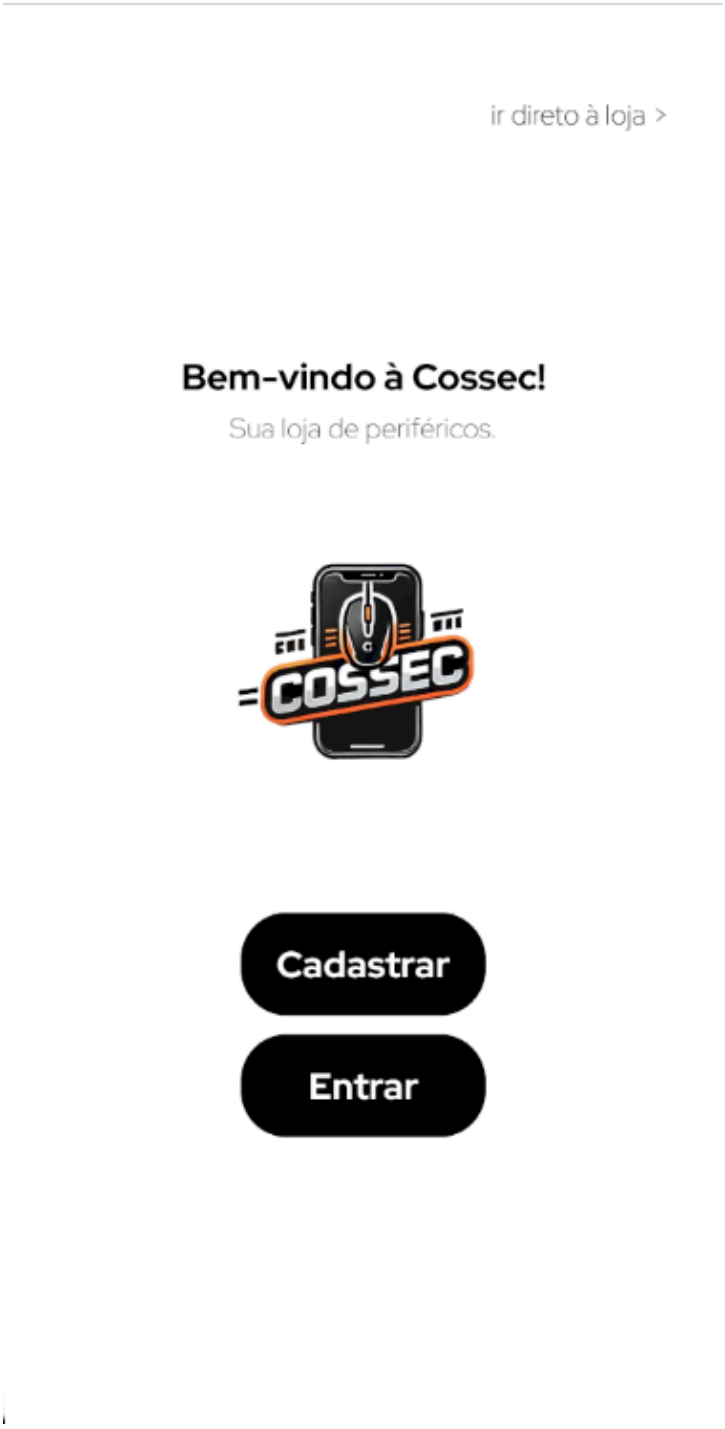
Documentação Projeto Cossec.

Qual o objetivo do projeto?

O aplicativo Cossec é uma loja de periféricos projetada para proporcionar uma experiência prática e eficiente aos usuários. Com funcionalidades de cadastro e login, o aplicativo permite que os clientes explorem uma ampla variedade de produtos, realizem compras diretamente pela plataforma e, além disso, cadastrem seus próprios produtos para venda. A interface foi cuidadosamente desenvolvida para ser intuitiva e fácil de usar, garantindo que os usuários, independentemente de sua familiaridade com tecnologia, possam navegar e interagir com as funcionalidades do aplicativo sem dificuldades.

Navegação entre rotas:

Home:



A tela Home é a introdução ao aplicativo Cossec, apresentando a proposta da loja de periféricos de forma clara e acessível. Com uma interface moderna e intuitiva, ela exibe o logotipo da loja, um texto de boas-vindas e oferece opções para navegar rapidamente pelas principais funcionalidades, como cadastro de novos usuários, login e acesso direto à loja. O design minimalista, com cores neutras e botões destacados, garante uma experiência agradável e de fácil adaptação, enquanto o uso de fontes personalizadas reforça a identidade visual do aplicativo.

Através do arquivo **MainNavigator** é responsável por gerenciar a navegação entre as telas do aplicativo, garantindo uma transição fluida entre as páginas e organizando as

rotas de maneira centralizada. Ele utiliza as bibliotecas @react-navigation/native e @**react-navigation/native-stack** para implementar uma navegação.

Dentro do componente MainNavigator, o NavigationContainer encapsula todo o fluxo de navegação, enquanto o Stack.Navigator define as rotas disponíveis no aplicativo. Cada rota é configurada por um Stack.Screen, que associa um nome a um componente de tela. A tela inicial do aplicativo é definida por meio da propriedade initialRouteName, configurada como "Home", ou seja, a navegação apenas começara através da pagina Home.

```
<NavigationContainer>
    <Stack.Navigator initialRouteName='Home'>
        <Stack.Screen name="Home" component={Home} options={{ headerShown: false }}/>
        <Stack.Screen name="Login" component={Login} options={{ headerShown: false }}/>
        <Stack.Screen name="Cadastro" component={Cadastro} options={{ headerShown: false }}/>
        <Stack.Screen name="Categories" component={Categories} options={{ headerShown: false }}/>
        <Stack.Screen name="Desc" component={Desc} options={{ headerShown: false }}/>
        <Stack.Screen name="Profile" component={Profile} options={{ headerShown: false }}/>
        <Stack.Screen name="AddProd" component={AddProd} options={{ headerShown: false }}/>
        <Stack.Screen name="DeleteProd" component={DeleteProd} options={{ headerShown: false }}/>
    </Stack.Navigator>
</NavigationContainer>
```

E através dos botões presentes na tela eu posso navegar para as paginas de cadastro e login:

```
<TouchableOpacity style={styles.button} onPress={() => navigation.navigate('Cadastro')}>
  <Text style={styles.buttonText}>Cadastrar</Text>
</TouchableOpacity>
<TouchableOpacity style={styles.button} onPress={() => navigation.navigate('Login')}>
  <Text style={styles.buttonText}>Entrar</Text>
</TouchableOpacity>
```

Cadastro:

A tela de Cadastro permite ao usuário registrar-se preenchendo os campos de nome, email, senha, endereço e complemento. O layout é projetado de forma simples e intuitiva, utilizando estilização personalizada com bordas arredondadas e botões bem destacados.

Ao preencher todos os campos e pressionar o botão "Cadastrar", os dados são enviados para um servidor local utilizando o método POST via fetch. Caso o cadastro seja bem-sucedido, o usuário recebe uma mensagem de confirmação e é redirecionado para a tela de login. Em caso de erro, mensagens apropriadas são exibidas, como falhas de conexão ou campos incompletos.

Além disso, o botão "Entrar" permite navegar diretamente para a tela de login sem concluir o cadastro. Um indicador de carregamento aparece enquanto as fontes são carregadas ou durante o processo de envio do formulário, garantindo uma experiência mais interativa e responsiva.

```
const handleRegister = async () => {
    if (!name || !email || !password || !address || !complement) {
        alert('Por favor, preencha todos os campos!');
        return;
    }
    setLoading(true);

    try {
        const response = await fetch('http://10.0.2.2:3000/usuarios', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                id: Date.now().toString(), // Gerando ID único
                nome: name,
                senha: password,
                endereco: address,
                complemento: complement,
                email: email,
            }),
        });
        if (response.ok) {
            alert('Usuário cadastrado com sucesso!');
            setName('');
            setEmail('');
            setPassword('');
            setAddress('');
            setComplement('');
            navigation.navigate('Login');
        } else {
            alert('Erro ao cadastrar o usuário.');
        }
    } catch (error) {
        console.error(error);
        alert('Erro ao conectar-se ao servidor.');
    } finally {
        setLoading(false);
    }
};
```

Login:

A página de login conta com validação básica para os campos de email e senha, exibindo mensagens de erro em caso de campos vazios ou login inválido. A interface apresenta um título com o logotipo, campos estilizados para entrada de email e senha, e botões para "Entrar" e "Cadastrar". Há um link "Esqueci a senha" e uma mensagem de crédito no rodapé.

```
const handleLogin = async () => {
    if (!email || !password) {
        Alert.alert('Erro', 'Por favor, preencha todos os campos.');
        return;
    }

    setLoading(true);

    try {
        const response = await fetch(`http://10.0.2.2:3000/usuarios?email=${email}&senha=${password}`);
        const data = await response.json();

        if (data.length > 0) {
            Alert.alert('Sucesso', 'Login realizado com sucesso!');
            navigation.navigate('Categories', { userName: data[0].nome, email: data[0].email, idCriador: data[0].id });
        } else {
            Alert.alert('Erro', 'Email ou senha incorretos.');
        }
    } catch (error) {
        console.error(error);
        Alert.alert('Erro', 'Não foi possível conectar ao servidor.');
    } finally {
        setLoading(false);
    }
};
```

Categories:

A tela de Categories exibe uma lista de produtos organizados por categorias (como mouse, teclado, headset, etc.) e permite ao usuário navegar entre elas. Ela usa o hook useEffect para buscar os dados de produtos de uma API simulada via json-server e exibe esses produtos em um layout de lista horizontal utilizando o FlatList. O usuário pode visualizar mais produtos de cada categoria clicando em "Ver mais". A interface é organizada em uma estrutura com um cabeçalho contendo o logo e o nome do usuário (que pode ser acessado através do Profile), seguido por uma barra de pesquisa e a exibição dos produtos filtrados por categoria. A tela também possui botões estilizados para navegação e interação com as categorias. O estilo é limpo e moderno.

```
useEffect(() => {
  const fetchProdutos = async () => {
    try {
      const response = await fetch('http://10.0.2.2:3000/produtos');
      const data = await response.json();
      setProdutos(data);
    } catch (error) {
      console.error('Erro ao buscar produtos:', error);
    }
  };

  fetchProdutos();
}, []);

const renderProdutosPorCategoria = (categoria) => {
  const produtosFiltrados = produtos.filter((produto) => produto.categoria === categoria);
  return (
    <View>
      <FlatList
        data={produtosFiltrados}
        horizontal
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => <CardProduct produto={item} />}
      />
    </View>
  );
};
```

Desc:

A tela de Desc exibe os detalhes de um produto selecionado, incluindo nome, preço e descrição. O layout inclui um botão para voltar à tela anterior, e informações como o nome e preço do produto, que são exibidos em destaque. A descrição do produto é mostrada em uma seção separada, com um botão estilizado para a ação de "Comprar". O estilo é limpo e focado nos detalhes do produto, cores contrastantes para facilitar a leitura. A navegação entre as telas é realizada utilizando o navigation.goBack().

```
const { produto } = route.params;
```

```
<TouchableOpacity onPress={() => navigation.goBack()}>
  <Text style={styles.shopText}>{"< Voltar"}</Text>
</TouchableOpacity>
```

AllCategorie e AllProduct:

As telas AllProduct e AllCategorie compartilham a mesma base de estilização, com foco na exibição de produtos em categorias específicas. Ambas utilizam o FlatList para exibir os produtos de forma organizada, com dois produtos por linha, e um botão de navegação para visualizar mais produtos. A principal diferença entre elas está no tipo de dados que elas processam: enquanto AllCategorie exibe produtos filtrados com

base na busca ou na categoria, AllProduct exibe todos os produtos de uma categoria selecionada.

```
const handleSearch = () => {
  const filteredProdutos = produtos.filter((produto) =>
    produto.nome.toLowerCase().includes(searchQuery.toLowerCase())
  );
  navigation.navigate('AllCategorie', { filteredProdutos });
};

useEffect(() => {
  const fetchProdutos = async () => {
    try {
      const response = await fetch('http://10.0.2.2:3000/produtos');
      const data = await response.json();
      setProdutos(data);
    } catch (error) {
      console.error('Erro ao buscar produtos:', error);
    }
  };

  fetchProdutos();
}, []);

const renderProdutosPorCategoria = (categoria) => {
  const produtosFiltrados = produtos.filter((produto) => produto.categoria === categoria);
  return (
    <View>
      <FlatList
        data={produtosFiltrados}
        horizontal
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => <CardProduct produto={item} />}
      />
      <View style={styles.buttonContainer}>
        <TouchableOpacity
          style={styles.button}
          onPress={() => navigation.navigate('AllProduct', { filteredProdutos: produtosFiltrados })}
        >
          <Text style={styles.buttonText}>Ver mais</Text>
        </TouchableOpacity>
      </View>
    </View>
```

Profile:

A tela de **Profile** exibe o perfil do usuário, mostrando seu nome, email. Ao entrar na tela, o usuário pode voltar para a tela anterior usando o botão de voltar no topo. As opções de navegação incluem a visualização da lista de compras, a adição ou exclusão de produtos, além de um botão de logout que redireciona para a tela inicial. A tela é estilizada com botões grandes e com bordas arredondadas, dando um visual moderno e intuitivo.

```
const navigation = useNavigation();
const route = useRoute();
const { userName, email, idCriador } = route.params || {};
```

AddProd:

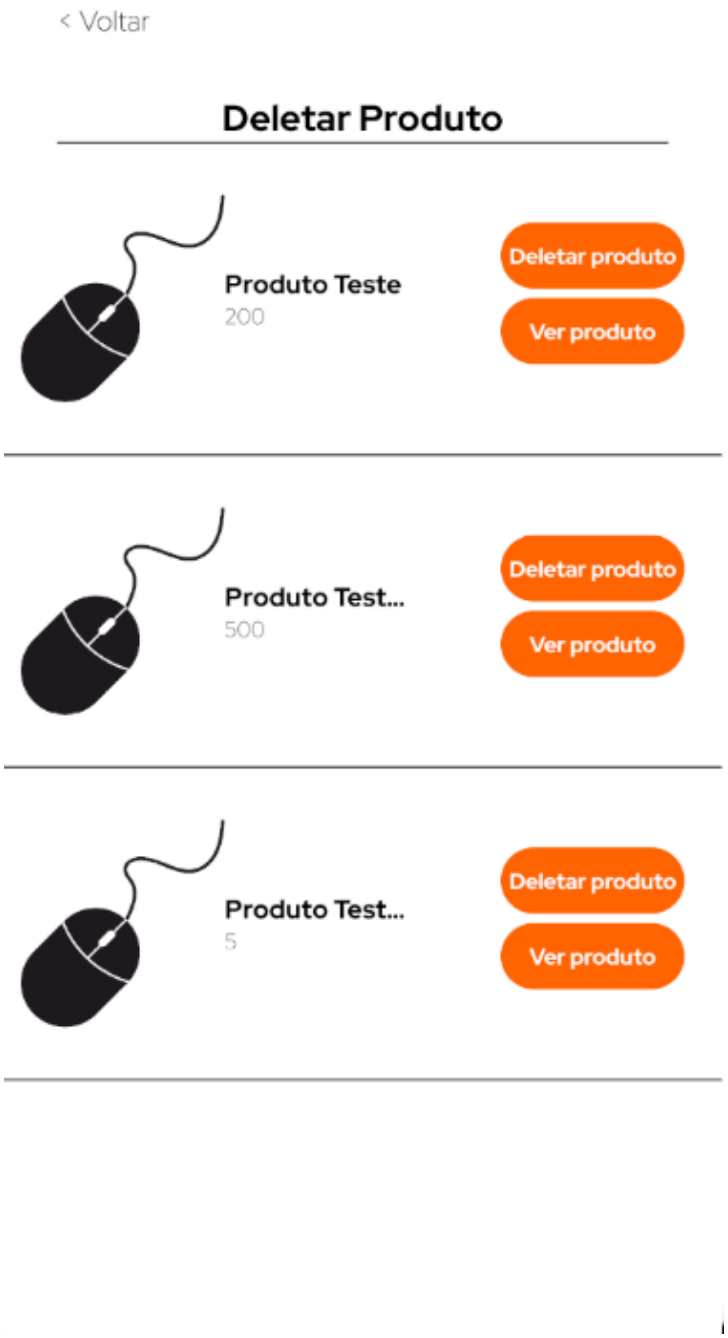
A tela AddProduct permite que o usuário adicione um novo produto ao sistema. Ela possui campos para inserir o nome do produto, categoria, preço e descrição. Ao preencher esses campos e clicar no botão "Adicionar", os dados são enviados para o servidor usando uma requisição POST. Enquanto o produto é adicionado, o botão exibe um indicador de carregamento. Caso algum campo obrigatório não seja preenchido, o sistema alerta o usuário para completar todos os campos.

```
const handleAddProduct = async () => {
  if (!productName || !category || !price || !description) {
    alert('Por favor, preencha todos os campos!');
    return;
  }

  setLoading(true);

  try {
    const response = await fetch('http://10.0.2.2:3000/produtos', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        id: Date.now().toString(),
        nome: productName,
        categoria: category,
        preco: parseFloat(price),
        desc: description,
        idDoCriador: criadorId
      }),
    });
    if (response.ok) {
      alert('Produto adicionado com sucesso!');
      setProductName('');
      setCategory('');
      setPrice('');
      setDescription('');
    } else {
      alert('Erro ao adicionar o produto.');
    }
  } catch (error) {
    console.error('Erro:', error);
    alert('Ocorreu um erro ao conectar-se ao servidor.');
  } finally {
    setLoading(false);
  }
};
```

DeleteProd:



A tela DeleteProd é responsável por listar e permitir a exclusão de produtos associados a um criador específico. Ao carregar a página, ela faz uma requisição GET para buscar os produtos e exibi-los em uma lista. Os produtos que não pertencem ao criador identificado por idCriador são filtrados e não são mostrados. Para excluir um produto, o usuário pode clicar no botão “Deletar produto” que chama a função removeProduto para atualizar o estado e remover o item da lista, e caso o usuário queria ver mais detalhes do produto, ele pode utilizar o botão “Ver produto” e será direcionado à pagina Desc do mesmo. Caso não haja produtos para exibir, uma mensagem "Nenhum produto encontrado" é mostrada. O estilo da tela é simples, com fontes personalizadas. Se a requisição de produtos falhar, um alerta de erro é exibido. A tela também utiliza um FlatList para renderizar os itens de forma otimizada.

Json-Server:

O json-server é uma ferramenta de desenvolvimento extremamente útil para simular uma API RESTful em ambientes de teste ou protótipos. Ele permite criar rapidamente um servidor que simula um backend real utilizando um arquivo JSON como banco de dados. No seu projeto, o json-server está sendo utilizado para fornecer dados simulados para a aplicação, sem a necessidade de um backend real.

Conclusão:

Este projeto cumpre os objetivos propostos, apresentando uma solução funcional e modular que reflete os conceitos aprendidos em aula. A estrutura bem organizada e a interface amigável proporcionam uma ótima experiência ao usuário.

Repositório:

O código fonte está disponível no GitHub:

🌐 GitHub - lucassdolv/NewProjectCossec