Documentação da missão 1

Nome: Israel dos Santos Hamdan D'Araújo
matricula: 202303592086
email: contatoIsraelHamdan@gmai.com

Instruções

Requisitos:
Para rodar a aplicação é necessário o Nodejs e o Expo instalados na sua máquina
Além disso é necessário um dispositivo android ou iOS com o app Expo Go instalado

Primeiros passos

1. Clonar o projeto **https://github.com/IsraelHamdan/Missao-pratica-1.git**
2. Instalar dependências, dentro da pasta missao_pratica, usar o comando npm install para instalar todas as dependências do projeto
3. Colocar o seu endereço de ip dentro da variável api, no arquivo useFornecedores.ts, para isso é necessário saber o seu endereço de IP, isso pelo motivo de que, para rodar o projeto em um dispositivo mobile, vc precisa do endereço da API dele, e como não estamos rodando tudo dentro do mesmo dispositivo, o localhost não funciona!
4. na pasta missao_pratica uusar o comando json-server --watch db.json --host 0.0.0.0 --port 3000 para iniciar o backend da aplicação
5. Depois de iniciar o servidor do backend é necessário iniciar o Expo, então dentro da pasta missao_pratica dar o comando npx expo start
6. Escanear o QR code com o app do expo então a aplicação irá iniciar

CÓDIGOS

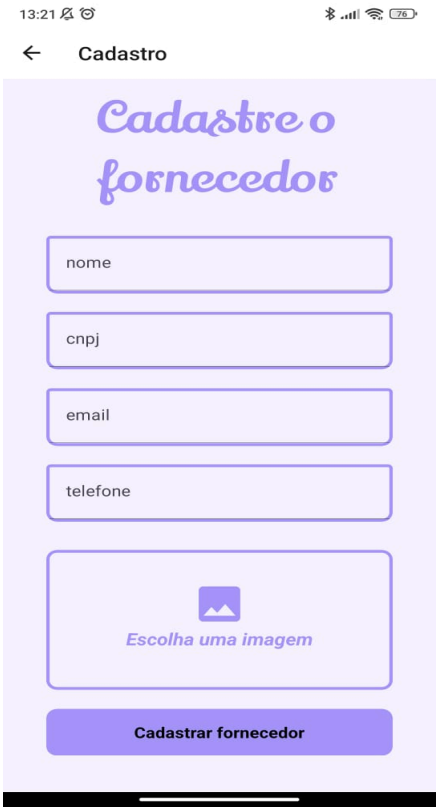
Tela de inicial

issao_pratica > components > HomeScreen > HomeScreen.tsx > HomeScreen

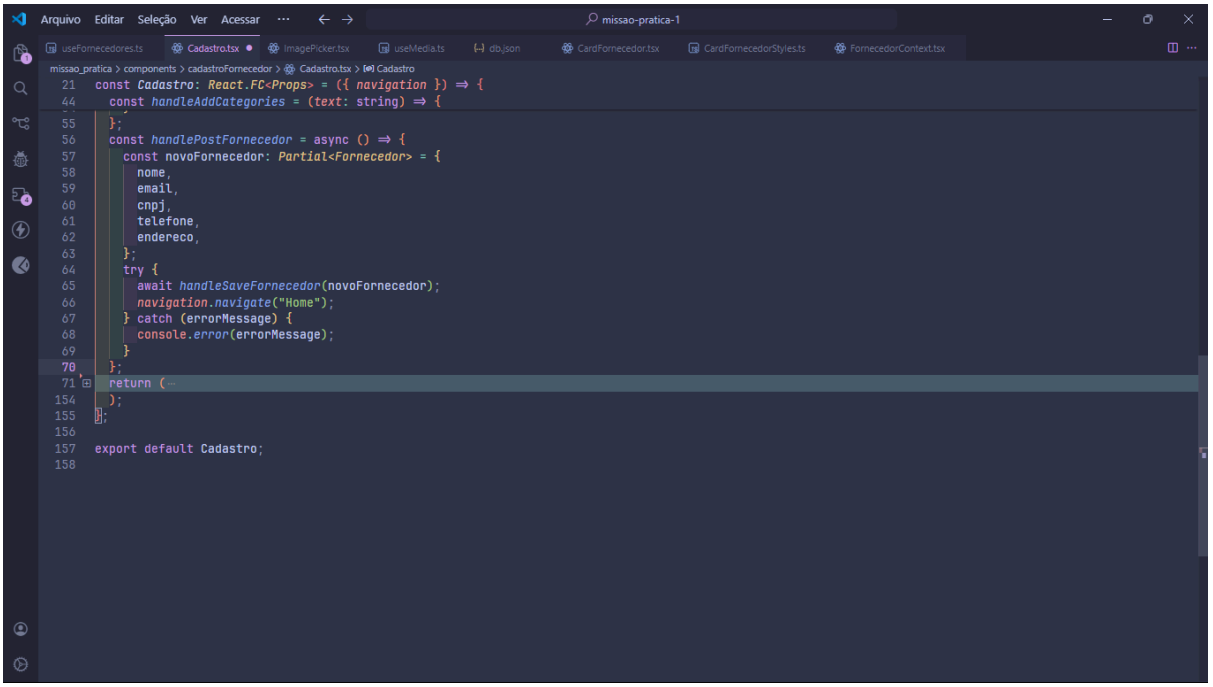
```tsx
import React from "react";
import { View, SafeAreaView } from "react-native";
import { Text, Button } from "react-native-paper";
import Styles from "./HomeScreenStyles";
import { Props } from "../../utils/Props";
const HomeScreen: React.FC<Props> = ({ navigation }) => {
  return (
    <SafeAreaView style={Styles.continer}>
      <View style={Styles.titleView}>
        <Text style={Styles.h1}>Meeting</Text>
      </View>
      <View style={Styles.btnView}>
        <Button
          style={Styles.btn}
          onPress={() => navigation.navigate("Cadastro")}>
          <Text style={Styles.textBtn}>Cadastrar fornecedor</Text>
        </Button>
        <Button
          style={Styles.btn}
          onPress={() => {
            navigation.navigate("Lista");
          }}>
          <Text style={Styles.textBtn}>Listar fornecedores</Text>
        </Button>
      </View>
    </SafeAreaView>
  );
};

export default HomeScreen;
```

A tela inicial foi bem simples, não envolve muita lógica ela é a porta de entrada para a aplixação, então eu implementei uma roterização, para quando o os botões fossem clicados, o usuário ser redirecionado, na seção de funcionamento interno explico como isso se da

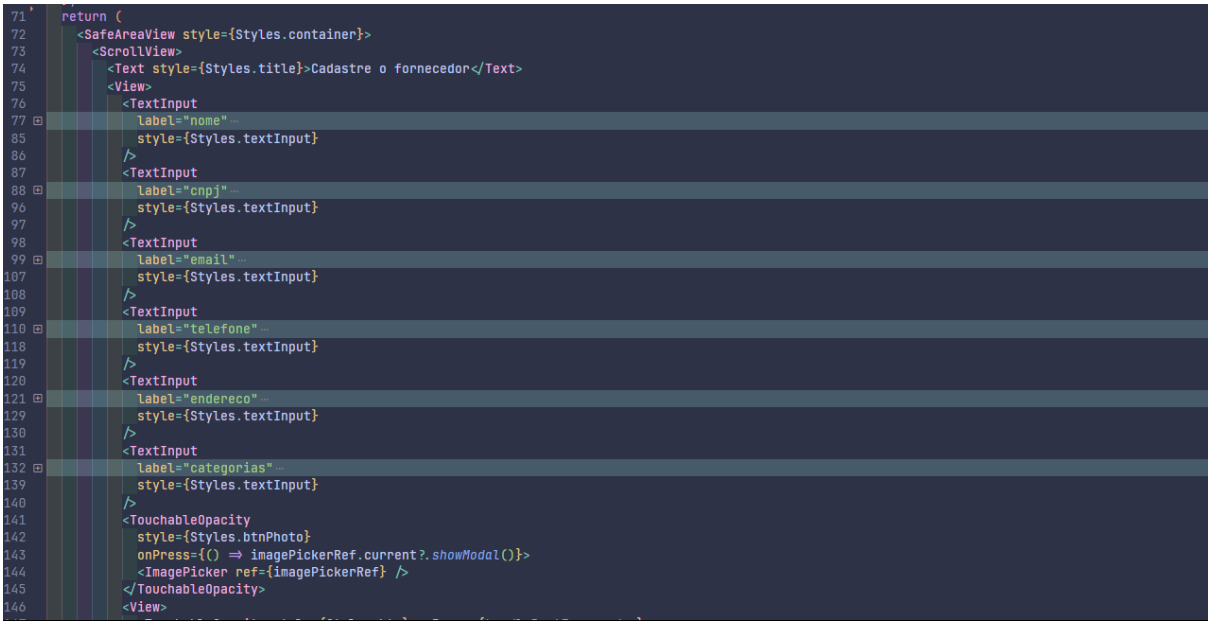
Tela de cadastro




```tsx
const Cadastro: React.FC<Props> = ({ navigation }) => {
  const [nome, setNome] = useState<string>("");
  const [cnpj, setCnpj] = useState<string>("");
  const [email, setEmail] = useState<string>("");
  const [telefone, setTelefone] = useState<string>("");
  const [imagem, setImagem] = useState<string>("");
  const [endereco, setEndereco] = useState<string>("");
  const [categorias, setCategorias] = useState<string[]>([]);
  const imagePickerRef = useRef<ImagePickerHandle>(null);
  const { fornecedorData, updateFornecedorData } = useFornecedorContext();

  const {
    fornecedores,
    isLoading,
    errorMessage,
    findFornecedores,
    handleSaveFornecedor,
  } = useFornecedores();

  useEffect(() => {
    findFornecedores();
  }, []);

  const handleAddCategories = (text: string) => {
    const categoryArray = text
      .split(",")
      .map((category) => category.trim())
      .filter((category) => category !== "");
    const uniqueCategories = [...new Set(categoryArray)];
    if (Array.isArray(categoryArray)) {
      updateFornecedorData({
        categorias: uniqueCategories,
      });
```



Já a parte de cadastro foi a parte mais difícil, pois nessa aplicação foi utilizado context api, para que fosse possível atualizar globalmente o estado do fornecedor, para que todas as partes da aplicação tivessem o mesmo dado, no mesmo momento, a parte de categorias e a parte de imagem foram as mais trabalhosas de montar, pois eu precisava solicitar permissão do usuário para usar sua galeria, e também tive que tratar a imagem (explicarei com mais detalhes na parte de funcionamento interno) já a questão das categorias, foi difícil por causa da tipagem do TypeScript, pois ela na verdade , é um array de strings e isso deu um severo trabalho para ajustar cada parte



```
            />
            <TouchableOpacity
              style={Styles.btnPhoto}
              onPress={() => imagePickerRef.current?.showModal()}>
              <ImagePicker ref={imagePickerRef} />
            </TouchableOpacity>
            <View>
              <TouchableOpacity style={Styles.btn} onPress={handlePostFornecedor}>
                <Text style={Styles.btnText}>Cadastrar fornecedor</Text>
              </TouchableOpacity>
            </View>
          </View>
        </ScrollView>
      </SafeAreaView>
    );
  };

  export default Cadastro;
```

Tela de listagem

```
const ListaFornecedores: React.FC<Props> = ({ navigation }) => {
  const { fornecedores, isLoading, errorMessage, findFornecedores } =
    useFornecedores();
  useEffect(() => {
    findFornecedores();
  }, []);

  if (isLoading) {…
  }

  if (errorMessage) {…
  }
  return (
    <SafeAreaView style={Styles.container}>
      <Button
        onPress={() => {
          navigation.navigate("Cadastro");
        }}
        style={styles.btn}>
        Cadastrar Fornecedor
      </Button>
      <FlatList
        data={fornecedores}
        keyExtractor={(item, index) => index.toString()}
        renderItem={({ item }) => <CardFornecedor fornecedor={item} />}
      />
    </SafeAreaView>
  );
};
```

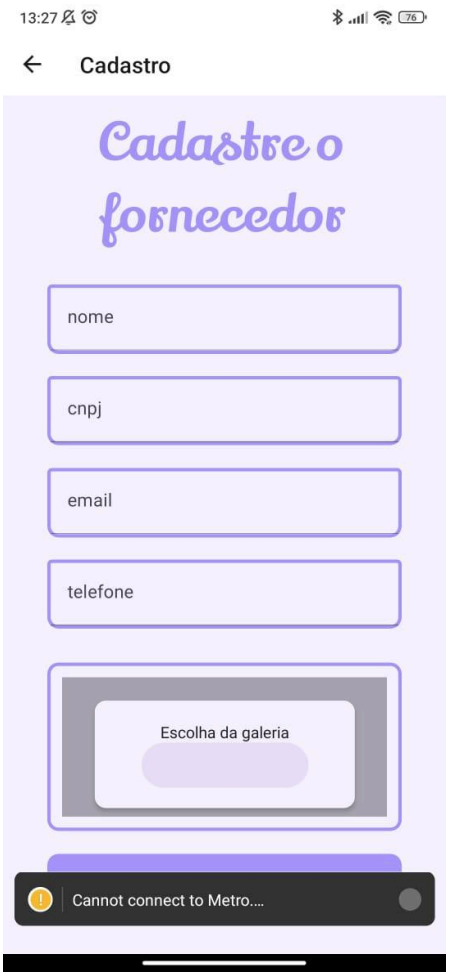
A parte de listagem envolve basicamente buscar os dados do fornecedor, isso quem faz é o FlatList e a parte de renderizar a imagem é feita pelo CardFornecedor, que é mostrado a abaixo

Cards da tela de listagem

```tsx
const CardFornecedor: React.FC<CardFornecedorProps> = ({ fornecedor }) => {
  return (
    <View style={Styles.card}>
      <Image
        style={Styles.cardImage}
        source={{ uri: fornecedor.imagem }}
        resizeMode="stretch"
      />
      <View style={Styles.textContainer}>
        <View style={Styles.cardsInfoRow}>
          <Text style={Styles.cardInfo}>Nome </Text>
          <Text style={Styles.cardInfoText}>{fornecedor.nome}</Text>
        </View>
        <View style={Styles.cardsInfoRow}>···
        </View>
        <View style={Styles.cardsInfoRow}>···
        </View>
        <View style={Styles.cardsInfoRow}>···
        </View>
        <View style={Styles.cardsInfoRow}>···
        </View>
      </View>
    </View>
  );
};
```

Seletor de imagens

```tsx
const ImagePicker = forwardRef((props, ref) => {
  const [visible, setVisible] = useState<boolean>(false);
  const { handleTakeGalleryImage, uri } = useMedia();
  const showModal = () => setVisible(true);
  const hideModal = () => setVisible(false);

  useImperativeHandle(ref, () => ({
    showModal,
    hideModal,
  }));

  return (
    <Provider>
      <View style={Styles.view}>
        {uri ? (
          <View>
            <Ionicons size={100} color={"#A594F9"} name="checkmark-circle" />
          </View>
        ) : (
          <Portal>
            {!visible ? (
              <View style={Styles.btn}>
                <Icon size={50} source="image" color="#A594F9" />
                <Text style={Styles.textBtn}>Escolha uma imagem</Text>
              </View>
            ) : (
              <Modal
                visible={visible}
                onDismiss={hideModal}
                contentContainerStyle={Styles.modal}>
                <Text>Escolha da galeria</Text>
                <Button mode="contained-tonal" onPress={handleTakeGalleryImage}>
                  <View style={Styles.icons}>
                    <Ionicons size={48} color={"#A594F9"} name="image" />
                  </View>
                </Button>
              </Modal>
            )}
          </Portal>
        )}
      </View>
    </Provider>
  );
});
```



Partes internas do funcionamento:

Usefornecedores => Hook customizado que faz o cadastro do usuário

```tsx
};

export const useFornecedores = () => {
  const [fornecedores, setFornecedores] = useState<Fornecedor[]>([]);
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const [errorMessage, setErrorMessage] = useState<string | null>(null);
  const { handleTakeGalleryImage } = useMedia();
  const { fornecedorData } = useFornecedorContext();

  const findFornecedores = async () => {
    setIsLoading(true);
    setErrorMessage(null);
    try {
      const data = await list();
      setFornecedores(data);
    } catch (err: any) {
      setErrorMessage(err.message);
    } finally {
      setIsLoading(false);
    }
  };
```

```tsx
  };
  const handleSaveFornecedor = async (
    fornecedor: Partial<Fornecedor>
  ): Promise<void> => {
    setIsLoading(true);
    setErrorMessage(null);
    try {
      const novoFornecedor: Fornecedor = {
        ...(fornecedorData as Fornecedor),
        ...(fornecedor as Fornecedor),
      };
      await postFornecedoresOnJSON(novoFornecedor);
    } catch (err) {
      console.error(`Erro ao salvar o fornecedor: ${err}`);
    }
  };
  const postFornecedoresOnJSON = async (fornecedor: Fornecedor) => {
    setIsLoading(true);
    setErrorMessage(null);
    try {
      const fornecedorCadastrado = await cadastro(fornecedor);
      if (fornecedorCadastrado) {
        setFornecedores((prev) => [...prev, fornecedorCadastrado]);
      }
    } catch (err: any) {
      console.error(err);
      setErrorMessage(err);
    } finally {
      setIsLoading(false);
    }
```

```
54    };
55    const handleSaveFornecedor = async (
56      fornecedor: Partial<Fornecedor>
57    ): Promise<void> => {
58      setIsLoading(true);
59      setErrorMessage(null);
60      try {
61        const novoFornecedor: Fornecedor = {
62          ...(fornecedorData as Fornecedor),
63          ...(fornecedor as Fornecedor),
64        };
65        await postFornecedoresOnJSON(novoFornecedor);
66      } catch (err) {
67        console.error(`Erro ao salvar o fornecedor: ${err}`);
68      }
69    };
70    const postFornecedoresOnJSON = async (fornecedor: Fornecedor) => {
71      setIsLoading(true);
72      setErrorMessage(null);
73      try {
74        const fornecedorCadastrado = await cadastro(fornecedor);
75        if (fornecedorCadastrado) {
76          setFornecedores((prev) => [...prev, fornecedorCadastrado]);
77        }
78      } catch (err: any) {
79        console.error(err);
80        setErrorMessage(err);
81      } finally {
82        setIsLoading(false);
83      }
```

Esse hook é responsável por receber todos os dados do usuário e cadastrá-lo no banco de dados

useMedia => Hook responsável por pegar a imagem da galeria do usuário, converter ela em base64, salvar na memória interna do dispositivo e retornar o caminho dela no dispositvo

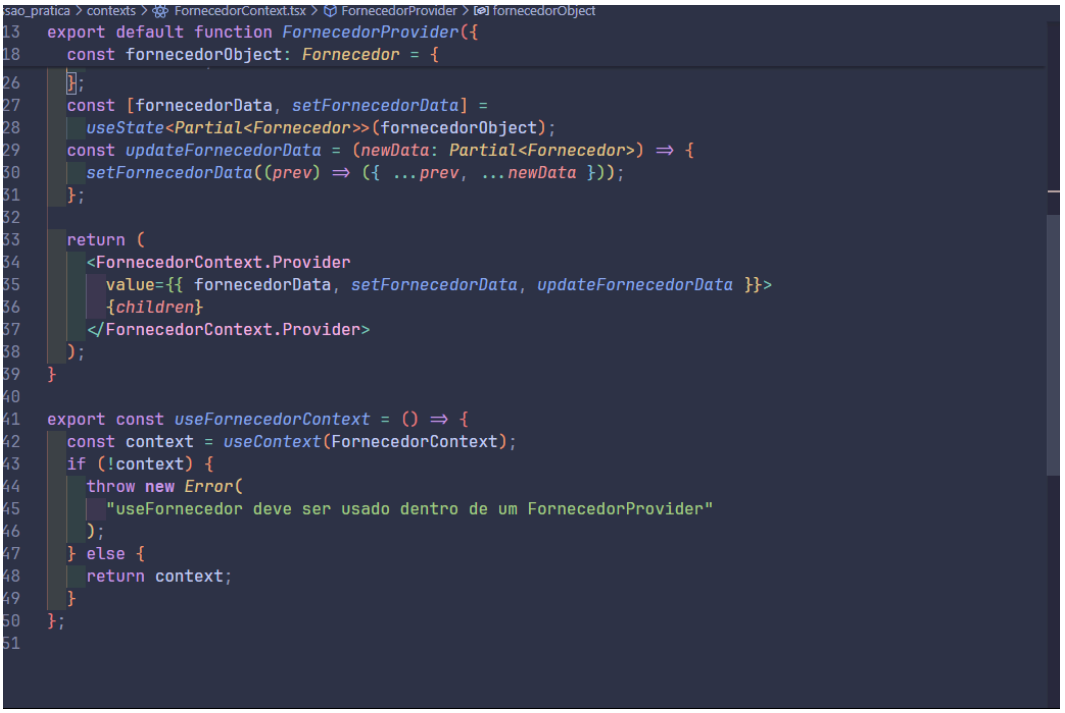
```
// requisita acesso a galeria!
const requestPermissionToAcessGallery = async () => {
  try {
    const { status } = await ImagePicker.requestMediaLibraryPermissionsAsync();
    if (status != "granted") {
      Alert.alert(
        "Permissão negada, precisamos de permissão para acessar sua galeria"
      );
    } else {
      return { granted: status === "granted" };
    }
  } catch (error) {
    console.log("🚀 ~ requestPermissionToAcessGallery ~ error:", error);
    return { granted: false };
  }
};
// seleciona a imagem da galeria e retorna o endereco dela
```

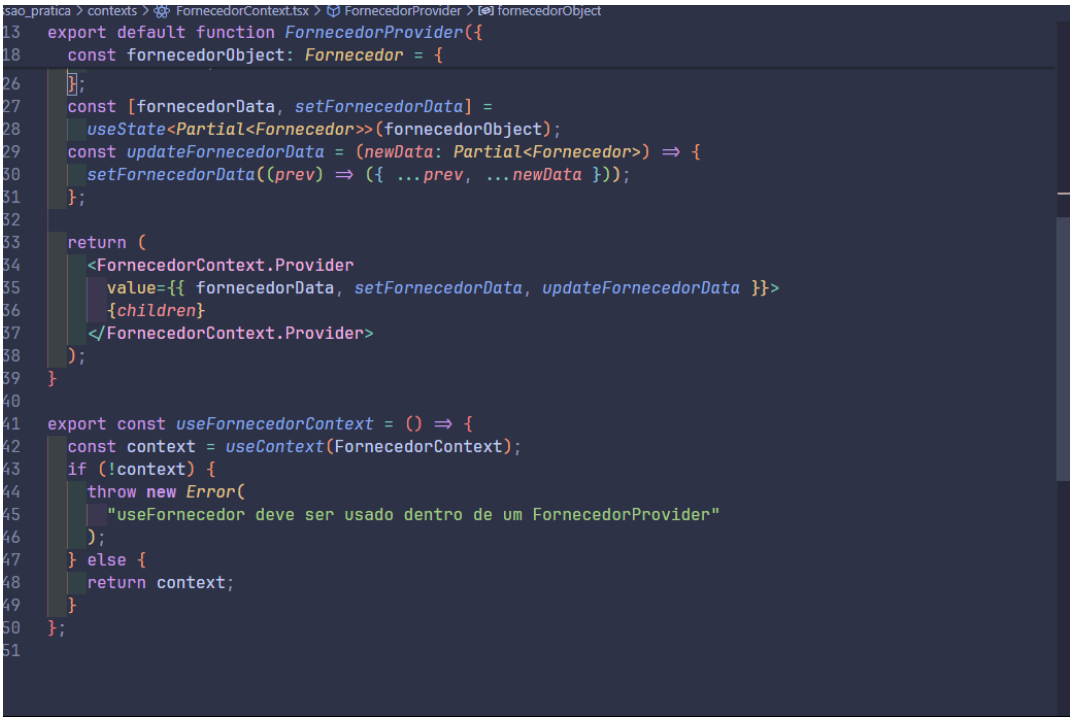
```
};
// seleciona a imagem da galeria e retorna o endereço dela
const selectImageFromGallery = async () => {
  const permission = await requestPermissionToAcessGallery();
  if (!permission?.granted) {
    Alert.alert(
      "Permissão negada",
      "Precisamos de permissão para acessar sua galeria!"
    );
    return null;
  }
  const result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    aspect: [6, 5],
    quality: 1,
  });
  return result.canceled ? null : result.assets[0].uri;
};
```


```
};
// seleciona a imagem da galeria e retorna o endereço dela
const selectImageFromGallery = async () => {
  const permission = await requestPermissionToAcessGallery();
  if (!permission?.granted) {
    Alert.alert(
      "Permissão negada",
      "Precisamos de permissão para acessar sua galeria!"
    );
    return null;
  }
  const result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    aspect: [6, 5],
    quality: 1,
  });
  return result.canceled ? null : result.assets[0].uri;
};
```

FornecedorContext

Como mencionado anteriormente este projeto usa ContextApi do React para fazer o gerenciamento de estados, para isso é necessário criar um provedor de contexto que é um componente





Além disso é necessário que a aplicação inteira esteja dentro do contexto global de dados, como mostrado na imagem

```
return (
  <FornecedorProvider>
    <NavigationContainer>
      <PaperProvider>
        <Stack.Navigator>
          <Stack.Screen name="Home" component={HomeScreen} />
          <Stack.Screen name="Cadastro" component={Cadastro} />
          <Stack.Screen name="Lista" component={ListaFornecedores} />
        </Stack.Navigator>
      </PaperProvider>
    </NavigationContainer>
  </FornecedorProvider>
);
```

Roteamento dentro da aplicação

O roteamento de páginas dentro do app é dividido em 2 partes, a primeira são as props de roteamento

```
import { StackNavigationProp } from "@react-navigation/stack";

export type RootStackParamList = {
  Home: undefined;
  Cadastro: undefined;
  Lista: undefined;
};
export type RootStackNavigationProps = StackNavigationProp<RootStackParamList>;
export type Props = {
  navigation: RootStackNavigationProps;
};
```


```
return (
  <FornecedorProvider>
    <NavigationContainer>
      <PaperProvider>
        <Stack.Navigator>
          <Stack.Screen name="Home" component={HomeScreen} />
          <Stack.Screen name="Cadastro" component={Cadastro} />
          <Stack.Screen name="Lista" component={ListaFornecedores} />
        </Stack.Navigator>
      </PaperProvider>
    </NavigationContainer>
  </FornecedorProvider>
);
```


Envolver toda a aplicação dentro do container de navegação (NavigationContainer) e em cada componente do roteamento passar as props

```
const HomeScreen: React.FC<Props> = ({ navigation }) => {
  return (
```


```
const HomeScreen: React.FC<Props> = ({ navigation }) => {
  return (
```


```
const ListaFornecedores: React.FC<Props> = ({ navigation }) =>
```

Isso é necessário para que o roteador saiba para onde mandar o usuário no momento em que ele clicar para ir para outra parte, neste caso temos 3 telas, mas e se tivéssemos 50, 60 telas, por isso é necessário o roteamento bem armado.
O paper provider que você também pode ver na imagem, é o Provider do React Native Paper, que é uma biblioteca de estilização bem simples para RN

Conclusão

Foi muito desafiador e em alguns momentos desesperador desenvolver essa aplicação, pois nunca eu havia trabalhado com Native antes e agora tive uma baita experiência, tive muito aprendizado, muitos erros, bugs e falhas, mas consegui concluir!