Meeting Solutions

Matheus José Ribeiro de Moura 2023 0713 6158

Polo Rua Tereza - Petrópolis - RJ
Vamos criar um App – Turma 9001 – 2025.1 FLEX

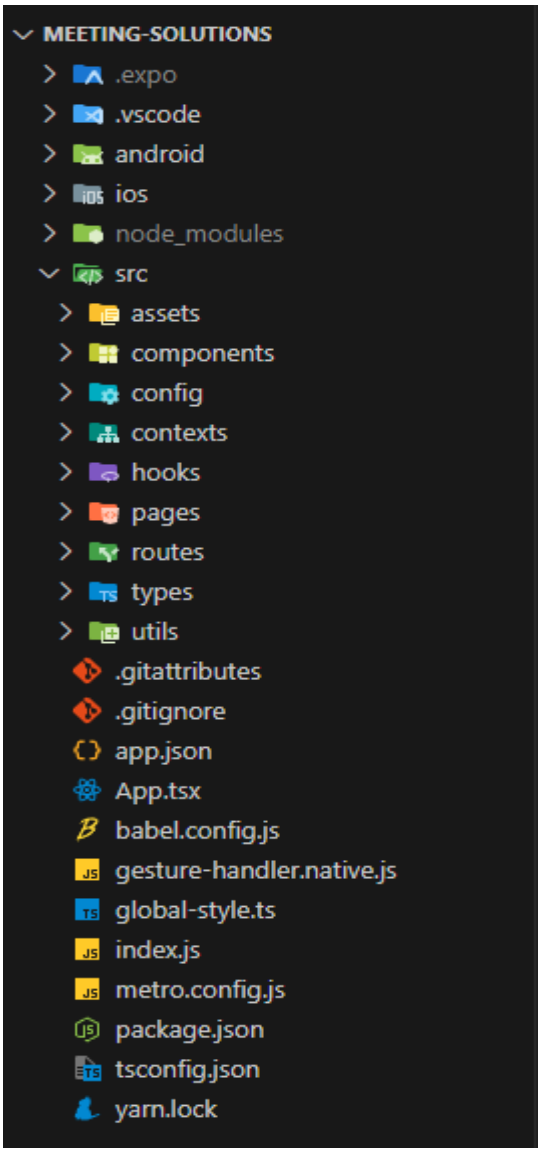
Objetivo da Prática

A prática tem como objetivo criar um app para a empresa “Meeting” em React Native com um cadastro de fornecedores, atualização dos dados dos fornecedores, uma lista de fornecedores cadastrados e upload de imagens de perfil para os fornecedores.

Nesse sentido foi elaborado o app Meeting Solutions, aplicando todas as necessidades de forma intuitiva e bem performática. O projeto foi criado com expo (SDK 52) mas configurado de forma que possua as pastas ios e android para melhor personalização.

Repositóio GIT: https://github.com/MatheusJRM/meeting-solutions

- **Estrutura do projeto:**



- **src** → Armazena todo código específico criado para o app.
- **assets** → Armazena imagens e possíveis ícones da aplicação.
- **components** → Armazena os componentes globais, reutilizáveis, da aplicação.
- **config** → Armazena as configurações necessárias para o projetos, geralmente de bibliotecas externas, no caso do meeting-solutions, ele armazena uma configuração para o typescript reconhecer import de imagens.
- **contexts** → Armazena os contextos, onde a lógica ali contida pode ser acessada por qualquer componente da aplicação.
- **hooks** → Armazena os hooks personalizados da aplicação, pode conter tanto um export mais eficiente dos contexts quanto hooks criados ali mesmo.
- **pages** → Armazena as páginas navegáveis da aplicação, dividas entre publicas e privadas.
- **routes** → Armazena a lógica e as rotas da aplicação, no caso do projeto foi utilizado somente a Stack Navigation.

- **types** → Armazena os tipos da aplicação, seja para estados, tipos de retorno de funções, tipos de paramêtros e etc.

- **utils** → Armazena funções e estados genéricos e reutilizáveis para diferentes locais da aplicação.

- **Bibliotecas externas utilizadas na Aplicação:**

```
"dependencies": {
 "@react-native-async-storage/async-storage": "^2.1.2",
 "@react-navigation/native": "^7.0.15",
 "@react-navigation/stack": "^7.1.2",
 "expo": "~52.0.37",
 "expo-image-picker": "~16.0.6",
 "expo-status-bar": "~2.0.1",
 "react": "18.3.1",
 "react-native": "0.76.7",
 "react-native-gesture-handler": "^2.24.0",
 "react-native-safe-area-context": "^5.3.0",
 "react-native-screens": "^4.9.1"
}
```

- **Contexto async-storage-context.tsx**

A lib @react-native-async-storage/async-storage tem grande importância no fluxo e gerência na persistência de dados da aplicação. O arquivo ***async-storage-context.tsx*** armazena o estado providers utilizado na page home para popular uma lista com os fornecedores cadastrados. Como uma aplicação somente com client-side, o login se da pelo estado isAuthenticated que gerencia quais páginas serão mostrados para o usuário (públicas, caso não estiver autenticado ou privadas caso já esteja).

Ao iniciar o app a função handleFetchData é disparada e é verificado no async-storage os valores de autenticação e dados de fornecedores. Caso exista um valor true salvo de autenticação o usuário é logado imediatamente e direcionado para a primeira tela privada: Home. Caso haja um valor falso, o app mostra a tela *landing,* dando a opção para o usuário entrar no app novamente.

Tendo dados de fornecedores cadastrados, o estado de providers será preenchido, renderizando os cards com as informações dos fornecedores cadastrados da útlima sessão.

- Código correspondente:

```
import { createContext, useCallback, useEffect, useState } from "react";
import AsyncStorage from "@react-native-async-storage/async-storage";
import {
 AsyncStorageProps,
 AsyncStorageProviderProps,
 ProviderDataProps,
} from "types/async-storage-context-types";

const AsyncStorageContext = createContext<AsyncStorageProps | undefined>(
```

```
 undefined
);

const AsyncStorageProvider = ({ children }: AsyncStorageProviderProps) =>
{
 const [providers, setProviders] = useState<ProviderDataProps[]>([]);
 const [isAuthenticated, setIsAuthenticated] = useState<boolean>(false);

 const handleLogin = useCallback(async (): Promise<void> => {
  setIsAuthenticated(true);
  await AsyncStorage.setItem("@isAuthenticated", "true");
 }, []);

 const handleUpdateAllProviders = useCallback(
  async (providersToSave: ProviderDataProps[]): Promise<void> => {
   try {
    await AsyncStorage.setItem(
     "@provider-data",
     JSON.stringify(providersToSave)
    );
    console.info("Providers atualizados com sucesso!");
   } catch (error) {
    console.error("Erro ao atualizar providers:", error);
   }
  },
  []
 );

 const handleUpdateProvider = useCallback(
  async (providerToSave: ProviderDataProps): Promise<void> => {
   try {
    setProviders((prev) => {
     const updatedProviders = prev.map((provider) => {
      if (provider.id === providerToSave.id) return providerToSave;
      return provider;
     });
     handleUpdateAllProviders(updatedProviders);

     return updatedProviders;
    });
    console.info("Providers atualizados com sucesso!");
   } catch (error) {
    console.error("Erro ao atualizar providers:", error);
   }
  },
  []
 );

 const handleFetchData = useCallback(async (): Promise<void> => {
  const isAuthenticatedData = await
AsyncStorage.getItem("@isAuthenticated");
  const data = await AsyncStorage.getItem("@provider-data");
  if (isAuthenticatedData) {
   setIsAuthenticated(JSON.parse(isAuthenticatedData));
```

```
  if (data) {
   setProviders(JSON.parse(data));
  } else {
   setProviders([]);
  }
 }
}, [providers, isAuthenticated]);

const handleAddProvidersData = useCallback(
 async (data: ProviderDataProps): Promise<void> => {
  try {
   setProviders([...providers, data]);
   handleUpdateAllProviders([...providers, data]);
  } catch (error) {
   console.error("Erro ao salvar provider:", error);
  }
 },
 [providers]
);

const handleAddProfileImage = useCallback((id: string, image: string) => {
 setProviders((prev) => {
  const updatedProviders = prev.map((provider) =>
   provider.id === id ? { ...provider, imgPerfil: image } : provider
  );

  handleUpdateAllProviders(updatedProviders);

  return updatedProviders;
 });
}, []);

const handleLogout = useCallback((): void => {
 AsyncStorage.clear().then(() => {
  setIsAuthenticated(false);
  setProviders([]);
 });
}, []);

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

return (
 <AsyncStorageContext.Provider
  value={{
   isAuthenticated,
   providers,
   handleFetchData,
   handleLogin,
   handleAddProvidersData,
   handleUpdateProvider,
   handleAddProfileImage,
   handleLogout,
```

```
   }}
  >
   {children}
  </AsyncStorageContext.Provider>
 );
};

export { AsyncStorageContext, AsyncStorageProvider };
```

- **ProviderCard**

Componente global que renderiza nome, cidade, telefone, produtos e imagem de perfil de um determinado fornecedor. Gerencia uma lógica para atualizar a imagem de perfil do usuário ao receber um clique na área onde a imagem é renderizada.

- **Código correspondente**:

```
import React from "react";
import { View, Text, Image, StyleSheet, TouchableOpacity } from "react-
native";
import Entypo from "@expo/vector-icons/Entypo";
import FontAwesome from "@expo/vector-icons/FontAwesome";
import { useAsyncStorageContext } from "hooks/useAsyncStorageContext";
import { ProviderCardProps } from "types/provider-card-types/provider-
card-type";
import { handleUploadImage } from "utils/image-upload";

export const ProviderCard = ({
 providerData,
 handleNavigate,
}: ProviderCardProps) => {
 const { handleAddProfileImage } = useAsyncStorageContext();
 const { id, nome, cidade, telefone, tiposProduto, imgPerfil } =
providerData;

 const handleUpdateProfileImage = async () => {
  const image = await handleUploadImage();
  if (image) {
   handleAddProfileImage(id, image);
  }
 };

 return (
  <View style={styles.card}>
   <TouchableOpacity onPress={handleUpdateProfileImage}>
    {imgPerfil ? (
     <Image source={{ uri: imgPerfil }} style={styles.image} />
    ) : (
     <View style={styles.noImage}>
      <Entypo name="image" size={24} color="black" style={styles.icon} />
      <Text style={styles.noImageText}>
       Clique para inserir uma imagem
      </Text>
```

```
    </View>
   )}
  </TouchableOpacity>

  <View style={styles.infoContainer}>
   <View
    style={{
     flexDirection: "row",
     justifyContent: "space-between",
    }}
   >
    <Text style={styles.name}>{nome}</Text>
    <TouchableOpacity onPress={() => handleNavigate(providerData)}>
     <FontAwesome name="pencil-square-o" size={24} color="black" />
    </TouchableOpacity>
   </View>
   <Text style={styles.text}>Cidade: {cidade}</Text>
   <Text style={styles.text}>Telefone: {telefone}</Text>
   <Text style={styles.text}>Produtos: {tiposProduto.join(", ")}</Text>
  </View>
 </View>
);
};

const styles = StyleSheet.create({
 card: {
  flexDirection: "row",
  alignItems: "center",
  backgroundColor: "#fff",
  borderRadius: 10,
  padding: 15,
  shadowColor: "#000",
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.1,
  shadowRadius: 5,
  elevation: 3,
  gap: 10,
 },
 image: {
  width: 100,
  height: 100,
  borderRadius: 60,
  marginRight: 15,
 },
 noImage: {
  width: 100,
  marginRight: 15,
  alignItems: "center",
 },
 noImageText: {
  flexWrap: "wrap",
  fontSize: 10,
  textAlign: "center",
 },
```

```
 icon: {
  padding: 10,
 },
 infoContainer: {
  flex: 1,
 },
 name: {
  width: "90%",
  fontSize: 18,
  fontWeight: "bold",
  marginBottom: 5,
 },
 text: {
  fontSize: 14,
  color: "#555",
  marginBottom: 3,
 },
});
```

- **Principais telas:**
    - **landing-page:** Primeira tela do sistema, chama a função handleLogin para armazenar o estado true de autenticação no async-storage. É uma tela publica, então todos usuários podem acessar, autenticados ou não.

- **Código correspondente**:

```
import { Image, StyleSheet, Text, View } from "react-native";
import Logo from "assets/images/logo.png";
import { DefaultButton } from "components/default-button/default-button";
import { useAsyncStorageContext } from "hooks/useAsyncStorageContext";

export const LandingPage = () => {
 const { handleLogin } = useAsyncStorageContext();
 return (
  <View style={styles.container}>
   <Image source={Logo} style={styles.logo} />
   <Text style={styles.welcomeText}>Olá Fornecedor, seja bem-vindo!</Text>
   <DefaultButton
    onPress={handleLogin}
    color="#6c7ff0"
    colorPressed="#e4d8d8"
   >
    <Text style={styles.buttonText}>ENTRAR</Text>
   </DefaultButton>
   <Text style={styles.footerText}>
    <Text style={styles.italicText}>Delivered by MatheusJRM</Text>
   </Text>
  </View>
 );
};

const styles = StyleSheet.create({
 container: {
  flex: 1,
  justifyContent: "center",
  alignItems: "center",
  backgroundColor: "#fff",
 },
 logo: {
  tintColor: "#2f4cf0",
  marginBottom: 20,
 },
 welcomeText: {
  fontSize: 18,
  fontWeight: "600",
  color: "#333",
 },
 buttonText: {
  color: "#fff",
  fontSize: 16,
  fontWeight: "600",
 },
 footerText: {
  position: "absolute",
  bottom: 20,
  fontSize: 14,
  color: "#888",
 },
```

```
 italicText: {
  fontStyle: "italic",
 },
});
```

- **home-page**: Primeira tela privada do sistema, concentra a listagem de fornecedores renderizada pela FlatList por meio do componente ProviderCard, a navegação para as telas de registro de fornecedores no Header e atualização dos dados de um fornecedor pelo ProviderCard e o TextInput de pesquisa para filtrar os cards por *nome, cidade sede e produtos.* Caso haja algum fornecedor cadastrado o ProviderCard será renderizado e nele é possível editar a imagem de perfil do fornecedor em questão e, caso o fornecedor não tenha uma imagem cadastrada será possível atribuir uma clicando no ícone padrão que aparece no card. Ao clicar na setinha de voltar no header da home, será mostrado um modal alertando que se sair os dados da sessão atual serão excluidos, clicando em "Confirmar" será disparado a função handleLogout, que apagará os dados do async-storage e retornará o estado isAuthenticated para false, renderizando a tela publica *landing*, fazendo-se necessário um novo login.

← Home

Cadastro de fornecedores

Pesquisar

Coca-cola

Cidade: Atlanta

Telefone: (99) 99999-9999

Produtos: Refrigerantes

Apple

Cidade: Cupertino

Telefone: (88) 88888-8888

Produtos: Eletrônicos, entretenimento

Google

Cidade: Mountain View

Telefone: (77) 77777-7777

Produtos: Busca, mapas, publicidade, aplicativos, plataformas

Steam

Cidade: Bellevue

← Home

Cadastro de fornecedores

Pesquisar

Telefone: (77) 77777-7777

Produtos: Busca, mapas, publicidade, aplicativos, plataformas

Steam

Cidade: Bellevue

Telefone: (66) 66666-6666

Produtos: Jogos digitais

Microsoft

Cidade: Redmond

Telefone: (55) 55555-5555

Produtos: Eletrônicos, plataformas

Estacio

Cidade: Rio de Janeiro

Telefone: (44) 44444-4444

Produtos: Cursos presenciais, cursos a distância

← Home

Cadastro de fornecedores

Pesquisar

Telefone: (77) 77777-7777

Produtos: Busca, mapas, publicidade, aplicativos, plataformas

Steam

Cidade: Bellevue

Telefone: (66) 66666-6666

Produtos: Jogos digitais

Clique para inserir uma imagem

Microsoft

Cidade: Redmond

Telefone: (55) 55555-5555

Produtos: Eletrônicos, plataformas

Estacio

Cidade: Rio de Janeiro

Telefone: (44) 44444-4444

Produtos: Cursos presenciais, cursos a distância

← Home

Cadastro de fornecedores

eletrônicos

Apple

Cidade: Cupertino

Telefone: (88) 88888-8888

Produtos: Eletrônicos, entretenimento

Clique para inserir uma imagem

Microsoft

Cidade: Redmond

Telefone: (55) 55555-5555

Produtos: Eletrônicos, plataformas

← Home

Cadastro de fornecedores ⊕

atlanta

Coca-cola

Cidade: Atlanta

Telefone: (99) 99999-9999

Produtos: Refrigerantes

← Home

Cadastro de fornecedores ⊕

steam

Steam

Cidade: Bellevue

Telefone: (66) 66666-6666

Produtos: Jogos digitais

- **Código correspondente**:

```
import { useCallback, useState } from "react";
import {
 FlatList,
 NativeSyntheticEvent,
 StyleSheet,
 Text,
 TextInput,
 TextInputChangeEventData,
 TouchableOpacity,
 View,
} from "react-native";
import AntDesign from "@expo/vector-icons/AntDesign";
import { ProviderCard } from "components/provider-card/provider-card";
import { useAsyncStorageContext } from "hooks/useAsyncStorageContext";
import { HomePageProps } from "types/pages-types";
import { ProviderDataProps } from "types/async-storage-context-types";
import { handleSeachProvider } from "utils/provider-search";
import { useFocusEffect } from "@react-navigation/native";

export const HomePage = ({ navigation }: HomePageProps) => {
 const { providers } = useAsyncStorageContext();
 const [searchTerm, setSearchTerm] = useState("");

  const filteredProviders = useMemo<ProviderDataProps[]>(
  () =>
   providers.filter((provider) => handleSeachProvider(provider,
searchTerm)),
  [providers, searchTerm]
 );


 const handleChangeSearchTerm = useCallback(
  (e: NativeSyntheticEvent<TextInputChangeEventData>) => {
   setSearchTerm(e.nativeEvent.text);
  },
  [searchTerm]
 );

 const handleNavigateRegisterProviderPage = useCallback(
  () => navigation.navigate("registerProvider"),
  []
 );

 const handleNavigateUpdateProviderPage = useCallback(
  (providerData: ProviderDataProps) =>
   navigation.navigate("updateProvider", { providerData: providerData }),
  []
 );

 useFocusEffect(useCallback(() => setSearchTerm(""), []));

 return (
```

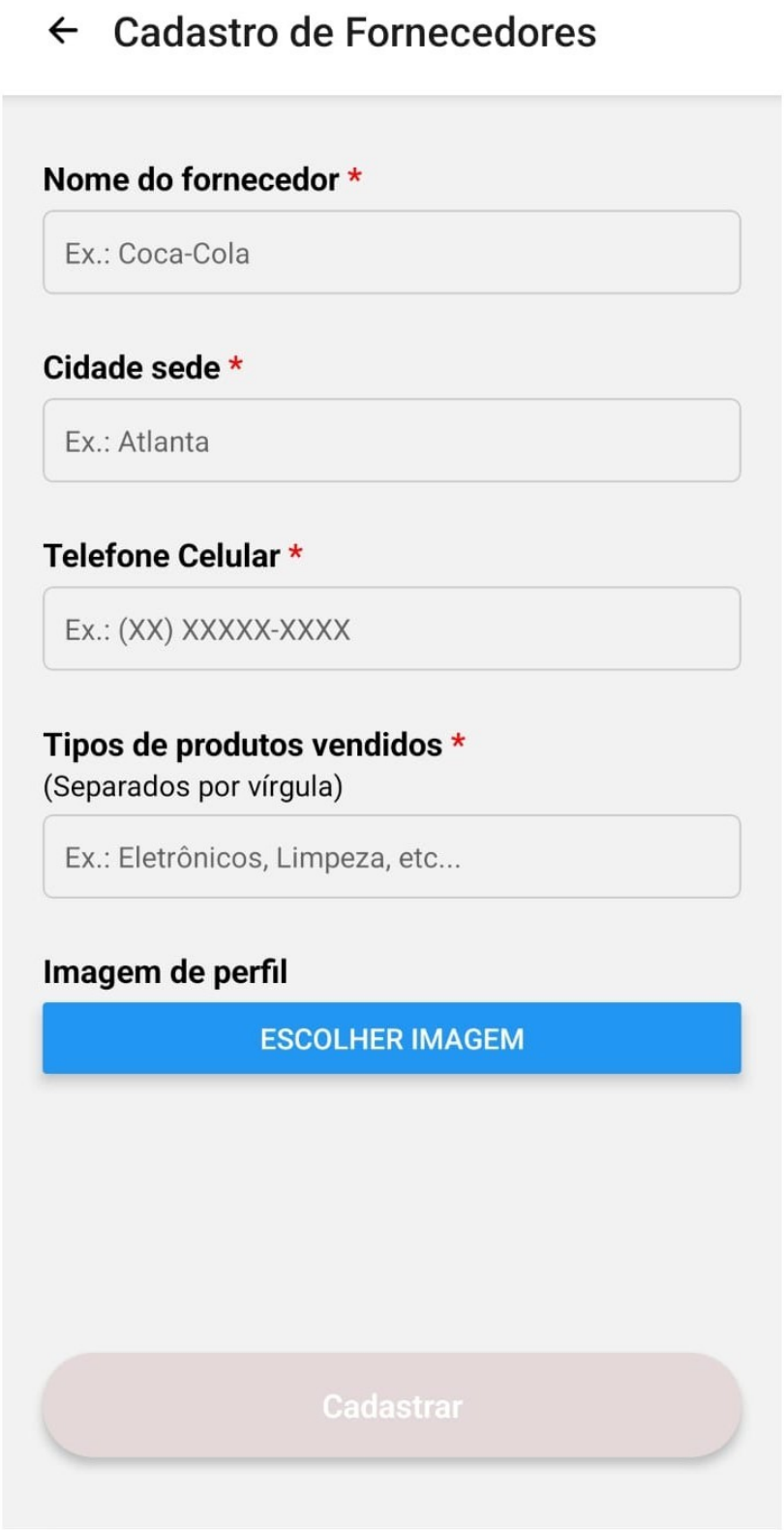
```
  <View style={styles.container}>
   <View style={styles.topContent}>
    <View style={styles.registerProviderContainer}>
     <Text style={styles.topContentText}>Cadastro de fornecedores</Text>
     <TouchableOpacity
      style={{ paddingRight: 15 }}
      onPress={handleNavigateRegisterProviderPage}
     >
      <AntDesign name="pluscircleo" size={24} color="black" />
     </TouchableOpacity>
    </View>
    <View style={styles.searchContainer}>
     <AntDesign name="search1" size={24} color="black" />
     <TextInput
      autoCapitalize="none"
      style={styles.searchInput}
      placeholder="Pesquisar"
      placeholderTextColor="#999"
      onChange={handleChangeSearchTerm}
      value={searchTerm}
     />
    </View>
   </View>

   {filteredProviders.length > 0 && (
    <FlatList
     data={filteredProviders}
     keyExtractor={(item) => item.id}
     contentContainerStyle={styles.flatListContainer}
     showsVerticalScrollIndicator={false}
     renderItem={({ item }) => (
      <ProviderCard
       key={item.id}
       providerData={item}
       handleNavigate={handleNavigateUpdateProviderPage}
      />
     )}
    />
   )}
  </View>
 );
};

const styles = StyleSheet.create({
 container: {
  flex: 1,
  padding: 15,
  gap: 15,
 },
 topContent: {
  gap: 15,
 },
 registerProviderContainer: {
  flexDirection: "row",
```

```
  justifyContent: "space-between",
  alignItems: "center",
 },
 topContentText: {
  fontSize: 18,
  fontWeight: "600",
  fontStyle: "italic",
 },
 searchContainer: {
  flexDirection: "row",
  alignItems: "center",
  backgroundColor: "#f0f0f0",
  borderRadius: 25,
  paddingHorizontal: 15,
  paddingVertical: 10,
  shadowColor: "#000",
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.1,
  shadowRadius: 4,
  elevation: 3,
 },
 searchInput: {
  flex: 1,
  marginLeft: 10,
  fontSize: 16,
  color: "#333",
 },
 flatListContainer: {
  flexGrow: 1,
  gap: 20,
 },
});
```

- **register-provider-page**: Concentra um formulário onde o usuário pode cadastrar um fornecedor preenchendo obrigatoriamente nome, cidade, telefone e produtos vendidos. Pode-se também relacionar uma imagem de perfil para o fornecedor por meio do uso da lib expo-image-picker. O componente possui uma verificação para habilitar o botão de cadastro após todos os campos necessários forem preenchidos corretamente e um validador de telefone celular, para sempre validar no formato (xx) xxxxx-xxxx.



- **Código correspondente**:

```
import React, { useCallback, useState } from "react";
import {
 View,
```

```
 Text,
 TextInput,
 Button,
 ScrollView,
 Alert,
 Image,
 TouchableOpacity,
} from "react-native";
import FontAwesome from "@expo/vector-icons/FontAwesome";
import { RegisterProviderPageProps } from "types/pages-types";
import { DefaultButton } from "components/default-button/default-button";
import { handleUploadImage } from "utils/image-upload";
import { useAsyncStorageContext } from "hooks/useAsyncStorageContext";
import { ProviderDataProps } from "types/async-storage-context-types";
import { formatPhone, isPhoneComplete } from "utils/phone-format-
functions";
import { isDisable } from "utils/form-provider-disable-check";
import { styles } from "utils/register-and-update-style";

export const RegisterProviderPage = ({
 navigation,
}: RegisterProviderPageProps) => {
 const { handleAddProvidersData } = useAsyncStorageContext();
 const [name, setName] = useState("");
 const [city, setCity] = useState("");
 const [phone, setPhone] = useState("");
 const [products, setProducts] = useState("");
 const [profileImage, setProfileImage] = useState<string | null>(null);

 const handleAddImage = useCallback(async (): Promise<void> => {
  const image = await handleUploadImage();
  setProfileImage(image);
 }, []);

 const handleRemoveImage = useCallback((): void => {
  setProfileImage(null);
 }, []);

 const handleRegister = useCallback(() => {
  const formattedProducts = products
   .split(",")
   .map((product) => product.trim())
   .filter((product) => product.length > 0);

  const providerData: ProviderDataProps = {
   id: String(Math.random()),
   nome: name.trim(),
   cidade: city.trim(),
   telefone: phone.trim(),
   tiposProduto: formattedProducts,
   imgPerfil: profileImage,
  };

  handleAddProvidersData(providerData).then(() => {
```

```
  Alert.alert("Sucesso", "Fornecedor cadastrado com sucesso!");
  console.info("Fornecedor cadastrado: ", providerData);
  navigation.navigate("home");
 });
}, [name, city, phone, products, profileImage]);

return (
 <ScrollView contentContainerStyle={styles.container}>
  <Text style={styles.label}>
   Nome do fornecedor <Text style={styles.icon}>*</Text>
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Ex.: Coca-Cola"
   value={name}
   onChangeText={setName}
  />

  <Text style={styles.label}>
   Cidade sede <Text style={styles.icon}>*</Text>
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Ex.: Atlanta"
   value={city}
   onChangeText={setCity}
  />

  <Text style={styles.label}>
   Telefone Celular <Text style={styles.icon}>*</Text>
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Ex.: (XX) XXXXX-XXXX"
   value={phone}
   onChangeText={(text) => setPhone(formatPhone(text))}
   keyboardType="phone-pad"
   maxLength={15}
  />
  {!!phone && !isPhoneComplete(phone) && (
   <Text style={{ color: isPhoneComplete(phone) ? "green" : "red" }}>
    Número incompleto
   </Text>
  )}

  <Text style={styles.label}>
   Tipos de produtos vendidos <Text style={styles.icon}>*</Text>
  </Text>
  <Text
   style={{
    fontSize: 14,
   }}
  >
   (Separados por vírgula)
```

```
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Ex.: Eletrônicos, Limpeza, etc..."
   value={products}
   onChangeText={setProducts}
  />

  <Text style={styles.label}>Imagem de perfil</Text>
  <View style={styles.imageButton}>
   <Button title="Escolher imagem" onPress={handleAddImage} />
  </View>
  {profileImage && (
   <>
    <TouchableOpacity onPress={handleRemoveImage}>
     <FontAwesome
      name="remove"
      size={24}
      color="black"
      style={styles.removeIcon}
     />
    </TouchableOpacity>
    <Image source={{ uri: profileImage }} style={styles.image} />
   </>
  )}

  <View style={styles.buttonContainer}>
   <DefaultButton
    onPress={handleRegister}
    color="#6c7ff0"
    colorPressed="#e4d8d8"
    disabled={isDisable(name, city, phone, products)}
   >
    <Text style={styles.buttonText}>Cadastrar</Text>
   </DefaultButton>
  </View>
 </ScrollView>
 );
};
```

- **update-provider-page:** Concentra um formulário onde o usuário pode atualizar um fornecedor existente preenchendo obrigatoriamente nome, cidade, telefone e produtos vendidos. Pode-se também relacionar uma imagem de perfil para o fornecedor por meio do uso da lib expo-image-picker. O componente possui uma verificação para habilitar o botão de cadastro após todos os campos necessários forem preenchidos corretamente e um validador de telefone celular, para sempre validar no formato (xx) xxxxx-xxxx. Ao entrar na tela os dados previamente cadastrados são automaticamente preenchidos pelo estado *providerData* passado como parâmetro na rota (navegação realizada ao clicar no ícone de lápis dentro do ProviderCard).

← Atualização de Fornecedor

Nome do fornecedor *

Apple

Cidade sede *

Cupertino

Telefone *

(88) 88888-8888

Tipos de produtos vendidos *
(Separados por vírgula)

Eletrônicos, entretenimento

Imagem de perfil

ESCOLHER IMAGEM

Atualizar

- **Código correspondente:**

```
import React, { useCallback, useState } from "react";
import {
 View,
 Text,
 TextInput,
 Button,
 ScrollView,
 Alert,
 Image,
 TouchableOpacity,
} from "react-native";
import FontAwesome from "@expo/vector-icons/FontAwesome";
import { UpdateProviderPageProps } from "types/pages-types";
import { DefaultButton } from "components/default-button/default-button";
import { handleUploadImage } from "utils/image-upload";
import { useAsyncStorageContext } from "hooks/useAsyncStorageContext";
import { ProviderDataProps } from "types/async-storage-context-types";
import { isDisable } from "utils/form-provider-disable-check";
import { styles } from "utils/register-and-update-style";

export const UpdateProviderPage = ({
 navigation,
 route,
}: UpdateProviderPageProps) => {
 const { providerData } = route.params;
 const { handleUpdateProvider } = useAsyncStorageContext();
 const [name, setName] = useState(providerData.nome);
 const [city, setCity] = useState(providerData.cidade);
 const [phone, setPhone] = useState(providerData.telefone);
 const [products, setProducts] = useState(
  providerData?.tiposProduto.join(", ")
 );
 const [profileImage, setProfileImage] = useState<string | null>(
  providerData.imgPerfil || null
 );

 const handleAddImage = useCallback(async (): Promise<void> => {
  const image = await handleUploadImage();
  setProfileImage(image);
 }, []);

 const handleRemoveImage = useCallback((): void => {
  setProfileImage(null);
 }, []);

 const handleUpdate = useCallback(() => {
  const formattedProducts = products
   .split(",")
   .map((product) => product.trim())
   .filter((product) => product.length > 0);
```

```
 const providerDataToSend: ProviderDataProps = {
  id: providerData.id,
  nome: name.trim(),
  cidade: city.trim(),
  telefone: phone.trim(),
  tiposProduto: formattedProducts,
  imgPerfil: profileImage,
 };

 handleUpdateProvider(providerDataToSend).then(() => {
  Alert.alert("Sucesso", "Fornecedor atualizado com sucesso!");
  console.info("Fornecedor atualizado: ", providerDataToSend);
  navigation.navigate("home");
 });
}, [name, city, phone, products, profileImage]);

return (
 <ScrollView contentContainerStyle={styles.container}>
  <Text style={styles.label}>
   Nome do fornecedor <Text style={styles.icon}>*</Text>
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Ex.: Coca-Cola"
   value={name}
   onChangeText={setName}
  />

  <Text style={styles.label}>
   Cidade sede <Text style={styles.icon}>*</Text>
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Ex.: Atlanta"
   value={city}
   onChangeText={setCity}
  />

  <Text style={styles.label}>
   Telefone <Text style={styles.icon}>*</Text>
  </Text>
  <TextInput
   style={styles.input}
   placeholder="Digite o telefone"
   value={phone}
   onChangeText={setPhone}
   keyboardType="phone-pad"
  />

  <Text style={styles.label}>
   Tipos de produtos vendidos <Text style={styles.icon}>*</Text>
  </Text>
  <Text
   style={{
```

```
      fontSize: 14,
     }}
    >
     (Separados por vírgula)
    </Text>
    <TextInput
     style={styles.input}
     placeholder="Ex.: Eletrônicos, Limpeza, etc..."
     value={products}
     onChangeText={setProducts}
    />

    <Text style={styles.label}>Imagem de perfil</Text>
    <View style={styles.imageButton}>
     <Button title="Escolher imagem" onPress={handleAddImage} />
    </View>
    {profileImage && (
     <>
      <TouchableOpacity onPress={handleRemoveImage}>
       <FontAwesome
        name="remove"
        size={24}
        color="black"
        style={styles.removeIcon}
       />
      </TouchableOpacity>
      <Image source={{ uri: profileImage }} style={styles.image} />
     </>
    )}

    <View style={styles.buttonContainer}>
     <DefaultButton
      onPress={handleUpdate}
      color="#6c7ff0"
      colorPressed="#e4d8d8"
      disabled={isDisable(name, city, phone, products)}
     >
      <Text style={styles.buttonText}>Atualizar</Text>
     </DefaultButton>
    </View>
   </ScrollView>
  );
};
```

- **Rotas**
  - **stack.routes.ts:** Concentra a lógica de rotas Stack disponíveis pela lib react-navigation. Como na aplicação é utilizado o header padrão da Stack Route, um hook que gerencia os estados do modal de alerta da aplicação é usado para poder ser acessado na função headerLeftRender e passar uma função personalizada a ele, permitindo um comportamento diferente para cada tela que fosse necessário.

- **Código correspondente**:

```
import React, { useCallback, useMemo } from "react";
import { Pressable } from "react-native";
import { useNavigation } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
import Ionicons from "@expo/vector-icons/Ionicons";
import { WarningModal } from "components/modals/warning-modal";
import { useAsyncStorageContext } from "hooks/useAsyncStorageContext";
import { useWarningModal } from "hooks/useWarningModal";
import { HomePage } from "pages/private/home-page/home-page";
import { LandingPage } from "pages/public/landing-page/landing-page";
import { UpdateProviderPage } from
"pages/private/update-provider-page/update-provider-page";
import { RegisterProviderPage } from "pages/private/register-provider-
page/register-provider-page";
import { RootStackParamList } from "types/navigation-types";
import { globalStyle } from "../../global-style";

const Stack = createStackNavigator<RootStackParamList>();

export const StackRoutes = () => {
 const { goBack } = useNavigation();
 const { isAuthenticated, handleLogout } = useAsyncStorageContext();
 const { warningModalIsVisible, handleCloseModal, handleShowModal } =
  useWarningModal();

 const handleConfirmFunction = useCallback(() => {
  handleLogout();
  handleCloseModal();
 }, []);

 const handleHeaderLeftRender = useCallback(
  (onPress: () => void) => (
   <Pressable
    style={({ pressed }) => [
     globalStyle.button,
     pressed && globalStyle.buttonPressed,
    ]}
    onPress={onPress}
   >
    <Ionicons name="arrow-back" size={20} color="black" />
   </Pressable>
  ),
  []
 );

 const publicRoutes = useMemo(
  () => (
   <Stack.Screen
    name="landing"
    component={LandingPage}
    options={{ headerShown: false }}
   />
```

```
  ),
  []
 );

 const privateRoutes = useMemo(
  () => (
   <React.Fragment>
    <Stack.Screen
     name="home"
     component={HomePage}
     options={{
      headerTitle: "Home",
      headerLeft: () => handleHeaderLeftRender(handleShowModal),
      gestureEnabled: false,
     }}
    />
    <Stack.Screen
     name="registerProvider"
     component={RegisterProviderPage}
     options={{
      headerTitle: "Cadastro de Fornecedores",
      headerLeft: () => handleHeaderLeftRender(goBack),
      gestureEnabled: false,
     }}
    />
    <Stack.Screen
     name="updateProvider"
     component={UpdateProviderPage}
     options={{
      headerTitle: "Atualização de Fornecedor",
      headerLeft: () => handleHeaderLeftRender(goBack),
      gestureEnabled: false,
     }}
    />
   </React.Fragment>
  ),
  [warningModalIsVisible]
 );

 return (
  <>
   <Stack.Navigator>
    {isAuthenticated ? privateRoutes : publicRoutes}
   </Stack.Navigator>
   <WarningModal
    visible={warningModalIsVisible}
    onClose={handleCloseModal}
    title="Deseja mesmo sair?"
    content="Ao sair todos cadastros serão perdidos. Se quiser manter os
dados apenas feche o app, sem sair."
    buttonTextCancel="Cancelar"
    buttonTextConfirm="Confirmar"
    handleActionCancel={handleCloseModal}
    handleActionConfirm={handleConfirmFunction}
```

```
   />
  </>
 );
};
```

- **Função providerSearch**

A função providerSearch é chamada na tela home e realiza um filtro entre os itens da FlatList (ProviderCard). A função recebe os dados de um provider e um “termo de pesquisa”. Ela formata esse termo para remover caracteres especiais, espaços em branco, acentos e converter para minúsculo, depois o compara com os dados do provider (nome, cidade e produtos vendidos). Essa função retorna um boolean que é usado para filtrar os providers correspondentes ao termo de pesquisa na página home.

- **Código correspondente**:

```
import { ProviderDataProps } from "types/async-storage-context-types";

const normalizeString = (str: string) => {
 return str
  .normalize("NFD") // Normaliza caracteres acentuados (ex: "é" -> "e")
  .replace(/[\u0300-\u036f]/g, "") // Remove diacríticos (acentos)
  .replace(/[^a-zA-Z0-9]/g, "") // Remove caracteres especiais e espaços
  .toLowerCase(); // Converte para minúsculas
};

export const handleSeachProvider = (
 provider: ProviderDataProps,
 searchTerm: string
) => {
 const normalizedString = normalizeString(searchTerm);

 return (
  normalizeString(provider.nome).toLowerCase().includes(normalizedString)
||
  normalizeString(provider.cidade).toLowerCase().includes(normalizedString)
||
  provider.tiposProduto.some((produto) =>
   normalizeString(produto).toLowerCase().includes(normalizedString)
  )
 );
};
```