



Estácio

Campus: Polo Centro II - Guarulhos - SP

Curso: Desenvolvimento full stack

Disciplina: RPG0023 - Vamos criar um app!

Turma: 2025.1

Aluno: Pedro Wilson Araújo Avilar

Matrícula: 2023 0916 8251

- **Título**

Missão Prática | Mundo 4 | Nível 1

- **Material de apoio**

<https://sway.cloud.microsoft/s/0ATSB44zhZMGtTAj/embed>

- **Objetivo**

O aplicativo de cadastro de fornecedores foi desenvolvido, em React Native, com o objetivo de facilitar o gerenciamento de fornecedores, permitindo que os usuários cadastrem, editem, excluam e busquem fornecedores em uma lista organizada com informações detalhadas, praticidade, eficiência e economizando recursos.

- **Repositório git**

https://github.com/PedroAvilar/Cadastro_Fornecedores

- **Tecnologias utilizadas**

- React Native: para o desenvolvimento da aplicação.
- Expo Router: para gerenciar rotas e navegação entre telas.
- Context API: para o gerenciamento global do estado dos fornecedores.
- React Hooks: para manipulação de estado e efeitos colaterais.
- Expo ImagePicker: para a seleção de imagem da galeria do usuário.
- StyleSheets: para estilização dos componentes.

- **Funcionalidades implementadas**

- Listagem de fornecedores: exibir a lista de fornecedores cadastrados.
- Cadastro de fornecedores: usando um formulário para inserir os dados, incluindo uma imagem.
- Pesquisa de fornecedores: por meio de uma caixa de texto, exibir os fornecedores cadastrados por qualquer informação cadastrada.
- Edição de fornecedores: modificar um ou mais dados, ou imagem, de algum fornecedor cadastrado.
- Exclusão de fornecedores: excluir um fornecedor da lista após a confirmação do usuário para a ação.

- **Descrição dos arquivos e pastas**

Abaixo está a estrutura de diretórios e arquivos principais:

Cadastro_Fornecedores/

```
| -- app/ → Contém as telas principais do aplicativo.  
| | -- cadastro.js → Tela com um formulário para cadastrar um fornecedor.  
| | -- editar.js → Tela para editar um fornecedor que está cadastrado.  
| | -- index.js → Tela inicial para listagem dos fornecedores.  
| -- contexto/ → Contém o contexto global da aplicação.  
| | -- contextoFornecedores.tsx → Gerenciamento dos fornecedores.  
| -- readme_expo/ → Contém o README padrão do aplicativo.  
| | -- README.md → Comandos para instalar dependências e executar.  
| -- styles/ → Contém os estilos globais do aplicativo.  
| | -- styles.js → Define os estilos usados em todas as telas.  
| -- utils/ → Contém as funções auxiliares.  
| | -- formatarTelefone.js → Função para formatar números de telefone.  
| | -- selecaoImagem.js → Função para selecionar imagens da galeria.
```

- **app/index.js**

Este arquivo é responsável pela tela inicial do aplicativo, onde todos os fornecedores cadastrados são listados, também sendo possível realizar buscas, navegar para a tela de cadastro ou de editar e excluir um fornecedor existente.

Principais funcionalidades:

- **Acesso aos dados via contexto:**

```
const {fornecedores, excluirFornecedor} = useFornecedores();
```

Utiliza o hook **useFornecedores** para acessar os dados e as funções de controle.

- **Busca/filtragem de fornecedores:**

```
const [filtro, setFiltro] = useState("");
```

Usa **filtro** como o estado que guarda o texto digitado pelo usuário para a busca no **TextInput**.

```
<TextInput
  style={styles.input}
  placeholder="Nome, telefone, categoria ou localização"
  value={filtro}
  onChangeText={setFiltro}
/>
```

Permite a busca dinâmica por qualquer informação presente nos fornecedores.

```
const fornecedoresFiltrados = fornecedores.filter(fornecedor => (
  fornecedor.nome.toLowerCase().includes(filtro.toLowerCase()) ||
  fornecedor.telefone.toLowerCase().includes(filtro.toLowerCase()) ||
  fornecedor.endereco.toLowerCase().includes(filtro.toLowerCase()) ||
  fornecedor.categoria.toLowerCase().includes(filtro.toLowerCase())
))
```

Retorna apenas os fornecedores que contêm o texto digitado em algum dos campos, ignorando maiúsculas e minúsculas com o uso do **toLowerCase()**.

○ Lista de fornecedores:

```
{fornecedoresFiltrados.length === 0 ? (
  <Text style={[styles.text, {padding: 30}]}>Nenhum fornecedor
  encontrado.</Text>
) : (
```

Caso nenhum fornecedor esteja cadastrado, ou a busca feita pelo usuário não encontre correspondência, é exibido a mensagem informando “Nenhum fornecedor encontrado.”

```
<FlatList
  style={{width: '100%'}}
  data={fornecedoresFiltrados}
  keyExtractor={(item) => item.id.toString()}
  renderItem={({item}) => (
```

A lista é exibida com **FlatList**, e tem como fonte de dados **fornecedoresFiltrados**, afetado pelo **TextInput** de busca caso haja texto digitado, não tendo, retorna a lista total de fornecedores.

```
<Image source={{uri: item.imagemUri}} style={styles.fotoPerfil}/>
<View>
  <Text style={styles.textNome}>👤 {item.nome}</Text>
  <Text style={styles.text}>☎ {item.telefone}</Text>
  <Text style={styles.text}>📍 {item.endereco}</Text>
  <Text style={styles.text}>📦 {item.categoria}</Text>
</View>
```

Para cada fornecedor é exibido a imagem de perfil usando **Image**, e as informações do fornecedor pelo props, por exemplo, **{item.nome}**, cada uma em um **Text**.

- **Navegar para a tela de editar:**

```
<TouchableOpacity
  style={[styles.botaoBase, styles.botaoEditar]}
  onPress={() => router.push(`/editar?id=${item.id}`)}>
  <Text style={styles.textBotao}>Editar</Text>
</TouchableOpacity>
```

Abaixo de cada fornecedor, exibido pela lista em **FlatList**, temos um botão escrito editar, esse botão usa o **router** do **expo-router**, para navegar para a tela de edição em **app/editar.js** passando o **id** do fornecedor a ser editado pela URL.

```
import { useRouter } from "expo-router";
const router = useRouter();
```

- **Exclusão com confirmação:**

```
<TouchableOpacity
  style={[styles.botaoBase, styles.botaoExcluir]}
  onPress={() => confirmarExclusao(item.id)}>
  <Text style={styles.textBotao}>Excluir</Text>
</TouchableOpacity>
```

Ao lado do botão de editar, abaixo de cada fornecedor em **FlatList**, fica o botão escrito excluir, ao ser pressionado, ele invoca a função **confirmarExclusao** passando o **id** do fornecedor.

```
const confirmarExclusao = (id) => {
  if (Platform.OS === 'web') {
    const confirmacao = window.confirm("Tem certeza que deseja excluir?");
    if (confirmacao) {
      excluirFornecedor(id);
    }
  } else {
    Alert.alert(
      "Excluir Fornecedor",
      "Tem certeza que deseja excluir?",
      [
        {text: "Cancelar", style: 'cancel'},
        {text: "Excluir", onPress: () => excluirFornecedor(id),
style: 'destructive'}
      ]
    );
  }
}
```

```

    ]
  )
}
}

```

Essa função recebe o **id** e exibe um alerta, dependendo da plataforma do usuário, pedindo a confirmação do usuário para a exclusão. Tendo a confirmação ele invoca **excluirFornecedor** repassando o **id** onde será excluído em **contexto/contextoFornecedores.tsx**.

- **app/cadastro.js**

Este arquivo é responsável pela tela de cadastro de um novo fornecedor, permitindo ao usuário preencher os dados e associar uma imagem de perfil antes de salvar.

Principais funcionalidades:

- **Gerenciamento dos estados locais:**

```

const [nome, setNome] = useState("");
const [telefone, setTelefone] = useState("");
const [endereco, setEndereco] = useState("");
const [categoria, setCategoria] = useState("");
const [imagemUri, setImagemUri] = useState("");

```

Armazena localmente os dados digitados nos campos do formulário de cadastro.

- **Acesso ao contexto global:**

```

const {adicionarFornecedor} = useFornecedores();

```

Utiliza o **useFornecedor** para acessar a função **adicionarFornecedor**, que insere o novo fornecedor no estado global.

- **Navegação:**

```

const router = useRouter();

```

Permite redirecionar para outras telas, como voltar para a tela de listagem clicando no botão voltar ou quando cadastrar um novo fornecedor.

```

<TouchableOpacity
  style={[styles.botaoBase, styles.botaoVoltar]}
  onPress={() => router.push('/')}>
  <Text style={styles.textBotao}>Voltar</Text>
</TouchableOpacity>

```

O botão de voltar fica na parte de cima da página, também disponível na página de edição de um fornecedor, em **app/editar.js**.

- **Função para salvar um fornecedor:**

```
const salvarFornecedor = () => {  
  if (!nome || !telefone || !endereco || !categoria) {  
    return alert('Por favor, preencha todos os campos.');  }  
  
  adicionarFornecedor({nome, telefone, endereco, categoria, imagemUri});  
  
  setNome('');  
  setTelefone('');  
  setEndereco('');  
  setCategoria('');  
  setImagemUri('');  
  
  router.push('/');  
}
```

Verifica se todos os campos foram preenchidos, caso sim, chama **adicionarFornecedor** passando os dados para o contexto. Após o cadastro, os campos são limpos e o usuário é redirecionado para a tela inicial de listagem pelo **router** da navegação.

- **Botão de imagem:**

```
{imagemUri ?  
  <Image source={{uri: imagemUri}} style={styles.fotoPerfil}/> : null}  
<TouchableOpacity  
  style={[styles.botaoBase, styles.botaoEditar]}  
  onPress={() => selecionarImagem(setImagemUri)}>  
  <Text style={styles.textBotao}>Nova</Text>  
</TouchableOpacity>
```

Com o operador ternário é definido explicitamente o que acontece quando **imagemUri** não está definida, sendo um retorno **null**, não haverá renderização. Assim, será renderizado **Image** somente se existir uma URI válida.

O botão permite ao usuário selecionar uma imagem da galeria, usando a função **selecionarImagem** do arquivo **utils/selecaoImagem.js**, que atualiza o estado de **imagemUri** e renderiza **Image**.

- **Campo de telefone com formatação:**

```
<TextInput  
  style={styles.input}
```

```

placeholder="(11)94433-2211"
value={telefone}
onChangeText={(texto) => setTelefone(formatarTelefone(texto))}
keyboardType="numeric"
/>

```

Enquanto o usuário digita o número de telefone ele automaticamente é formatado pela função **formatarTelefone**, em **utils/formatarTelefone.js**. Um campo que permite apenas números, ativando o teclado numérico em dispositivos móveis para ser usado no preenchimento.

- **Campos de entradas de informações:**

```

<Text style={styles.text}>Nome</Text>
<TextInput
  style={styles.input}
  placeholder="Pedro Wilson Araújo Avilar"
  value={nome}
  onChangeText={setNome}
/>
<Text style={styles.text}>Endereço</Text>
<TextInput
  style={styles.input}
  placeholder="Rua Arlindo Braga, 53, Guarulhos - SP"
  value={endereco}
  onChangeText={setEndereco}
/>
<Text style={styles.text}>Categoria</Text>
<TextInput
  style={styles.input}
  placeholder="Eletrodomésticos"
  value={categoria}
  onChangeText={setCategoria}
/>

```

Campos para o usuário inserir dados para nome, endereço e categoria, atualizando os estados locais dos dados.

- **Botão de cadastrar:**

```

<TouchableOpacity
  style={[styles.botaoBase, styles.botaoPrimario]}
  onPress={salvarFornecedor}>

```

```
<Text style={styles.textBotao}>Cadastrar</Text>
</TouchableOpacity>
```

Dispara a função **salvarFornecedor** para concluir o cadastro.

- **app/editar.js**

Este arquivo é responsável por permitir a edição de um fornecedor já cadastrado. Essa tela é aberta com um ID, de um fornecedor, na URL, passada pelo botão editar na tela inicial de listagem. O usuário pode editar os campos ou a imagem e salvar as alterações.

Principais funcionalidades:

- **Acesso ao contexto global:**

```
const {fornecedores, editarFornecedor} = useFornecedores();
```

Utiliza o **useFornecedor** para acessar a função **editarFornecedor**, que altera os dados do fornecedor no estado global.

- **Acesso ao ID via parâmetros da URL:**

```
const {id} = useLocalSearchParams();
```

Utiliza o hook **useLocalSearchParams()** do **expo-router** para acessar o **id** passado pela URL, que identifica qual fornecedor será editado.

- **Localiza o fornecedor pelo ID:**

```
const fornecedor = fornecedores.find(f => f.id === Number(id));
```

Busca o fornecedor dentro do contexto, transformando o **id** da URL para número e localizando com **find()**.

- **Preenche os estados com os dados do fornecedor:**

```
const [nome, setNome] = useState(fornecedor?.nome || "");
const [telefone, setTelefone] = useState(fornecedor?.telefone || "");
const [endereco, setEndereco] = useState(fornecedor?.endereco || "");
const [categoria, setCategoria] = useState(fornecedor?.categoria || "");
const [imagemUri, setImagemUri] = useState(fornecedor?.imagemUri || "");
```

Inicializa os estados dos campos com os dados do fornecedor, caso esteja disponíveis.

```
useEffect(() => {
  if (fornecedor) {
    setNome(fornecedor.nome);
    setTelefone(fornecedor.telefone);
    setEndereco(fornecedor.endereco);
    setCategoria(fornecedor.categoria);
    setImagemUri(fornecedor.imagemUri);
  }
}, [fornecedor]);
```



```
}  
}, [fornecedor])
```

O **useEffect** garante que, quando o fornecedor for carregado, os dados sejam preenchidos corretamente nos campos.

- **Edição da imagem do fornecedor:**

```
{imagemUri ?  
  <Image source={{uri: imagemUri}} style={styles.fotoPerfil}/> : null}  
<TouchableOpacity  
  style={[styles.botaoBase, styles.botaoEditar]}  
  onPress={() => selecionarImagem(setImagemUri)}>  
  <Text style={styles.textBotao}>Nova</Text>  
</TouchableOpacity>
```

Também fazendo uso do operador ternário, sendo que nesse caso renderiza a imagem atual, tendo um botão que aciona a função **selecionarImagem**, passando o **setImagemUri** para atualizar o estado com a nova imagem selecionada.

- **Campos de entrada editáveis:**

```
<Text style={styles.text}>Nome</Text>  
<TextInput  
  style={styles.input}  
  value={nome}  
  onChangeText={setNome}  
>  
<Text style={styles.text}>Telefone</Text>  
<TextInput  
  style={styles.input}  
  value={telefone}  
  onChangeText={(texto) => setTelefone(formatarTelefone(texto))}  
  keyboardType="numeric"  
>  
<Text style={styles.text}>Endereço</Text>  
<TextInput  
  style={styles.input}  
  value={endereco}  
  onChangeText={setEndereco}  
>  
<Text style={styles.text}>Categoria</Text>  
<TextInput
```

```
style={styles.input}
value={categoria}
onChangeText={setCategoria}
/>
```

Exibe os dados atuais e permite ao usuário modificar as informações. O campo de telefone segue o padrão da página de cadastro, invocando a função **formatarTelefone** em **utils/formatarTelefone.js** formatando automaticamente conforme o usuário digita.

- **Salvar as alterações:**

```
const salvarEdicao = () => {
  if (!nome || !telefone || !endereco || !categoria) {
    return alert('Por favor, preencha todos os campos.');
```

```
  }
  editarFornecedor(fornecedor.id, {nome, telefone, endereco, categoria,
imagemUri});
  router.push('/');
}
```

A função valida os campos e estando todos preenchidos, chama **editarFornecedor()** do contexto, passando o ID e os novos dados, para então, redirecionar o usuário de volta para a tela inicial com **router.push('/')**.

- **contexto/contextoFornecedores.tsx**

Este arquivo é responsável por gerenciar o estado global dos fornecedores usando o contexto do React. Ele permite que qualquer parte do aplicativo acesse, edite, adicione ou exclua fornecedores de forma centralizada.

Principais funcionalidades:

- **Definição da interface do fornecedor:**

```
interface Fornecedor {
  id: number;
  nome: string;
  telefone: string;
  endereco: string;
  categoria: string;
  imagemUri: string;
}
```

Define a estrutura esperada para cada fornecedor, garantindo que os dados tenham os tipos corretos com suporte do TypeScript.

- **Interface do contexto:**

```
interface FornecedoresContextType {  
  fornecedores: Fornecedor[];  
  adicionarFornecedor: (novoFornecedor: Omit<Fornecedor, 'id'>) => void;  
  editarFornecedor: (id: number, dadosAtualizados: Omit<Fornecedor, 'id'>) =>  
void;  
  excluirFornecedor: (id: number) => void;  
}
```

Define as funções e os dados que estarão disponíveis no contexto. Com o uso de **Omit<Fornecedor, 'id'>** porque o **id** é gerado internamente no contexto e não fornecido externamente.

- **Criação do contexto:**

```
const FornecedoresContext = createContext<FornecedoresContextType |  
undefined>(undefined);
```

Cria o contexto que armazenará os fornecedores e as funções de manipulação. Inicialmente definido como **undefined** até ser usado por um **Provider**.

- **Provedor do contexto:**

```
export default function FornecedoresProvider({ children }: { children:  
React.ReactNode }) {  
  const [fornecedores, setFornecedores] = useState<Fornecedor[]>([]);  
  ...  
  return (  
    <FornecedoresContext.Provider value={{ fornecedores, adicionarFornecedor,  
editarFornecedor, excluirFornecedor }}>  
      {children}  
    </FornecedoresContext.Provider>  
  );  
}
```

Envolve os componentes da aplicação para que eles tenham acesso ao contexto. O **useState** inicia a lista de **fornecedores** como vazia.

- **Adicionar fornecedor:**

```
const adicionarFornecedor = (novoFornecedor: Omit<Fornecedor, 'id'>) => {  
  const novoId = fornecedores.length > 0 ? Math.max(...fornecedores.map(f  
=> f.id)) + 1 : 1;  
  const fornecedorComId: Fornecedor = {id: novoId, ...novoFornecedor};  
  setFornecedores((prev) => [...prev, fornecedorComId]);  
}
```

O parâmetro **novoFornecedor** recebe um objeto que contém as propriedades de um fornecedor sem o campo **id**, pelo uso do **Omit<Fornecedor, 'id'>**. Em **novoid** o código verifica se há fornecedores na lista, se houver ele calcula o maior **id** existente com **Math.max(...fornecedores.map(f => f.id))** e adiciona **1** ao valor, caso a lista esteja vazia o **id** recebe o valor **1**.

O objeto **fornecedorComId** é criado combinando o **id** calculado com os dados do **novoFornecedor** usando o operador spread **...**.

O estado **setFornecedor** é chamado adicionando o novo fornecedor à lista, pegando o estado anterior com **prev** e adicionando **fornecedorComId** ao array.

- **Editar fornecedor:**

```
const editarFornecedor = (id: number, dadosAtualizados: Omit<Fornecedor, 'id'>) => {
  setFornecedores(prev =>
    prev.map(fornecedor =>
      fornecedor.id === id ? {...fornecedor, ...dadosAtualizados} : fornecedor
    )
  )
}
```

Recebe um **id** como parâmetro, e um objeto contendo os novos dados do fornecedor, excluindo o campo **id**, em **dadosAtualizados: Omit<Fornecedor, 'id'>**, para garantir que o ID original não seja alterado.

Atualiza o estado dos fornecedores com **setFornecedores(prev =>**, percorrendo a lista de fornecedores e modificando o fornecedor correto com **prev.map(fornecedor =>**, verificando se o fornecedor tem o mesmo ID com **fornecedor.id === id**, caso tenha, **{...fornecedor, ...dadosAtualizados}** atualiza os dados do fornecedor para os dados atualizados.

- **Excluir fornecedor:**

```
const excluirFornecedor = (id: number) => {
  setFornecedores(prev =>
    prev.filter(fornecedor => fornecedor.id !== id)
  )
}
```

A função recebe o **id** como parâmetro. O estado da lista de fornecedores **setFornecedor** é atualizado com o estado anterior com **prev** recebendo um novo estado, com **prev.filter**, onde terá apenas os fornecedores que contém um **id** diferente do **id** recebido como parâmetro. O resultado é o estado de fornecedores atualizado com a lista filtrada sem o fornecedor que possuía o **id** passado.

- **Hook personalizado:**

```
export function useFornecedores() {  
  const context = useContext(FornecedoresContext);  
  if (!context) {  
    throw new Error('useFornecedores deve ser usado dentro de FornecedoresProvider');  
  }  
  return context;  
}
```

Facilita o uso do contexto com um hook reutilizável, garantindo que o contexto esteja sendo utilizado dentro do provider, evitando erros.

- **styles/styles.ts**

Este arquivo contém os estilos reutilizáveis usados em todo o aplicativo. Ele centraliza a definição de cores, tamanhos, espaçamentos e formatações visuais com o StyleSheet do React Native, mantendo a consistência e organização do layout da aplicação.

- **Containers principais:**

```
containerScroll: {  
  flex: 1,  
  backgroundColor: '#ede',  
},
```

Usado para as telas do aplicativo, que são envolvidas por **ScrollView**, define altura flexível e uma cor de fundo.

```
container: {  
  flex: 1,  
  alignItems: 'center',  
  paddingTop: 30,  
},
```

Centraliza os elementos na horizontal e adiciona espaço superior.

- **Containers específicos:**

```
containerBotaoEditar: {  
  flexDirection: 'row',  
  justifyContent: 'center',  
  marginBottom: 30,  
},
```

Usado na tela inicial para alinhar os botões de editar e excluir lado a lado.

```
containerBotaoVoltar: {
  alignSelf: 'flex-start',
  marginLeft: 10,
},
```

Alinha o botão Voltar à esquerda da tela.

- **Textos:**

```
title: {
  fontSize: 25,
  fontWeight: 'bold',
  margin: 25,
  color: '#000',
},
```

Usado e títulos das páginas, como por exemplo, “Cadastrar fornecedor”.

```
text: {
  fontSize: 16,
  color: '#000',
},
```

Para textos comuns, como rótulos de campos do formulário.

```
textNome: {
  fontSize: 18,
  color: '#000',
  fontWeight: '500',
},
```

Usado para destacar o nome do fornecedor na listagem.

- **Inputs (campos de entrada de texto):**

```
input: {
  height: 40,
  width: 290,
  borderWidth: 2,
  backgroundColor: '#fff',
  borderColor: '#ccc',
  padding: 10,
  borderRadius: 10,
  marginBottom: 10,
},
```

Estilo uniforme para todos os campos de entrada, largura adaptadas para telas de celular e leve arredondamento das bordas.

- **Imagem de perfil do fornecedor:**

```
fotoPerfil: {  
  margin: 10,  
  width: 200,  
  height: 200,  
  borderRadius: 100,  
},
```

Exibe a imagem do fornecedor com formato circular, por o valor de **borderRadius** ser igual à metade do valor da largura como também da altura.

- **Padrões de botões:**

```
textBotao: {  
  color: '#fff',  
  fontSize: 16,  
  fontWeight: 'bold',  
},
```

Usado para o texto interno dos botões com contraste para botões coloridos por ter a cor branca definida.

```
botaoBase: {  
  height: 40,  
  paddingVertical: 10,  
  paddingHorizontal: 20,  
  borderRadius: 10,  
  alignItems: 'center',  
  justifyContent: 'center',  
  margin: 10,  
  elevation: 3,  
  shadowColor: '#000',  
  shadowOffset: {width: 0, height: 4},  
  shadowOpacity: 0.1,  
  shadowRadius: 6,  
},
```

Estilo base comum para todos os botões, definindo estilos como formato, sombra, espaçamento, altura e alinhamentos, permite o reaproveitamento com diferentes cores que são definidas de modo individual.

- **Variações de botões:**

```
botaoPrimario: {  
  width: 190,  
  backgroundColor: '#3a3',  
  marginBottom: 30,  
},
```

Botão principal para ações de “Adicionar”, na tela de listagem, “Salvar, em editar, e “Cadastrar” abaixo do formulário na página de cadastro e largura maior.

```
botaoVoltar: {  
  backgroundColor: '#888',  
  width: 100,  
},
```

Estilo para botão de navegação com cor neutra, diferenciando de ações principais e largura menor.

```
botaoEditar: {  
  backgroundColor: '#35f',  
  width: 100,  
},
```

Botão de edição com cor azul e largura igual ao botão “Voltar”.

```
botaoExcluir: {  
  backgroundColor: '#f33',  
  width: 100,  
},
```

Botão de exclusão com cor vermelha e largura igual ao botão “Voltar”.

- **utils/formatarTelefone.js**

Este arquivo exporta uma função utilitária que formata dinamicamente números de telefone digitados pelo usuário. Ele garante que o número seja exibido de forma padronizada, tanto para celular quanto para fixo.

- **Função:**

```
export const formatarTelefone = (numero) => {
```

A função recebe uma string com o número de telefone, ela é utilizada nos campos de entrada para aplicar a formatação em tempo real, conforme o usuário digita.

- **Limpeza e limitação de dígitos:**

```
let numeroFormatado = numero.replace(/\D/g, "").slice(0, 11);
```

Remove todos os caracteres não numéricos pelo uso de **replace**, substitui todos os caracteres não numéricos em **numero** por uma string vazia.

O **\d** indica qualquer caractere que não seja um número, usando o modificador global, **g**, para indicar que deve ser feito em todos os casos, não apenas no primeiro.

A combinação de **replace** com a expressão regular garante que a variável **numero** será transformada em uma string contendo somente os dígitos numéricos.

Após a remoção dos caracteres não numéricos, o método **.slice(0, 11)** extrai apenas os primeiros 11 caracteres da string, limitando o resultado em 11 dígitos.

- **Formatação para celular (11 dígitos):**

```
if (numeroFormatado.length === 11) {  
    return `(${numeroFormatado.slice(0, 2)}) ${numeroFormatado.slice(2,  
7)}-${numeroFormatado.slice(7)}`;  
}
```

Trecho executado caso a variável **numeroFormatado** tenha 11 dígitos.

numeroFormatado.slice(0, 2) seleciona os dois primeiros caracteres, o DDD, e os coloca entre parênteses.

numeroFormatado.slice(2, 7) seleciona os próximos 5 caracteres e adiciona o símbolo -.

numeroFormatado.slice(7) pega a parte restante do número.

Exemplo de entrada do usuário: 11944332211.

Saída formatada: (11) 94433-2211.

- **Formatação para telefone fixo (10 dígitos):**

```
else if (numeroFormatado.length === 10) {  
    return `(${numeroFormatado.slice(0, 2)}) ${numeroFormatado.slice(2,  
6)}-${numeroFormatado.slice(6)}`;  
}
```

Trecho executado caso a variável **numeroFormatado** tenha 10 dígitos.

Faz uso do **slice** e formatação parecida com a formatação para celular, com diferença na quantidade de números ao lado esquerdo do símbolo -.

- **Formatação parcial (ao digitar o DDD):**

```
else if (numeroFormatado.length >= 3) {  
    return `(${numeroFormatado.slice(0, 2)})  
${numeroFormatado.slice(2)}`;  
}
```

Faz a formatação inicial do DDD quando o comprimento de **numeroFormatado** for **>= 3**.

Exemplo de entrada: 1194.

Saída: (11) 94.

- **Retorno final para números incompletos:**

```
return numeroFormatado;
```

Caso ainda não tenha dígitos suficientes para aplicar qualquer formatação mais avançada, retorna apenas os números digitados.

- **utils/selecaoImagem.js**

Este arquivo contém uma função utilitária chamada **selecionarImagem**, responsável por abrir a galeria de imagens do dispositivo e permitir que o usuário selecione uma imagem, que será usada como foto do fornecedor. Essa função é usada nas telas de cadastro e edição de fornecedores.

- **Biblioteca:**

```
import * as ImagePicker from 'expo-image-picker';
```

A função faz uso da biblioteca **expo-image-picker**, específica para lidar com imagens no React Native.

- **Função selecionarImagem:**

```
export async function selecionarImagem(setImagemUri) {
```

A função é assíncrona porque envolve chamadas que dependem de permissões e interações do usuário.

Recebe o estado da imagem com **setImagemUri** como parâmetro, usada para atualizar o estado da URL da imagem no componente que chamou a função.

- **Solicitação de permissão:**

```
const {status} = await ImagePicker.requestMediaLibraryPermissionsAsync();  
if (status !== 'granted') {  
  alert('Necessário permissão para acessar a galeria.');
```

Solicita permissão para acessar a galeria de mídia do dispositivo. Se o usuário negar o acesso, a função exibe um alerta e interrompe a execução.

- **Abertura da galeria:**

```
const result = await ImagePicker.launchImageLibraryAsync({  
  mediaTypes: ImagePicker.Images,  
  allowsEditing: true,  
  aspect: [4, 4],  
  quality: 1,  
})
```

Abre a galeria de imagens, define o aspecto da imagem para um formato quadrado com **[4, 4]**, permitindo que o usuário recorte a imagem antes de confirmar e garante a melhor qualidade de imagem possível com o **quality: 1**.

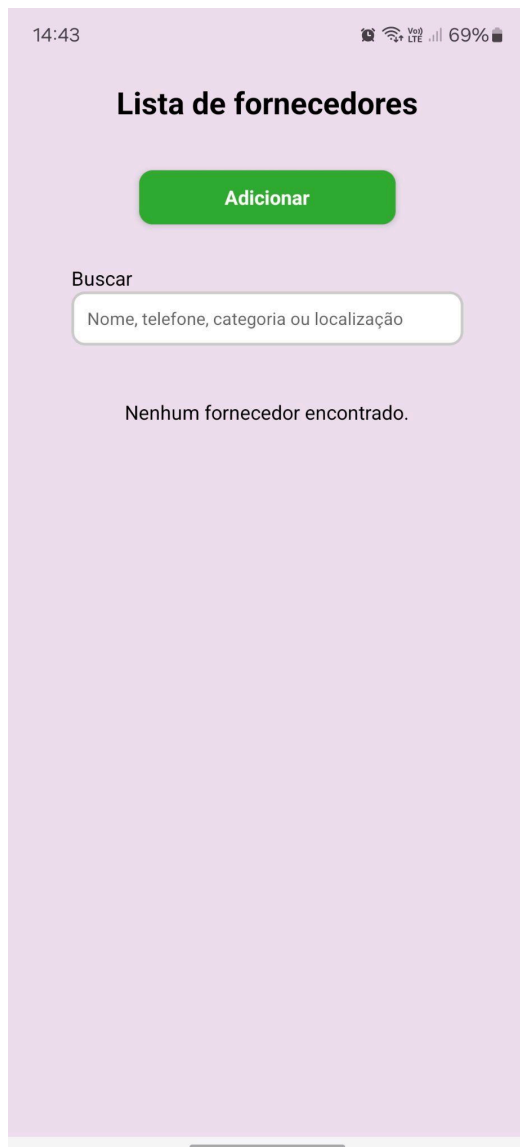
- **Atualização da URI da imagem:**

```
if (!result.canceled) {  
  setImagemUri(result.assets[0].uri || "")  
}
```

Primeiro verifica se o usuário confirmou a seleção, não cancelou, para assim, atualizar a URI da imagem no estado local do componente que chamou a função, permitindo que a imagem seja exibida na interface.

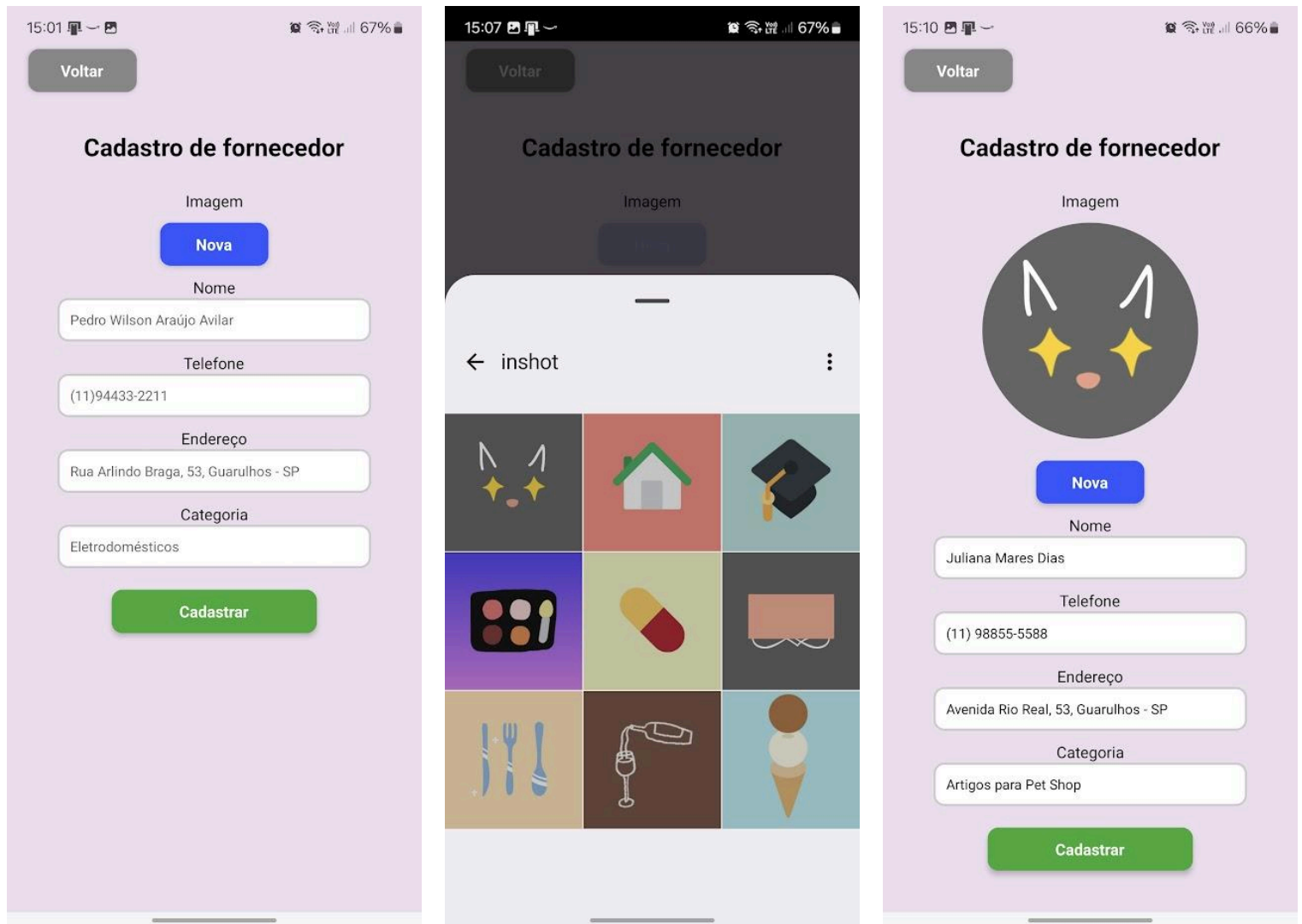
- **Capturas de tela e fluxos de interações em dispositivos móveis**

- **Tela inicial sem fornecedores cadastrados:**



Conteúdo da página **index.js**. Abaixo do título da página temos o botão “Adicionar”, em cor verde, que navega para a página **cadastro.js**, a barra de “Buscar” e a mensagem informando o usuário que não existem fornecedores cadastrados.

- **Tela de cadastro de um novo fornecedor:**



Essa tela é renderizada pelo arquivo **cadastro.js**.

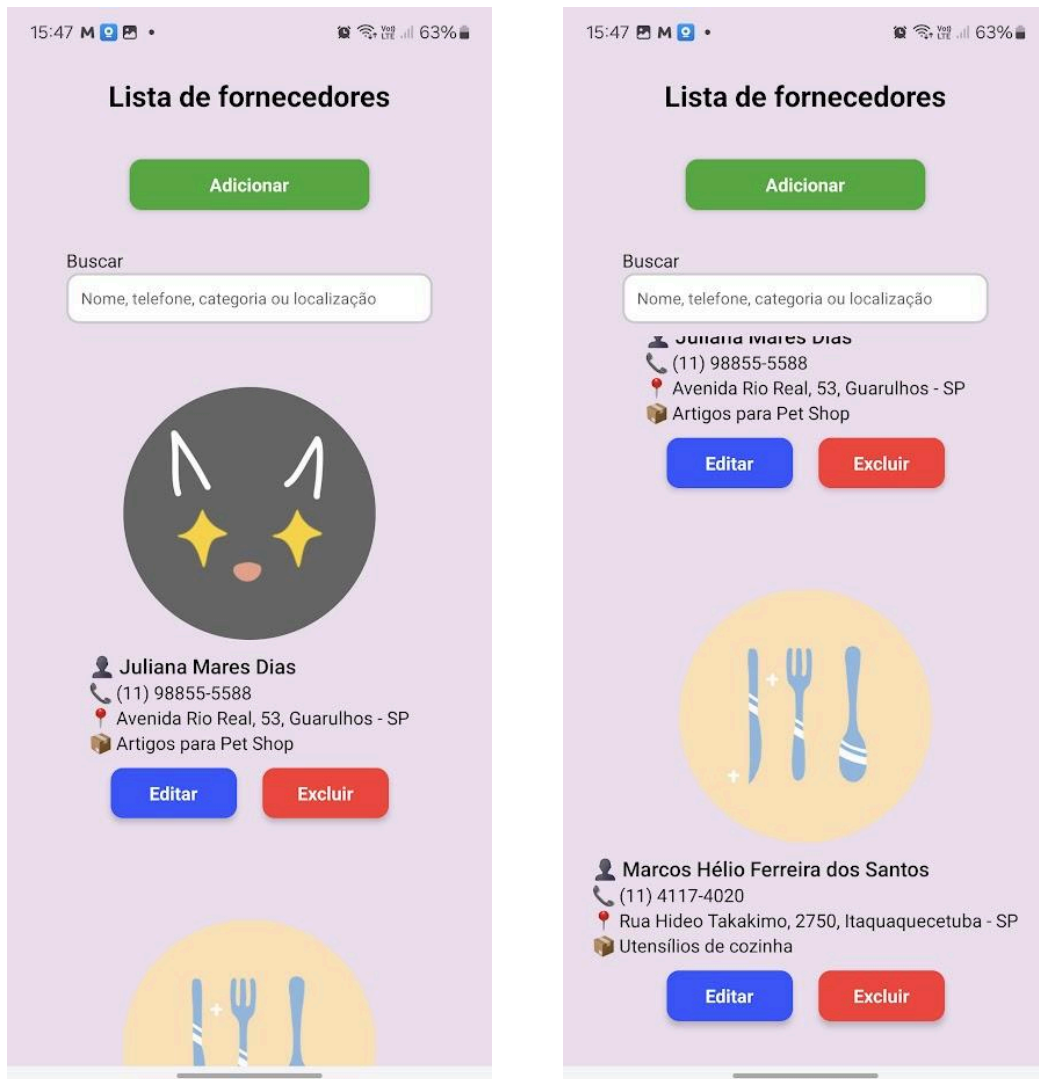
No início da página, ao lado esquerdo, está disponível o botão “Voltar”, para permitir liberdade ao usuário caso desista de cadastrar um novo fornecedor, navegando para a tela inicial, **index.js**.

Abaixo do título da página começa o formulário de preenchimento feito pelo usuário para a foto e informações do fornecedor, foto à esquerda.

No campo para imagem, tem um botão com o texto “Nova”, que quando clicado, abre a galeria do dispositivo, com a permissão do usuário, para escolha da imagem, foto ao centro, permitindo edição, e então atualiza a tela com a imagem selecionada, ainda mantendo o botão disponível caso o usuário decida mudar a imagem quantas vezes precisar antes de concluir o cadastro.

Após preencher todos os campos, foto à direita, o usuário clicando no botão verde com o texto “Cadastrar”, o fornecedor é salvo na aplicação e o usuário é redirecionado para a tela inicial.

○ **Tela inicial com fornecedores cadastrados:**



Quando temos fornecedores cadastrados, a tela inicial faz a listagem.

O título da página, o botão para adicionar um novo fornecedor e o campo de busca permanecem fixos no topo da página, permitindo a facilidade de busca e cadastro a qualquer momento pelo usuário, permitindo o rolar de tela para visualizar a lista de fornecedores, apenas na área da própria lista.

Dentro da listagem cada fornecedor é exibido começando pela imagem, seguida pelas informações de nome, telefone, endereço e categoria e, por fim, os botões de ações para editar ou excluir o fornecedor da listagem.

○ **Editar um fornecedor cadastrado:**

Voltar

Editar fornecedor

Imagem

Nova

Nome

Juliana Mares Dias

Telefone

(11) 98855-5588

Endereço

Avenida Rio Real, 53, Guarulhos - SP

Categoria

Artigos para Pet Shop

Salvar

Voltar

Editar fornecedor

Imagem

Nova

Nome

Juliana Mares Dias

Telefone

(11) 95555-5555

Endereço

Rua Macedo Lima, 555, Guarulhos - SP

Categoria

Artigos para Pet Shop

Salvar

Adicionar

Buscar

Nome, telefone, categoria ou localização

Juliana Mares Dias

(11) 95555-5555

Rua Macedo Lima, 555, Guarulhos - SP

Artigos para Pet Shop

Editar

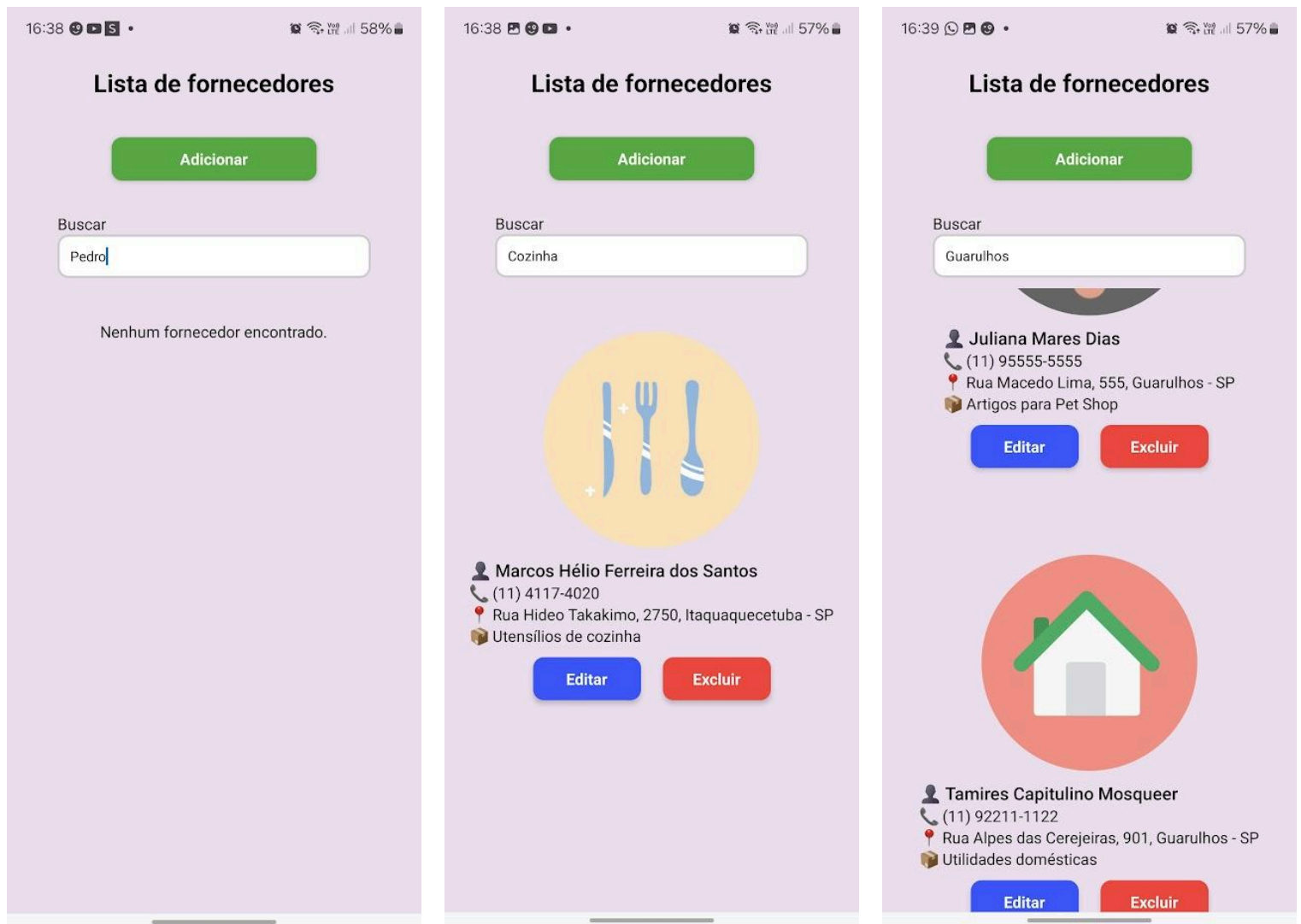
Excluir

Na tela inicial, quando clicado no botão azul com o texto “Editar”, em algum fornecedor, o usuário é redirecionado para a tela de edição, **editar.js**.

Nessa tela estão disponíveis as atuais informações do fornecedor em um formulário, foto à esquerda, com o botão no canto superior esquerdo de voltar, caso o usuário desista da edição.

É possível alterar apenas uma informação, foto ou todos os itens, como na foto ao centro, onde foi editada a foto, o telefone e o endereço. Após as edições necessárias, ao final do formulário, o usuário clica no botão verde, com o texto “Salvar”, para salvar as alterações, sendo redirecionado para a tela inicial, que agora apresenta os novos dados do fornecedor que foi alterado.

○ **Buscar fornecedores na tela inicial:**



Utilizando o campo de busca, na tela inicial, é possível pesquisar, dinamicamente, fornecedores dentro da listagem que correspondam ao texto digitado, por qualquer informação dos fornecedores.

Quando a pesquisa não corresponde a nenhum fornecedor cadastrado retorna a mensagem informando que nenhum foi encontrado, foto à esquerda com pesquisa por “Pedro”, como, no momento, nenhum fornecedor cadastrado tem o nome Pedro. Conforme é apagado o texto retorna a listagem completa dos cadastrados.

Na imagem ao centro foi realizado a busca por “Cozinha”, encontrando apenas o cadastro do fornecedor “Marcos Hélio Ferreira dos Santos”, que no campo de categoria possui a informação “Utensílios de cozinha”.

Pesquisando por “Guarulhos”, na foto à direita, temos a correspondência e exibição de dois fornecedores cadastrados, já que os dois no campo de localização possuem a palavra digitada.

○ **Excluir fornecedor na tela inicial:**



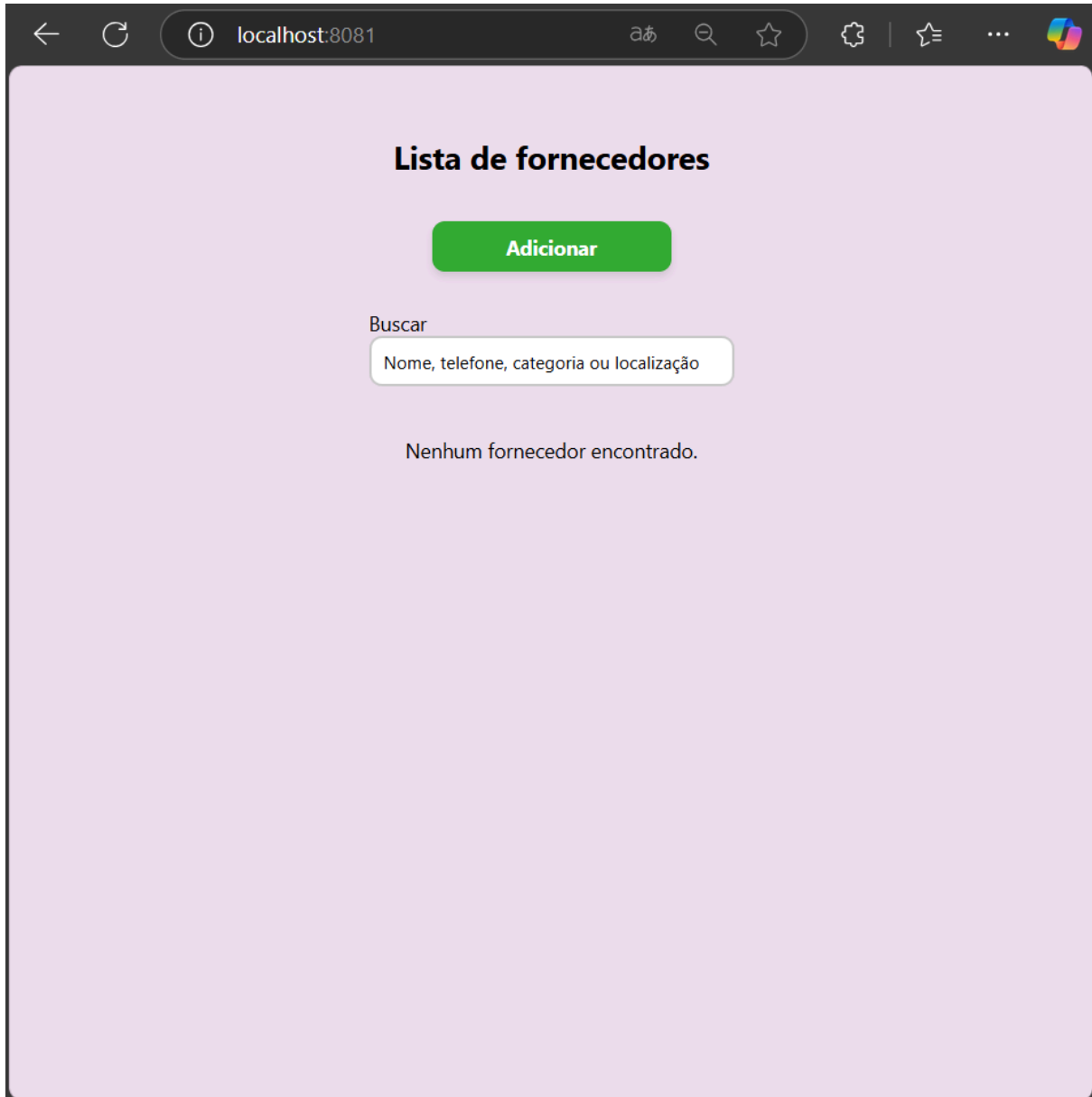
Na listagem de fornecedores, cada fornecedor possui um botão vermelho, escrito “Excluir”, quando pressionado, exibe um alerta para o usuário perguntando “Tem certeza que deseja excluir?” com as opções para cancelar ou excluir.

O usuário confirmando a exclusão, o fornecedor é apagado do aplicativo e a tela inicial é redesenhada sem a presença do fornecedor excluído.

- **Capturas de tela em navegadores web**

As interações e fluxos do programa foram testadas tanto em dispositivos móveis como navegadores web em computadores, funcionando de forma semelhante em ambas plataformas.

- **Tela inicial sem fornecedores cadastrados:**




- **Tela de cadastro de um novo fornecedor:**

← ↻ ⓘ localhost:8081/cadastro 🔍 ☆ ⚙️ | ☆ ≡ ... 🌈

Voltar

Cadastro de fornecedor

Imagem



Nova

Nome

Juliana Mares Dias

Telefone

(11) 9855-5588

Endereço

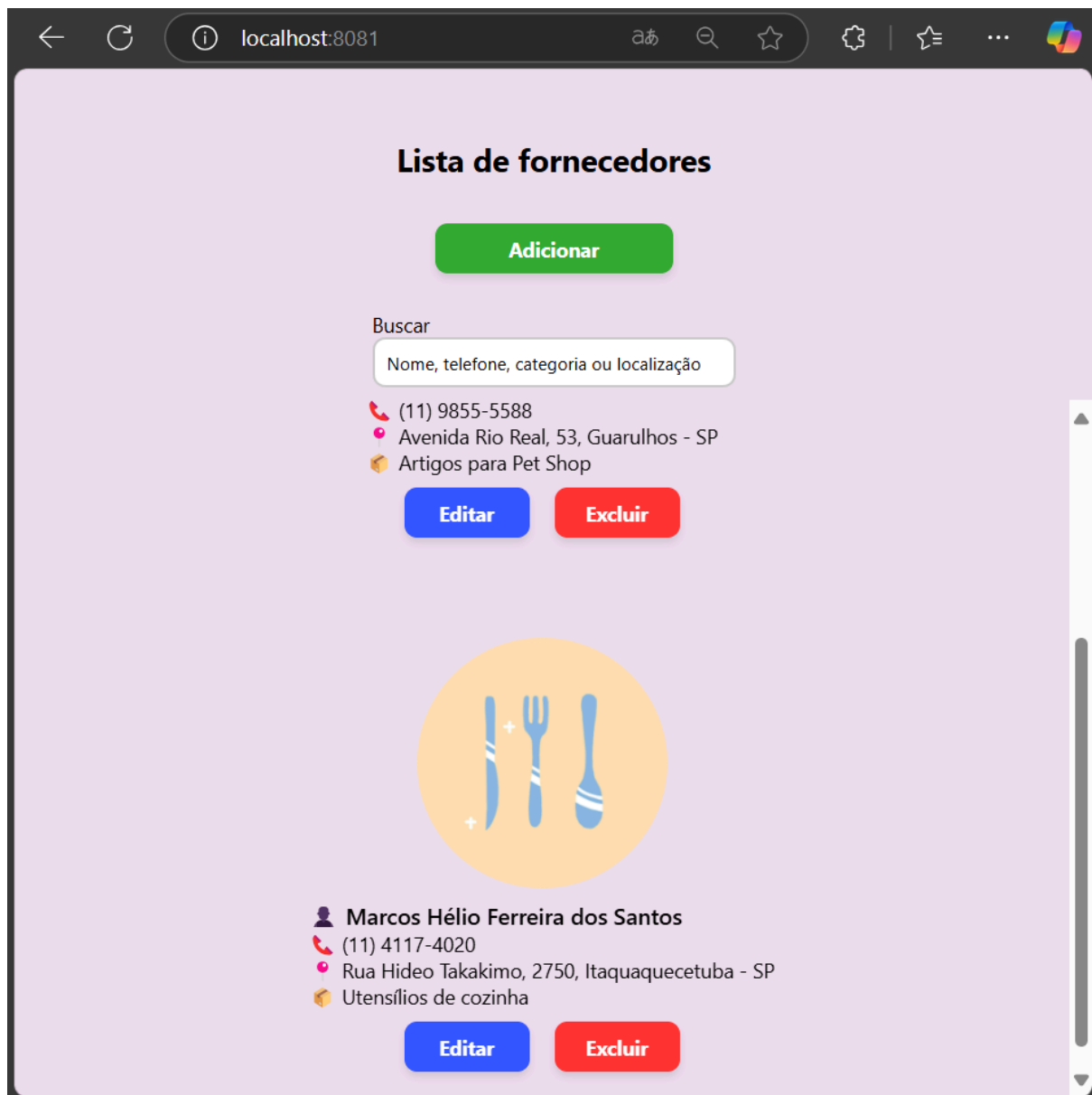
Avenida Rio Real, 53, Guarulhos - SP

Categoria

Artigos para Pet Shop

Cadastrar

- Tela inicial com fornecedores cadastrados:




○ **Editar um fornecedor cadastrado:**

localhost:8081/editar?id=1

Voltar

Editar fornecedor

Imagem



Nova

Nome

Juliana Mares Dias

Telefone

(11) 95555-5555

Endereço

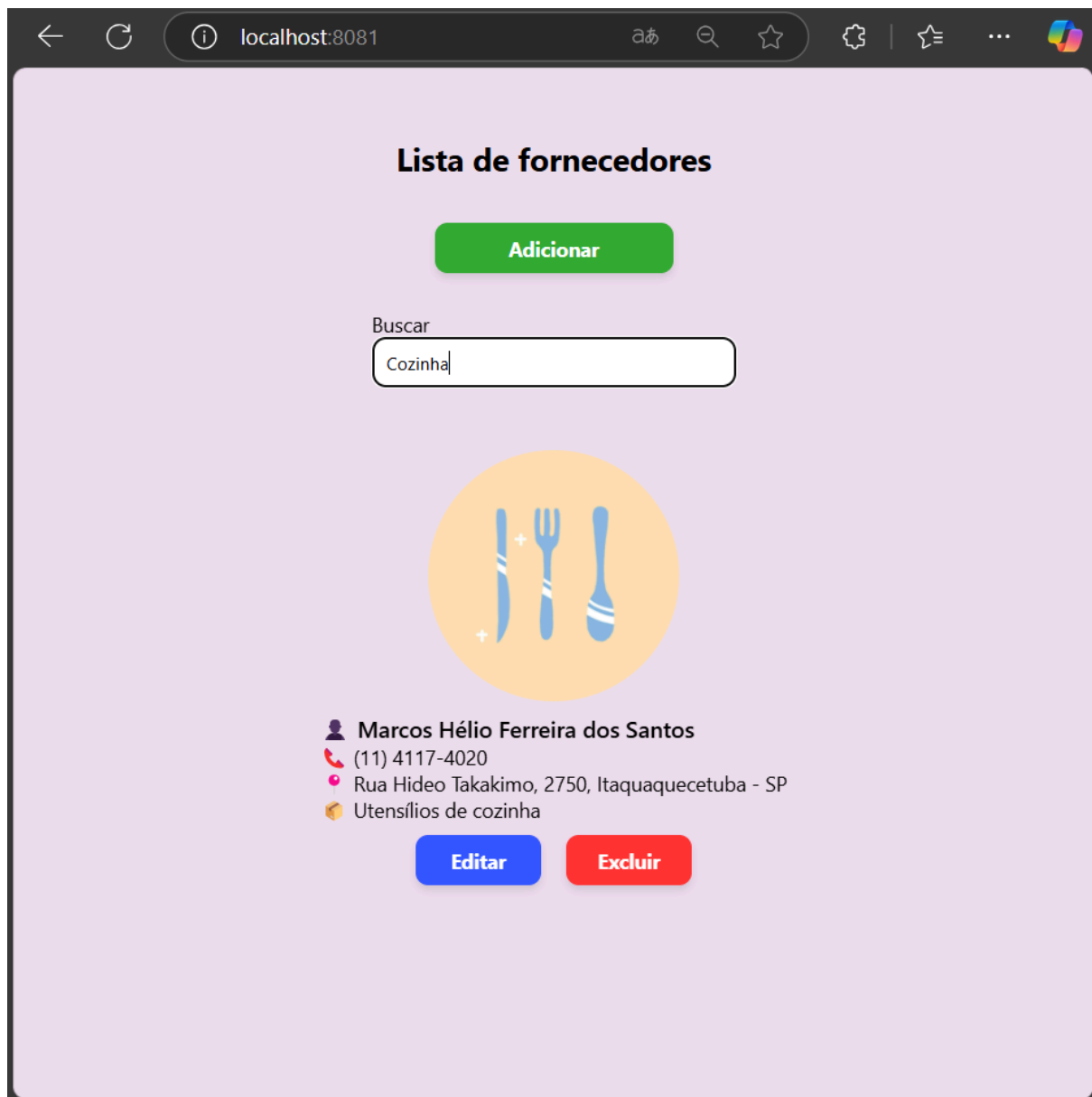
Rua Macedo Lima, 555, Guarulhos - SP

Categoria

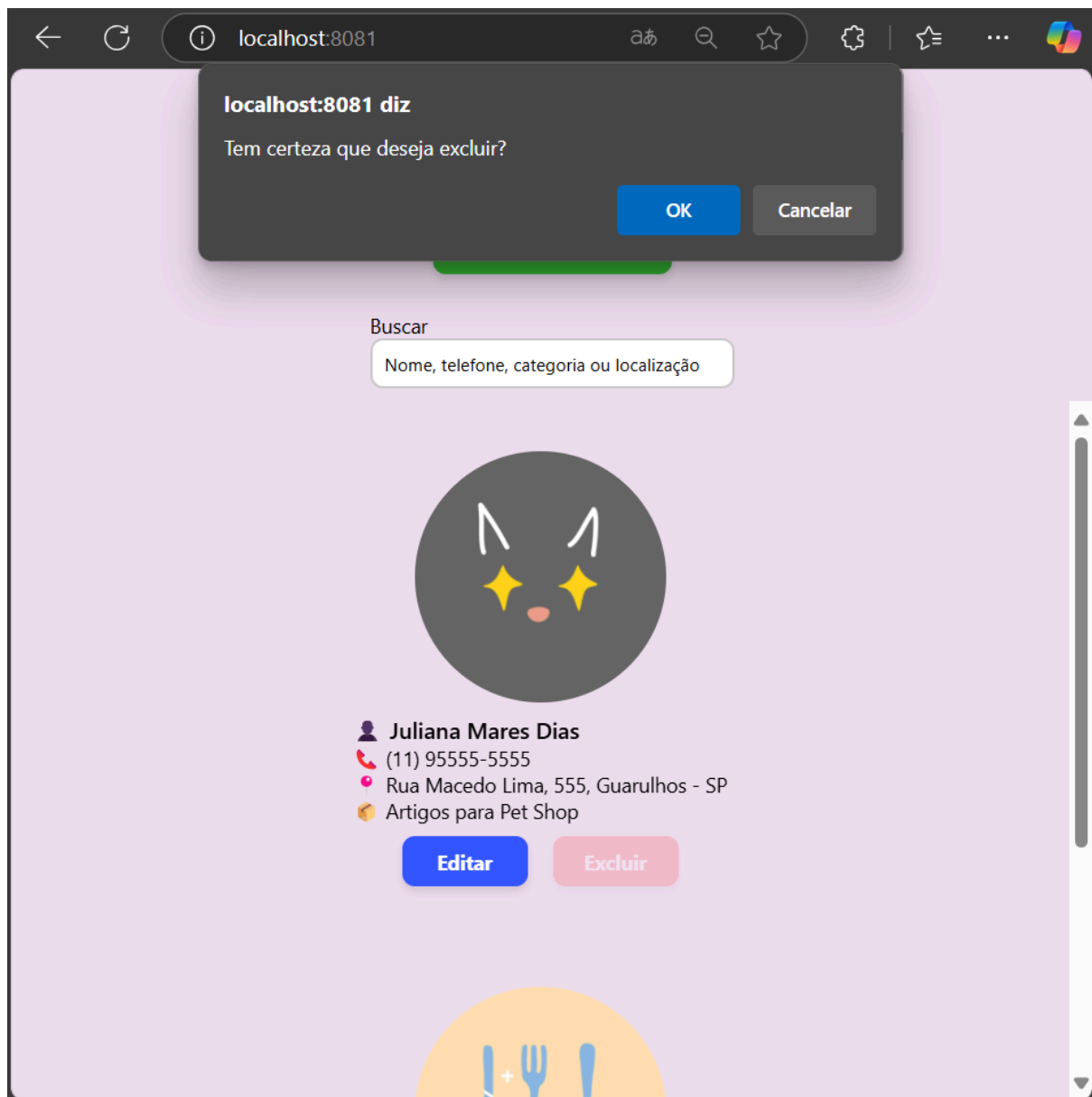
Artigos para Pet Shop

Salvar

- **Buscar fornecedores na tela inicial:**



- **Excluir fornecedor na tela inicial:**



- **Conclusão**

A missão prática resultou no desenvolvimento de um aplicativo funcional voltado para o cadastro e gerenciamento de fornecedores. Foi pensado, durante todo o processo, criar uma aplicação eficiente, com navegação fluida e estrutura de código clara e organizada, tendo uma interface intuitiva e amigável, facilitando a experiência do usuário para as ações necessárias.