projeto em React Native, criado para proporcionar uma experiência completa e interativa de gerenciamento de dados do usuário. Com React Native, construímos uma interface fluida, enquanto o useEffect e o useState ajudam a atualizar e armazenar o estado do usuário de forma prática. O uso de AsyncStorage permite salvar as informações localmente, garantindo que os dados persistam no dispositivo. Para comunicação com o backend, o axios facilita o envio e atualização dos dados, utilizando o endereço local 10.0.2.2 para integrar o emulador Android com o servidor. Além disso, o Alert.alert fornece feedback em tempo real, garantindo que o usuário seja notificado sobre o sucesso ou erro das operações, e a função de logout redefine a navegação para uma experiência intuitiva e completa.