PUC Minas – Coração Eucarístico

Turma 8221100/8221101

Projeto Integrado I: Desenvolvimento Móvel

Professor responsável

Ilo Amy Saldanha Rivero

Documentação Rent a House

Gabriel Batista de Almeida

Gabriel da Silva Cassino

Igor Cesar Avelar Leao

Índice - "Rent a House"

Visão Geral da Aplicação "Rent a House"

O projeto está disponível em: *https://github.com/kasshinokun/Projeto-Integrado-Desenvolvimento-Movel*
a versão final está em: *https://github.com/kasshinokun/Projeto-Integrado-Desenvolvimento-Movel/tree/main/Rent_a_House_App/Releases/TP_S4/entrega*

O aplicativo visa conectar proprietários e inquilinos, oferecendo recursos de listagem de imóveis, autenticação de usuários, chat em tempo real, e um fluxo de checkout para aluguel.

A arquitetura do aplicativo utiliza Flutter para o frontend, Firebase (Authentication, Firestore, Realtime Database) para o backend e persistência de dados, e SQLite para armazenamento local de dados críticos de usuário e chaves de acesso.

Estrutura Geral

O aplicativo é dividido em várias telas e serviços, cada um com uma responsabilidade específica:

- **Serviços (Services):** Camada de lógica de negócios e interação com o backend (Firebase, SQLite).
- **Provedores (Providers):** Gerenciamento de estado e dados para as telas.
- **Telas (Screens/Pages):** A interface do usuário e interação com o usuário.

Estrutura de Pastas do Projeto

A organização de pastas em um projeto Flutter é crucial para a escalabilidade e manutenibilidade. Abaixo está um modelo adotado:

```
.
├── lib/
│   ├── combined_connection_handler_screen.dart
│   ├── firebase_options.dart
│   ├── main.dart
│   └── myapp.dart
│   ├── providers/
│   │   ├── connectivity_provider.dart
│   │   ├── housesprovider.dart
│   │   └── internet_connection_provider.dart
│   ├── screens/
│   │   ├── chat_page.dart
│   │   ├── checkout_page.dart
│   │   ├── connection_pannel_screen.dart
│   │   ├── conversations_screen.dart
│   │   ├── details_screen.dart
│   │   ├── home_page.dart
│   │   ├── house_screen.dart
│   │   ├── login.dart
```

```
│   │   ├── myhouses_screen.dart
│   │   ├── no_internet_screen.dart
│   │   ├── profile_screen.dart
│   │   └── register_house_page.dart
│   ├── services/
│   │   ├── authservices.dart
│   │   └── fakeruser.dart
│   └── utils/
│   │   ├── database_helper.dart
│   │   ├── resultcep.dart
│   │   └── vviacep.dart components/)
├── assets/
│   ├── app/
│   │   ├── user.png
│   │   └── google_logo.jpg
│   ├── fonts/
│   │   └── custom_font.ttf
│   └── icons/
│       └── app_icon.png
├── android/
├── pubspec.yaml
├── pubspec.lock
├── README.md
└── (outros arquivos de configuração)
```

Classes Principais

Será descrito a seguir algumas partes principais para não ser extenso ao leitor.

1. AuthService (lib/services/authservices.dart)

Funcionalidade: Este serviço é o coração da autenticação do usuário. Ele gerencia o login, registro e logout, além de lidar com a persistência do estado de autenticação tanto online (Firebase Authentication) quanto offline (SharedPreferences e SQLite).

Funcionamento Geral:

- **Autenticação Firebase:** Utiliza FirebaseAuth para operações de login com e-mail/senha, registro e login anônimo.
- **Login Offline:** Implementa um mecanismo de login offline. Quando não há conexão com a internet ou o login online falha por problemas de rede, ele tenta autenticar o usuário usando credenciais salvas localmente no SQLite e SharedPreferences.
- **Persistência de Dados:**
    - **SharedPreferences:** Armazena o UID, e-mail e nome de exibição do usuário para manter o estado de login básico entre as sessões, mesmo offline.

    - **SQLite:** Salva o UID, e-mail, nome, senha com hash e o caminho da foto de perfil localmente para um login offline mais robusto e seguro (a senha é armazenada como um hash SHA256).
- **Gerenciamento de Foto de Perfil:** Salva a imagem de perfil do usuário (se fornecida como Base64 do Firestore) em um arquivo temporário local e armazena o caminho para exibição offline.
- FakeUser**:** Quando o usuário está offline e autenticado localmente, uma instância de FakeUser é usada para simular um objeto User do Firebase, garantindo que o restante do aplicativo possa acessar as informações básicas do usuário de forma consistente.
- **Notificação de Estado:** Estende ChangeNotifier para notificar os widgets que dependem do estado de autenticação (ex: se o usuário está logado ou não).

2. DatabaseHelper (lib/utils/database_helper.dart)

Funcionalidade: Este é um singleton que gerencia a criação, abertura e operações CRUD (Create, Read, Update, Delete) com o banco de dados SQLite local do aplicativo. Ele é essencial para a funcionalidade de login offline e para armazenar dados que precisam persistir no dispositivo, como as chaves de acesso de aluguel.

Funcionamento Geral:

- **Singleton:** Garante que haja apenas uma instância do DatabaseHelper em todo o aplicativo.
- **Inicialização do Banco de Dados:** O método _initDatabase abre ou cria o arquivo user_data.db no diretório de documentos do aplicativo.
- **Criação de Tabelas:** O método _onCreate define o esquema das tabelas. Atualmente, ele cria:
- users_offline: Armazena informações básicas do usuário para login offline (UID, e-mail, senha com hash, nome, foto de perfil Base64).
- rental_access_keys: Armazena as chaves de acesso geradas para os aluguéis (ID do imóvel, UID do usuário, chave de acesso, timestamp de criação).
- **Operações CRUD:** Fornece métodos para:
- insertUser: Inserir ou atualizar dados de usuário offline.
- getUserByEmail: Buscar um usuário offline pelo e-mail.
- deleteUser: Excluir um usuário offline.
- clearAllUsers: Limpar todos os usuários offline.
- insertAccessKey: Salvar as chaves de acesso de aluguel.
- getAccessKeysForUser: (Opcional) Buscar chaves de acesso para um usuário específico.

3. HousesProvider (lib/providers/housesprovider.dart)

Funcionalidade: Este ChangeNotifier é responsável por gerenciar e fornecer os dados dos imóveis para o restante do aplicativo. Ele interage diretamente com o Firestore para buscar, categorizar e ouvir atualizações em tempo real dos imóveis e suas fotos.

Funcionamento Geral:

- **Modelos de Dados:** Define os modelos House e HousePhoto para mapear os documentos do Firestore para objetos Dart.
- **Escuta em Tempo Real:** O método _listenToHouses usa snapshots() do Firestore para ouvir em tempo real todas as mudanças na coleção houses. Isso garante que a lista de imóveis seja sempre atualizada na UI.
- **Categorização:** Os imóveis são categorizados por tipo (_categorizedHouses) para facilitar a exibição em abas ou seções.
- **Busca de Detalhes:** Fornece métodos fetchHouseById e fetchAllHousePhotos para buscar detalhes de um imóvel específico e todas as suas fotos.
- **Estado de Carregamento e Erro:** Mantém variáveis _isLoading e _error para informar a UI sobre o status da operação de busca de dados.
- **Tratamento de Erro de Índice:** Inclui lógica para detectar erros de "índice faltando" do Firestore, extraindo a URL para criação do índice e fornecendo uma mensagem de erro útil ao desenvolvedor.

4. ProfileScreen (lib/screens/profile_screen.dart)

ProfileScreen é a interface onde o usuário pode visualizar e interagir com suas informações de perfil.

Propósito

A ProfileScreen tem como objetivo principal fornecer ao usuário um hub centralizado para:

- Visualizar suas informações pessoais (nome, e-mail, telefone, foto de perfil).
- Acessar estatísticas rápidas relacionadas ao seu uso do aplicativo (ex: imóveis alugados, favoritos).
- Realizar ações relacionadas à sua conta e aos imóveis que ele possui ou aluga.
- Gerenciar sua sessão (logout).

Funcionamento Geral

1. **Carregamento de Dados do Perfil:**
    - Ao ser inicializada (initState), a tela verifica o status de autenticação do usuário através do AuthService.
    - Para usuários autenticados (não anônimos), ela tenta buscar dados adicionais do perfil (como nome, e-mail, telefone e foto de perfil em Base64) da coleção users no Firestore.

- Se uma foto de perfil em Base64 for encontrada no Firestore, o AuthService tenta salvá-la em um arquivo temporário local e usa o caminho desse arquivo para exibição, garantindo que a foto possa ser vista mesmo offline.
- Para usuários anônimos ou se os dados não forem encontrados no Firestore, são exibidas informações genéricas (ex: "Convidado", "N/A").
- Um indicador de carregamento (CircularProgressIndicator) é exibido enquanto os dados do perfil estão sendo buscados.

2. **Exibição de Informações do Usuário:**
    - A tela apresenta uma seção de cabeçalho com a foto de perfil do usuário (ou um ícone de pessoa como fallback). Há um pequeno ícone de câmera sobreposto na foto de perfil, indicando uma futura funcionalidade de edição de foto.
    - Abaixo da foto, são exibidos o nome, e-mail e telefone do usuário.
3. **Seção de Estatísticas:**
    - Uma linha de colunas de estatísticas (_buildStatColumn) mostra informações como "Imóveis Alugados", "Favoritos" e "Avaliações". Atualmente, esses valores são estáticos ('5', '12', '4.8'), mas a estrutura está pronta para integrar dados dinâmicos de backend.
4. **Ações do Perfil (**_buildActionCard**):**
    - Uma lista de Cards clicáveis oferece diversas ações relacionadas à conta do usuário:
        - **"Editar Perfil"**: Destinado a futuras funcionalidades de atualização de nome, telefone, etc.
        - **"Minhas Chaves de Acesso"**: Para visualizar as chaves de acesso geradas para os aluguéis (funcionalidade a ser implementada).
        - **"Histórico de Aluguéis"**: Para ver aluguéis passados (funcionalidade a ser implementada).
        - **"Configurações"**: Para ajustes gerais do aplicativo (funcionalidade a ser implementada).
    - Cada cartão possui um ícone, título e uma seta indicando que é clicável.
5. **Botão de Logout:**
    - Um botão grande e proeminente "Sair" permite que o usuário encerre sua sessão. Ao ser clicado, ele chama o método signOutUser() do AuthService, que lida com o logout do Firebase e a limpeza dos dados de login offline.

6. **Acesso Condicional:**
    - As seções detalhadas do perfil (estatísticas e ações da conta) são exibidas apenas para usuários autenticados (não anônimos). Se o usuário for anônimo ou não estiver logado, uma mensagem amigável é mostrada, convidando-o a fazer login.
7. **Tratamento de Erros:**
    - Em caso de falha no carregamento dos dados do perfil, uma mensagem de erro é exibida na tela.

5. HomePage (lib/screens/home_page.dart)

Funcionalidade: A HomePage atua como a tela principal e o hub de navegação do aplicativo. Ela utiliza uma barra de navegação inferior curvada (CurvedNavigationBar) e um PageView para permitir que o usuário alterne entre diferentes seções do aplicativo.

Funcionamento Geral:

- **Navegação Principal:** A CurvedNavigationBar na parte inferior oferece acesso rápido a seções como "Home" (listagem de imóveis), "Alugar", "Perfil", "Ajustes", "Chat", "Pix" e "Status".
- PageView**:** As diferentes telas correspondentes a cada item da barra de navegação são carregadas em um PageView, permitindo uma transição suave entre elas.
- **Integração com** HousesProvider**:** A tela "Home" (que exibe os imóveis) é um Consumer do HousesProvider, reagindo a mudanças no estado de carregamento e erro dos imóveis.
- **Logout:** Inclui um botão de logout na AppBar que utiliza o AuthService para encerrar a sessão do usuário.
- **Exibição Condicional:** A tela "Home" exibe um indicador de carregamento, mensagens de erro ou uma mensagem de "nenhum imóvel disponível" com um botão de atualização, dependendo do estado do HousesProvider.

6. HouseScreen (lib/screens/house_screen.dart)

Funcionalidade: Esta tela é responsável por exibir a lista de imóveis disponíveis, categorizados por tipo. Ela se integra com o HousesProvider para obter os dados em tempo real e apresenta-os em abas (TabBarView).

Funcionamento Geral:

- Consumer<HousesProvider>**:** A tela observa o HousesProvider para reagir a mudanças nos dados dos imóveis (carregamento, erros, lista de imóveis).

- TabBar **e** TabBarView**:** Utiliza um TabController e TabBar para exibir as categorias de imóveis como abas. Cada aba contém um GridView que mostra os imóveis daquela categoria.
- **Cards de Imóveis:** Cada imóvel é exibido em um Card com informações como nome, endereço, preço e uma imagem (placeholder, pois as fotos são carregadas na DetailsScreen).
- **Navegação para Detalhes:** Ao clicar em um imóvel, o usuário é navegado para a DetailsScreen correspondente, passando o ID do imóvel.
- **Feedback de Estado:** Exibe CircularProgressIndicator durante o carregamento e mensagens de erro ou de lista vazia quando apropriado.

7. ChatPage (lib/screens/chat_page.dart) e ConversationsScreen (lib/screens/conversations_screen.dart)

Propósito: A ChatPage permite que os usuários se comuniquem em tempo real, facilitando a negociação e o esclarecimento de dúvidas sobre os imóveis.

Funcionamento Geral:

- **Integração com Firebase Realtime Database:** O chat utiliza o Firebase Realtime Database para armazenar e sincronizar as mensagens. Isso garante que as mensagens sejam entregues e exibidas instantaneamente para ambos os participantes da conversa.
- **Identificação da Conversa (**chatId**):** Um chatId único é gerado a partir dos UIDs (IDs de Usuário) dos dois participantes, garantindo que a mesma conversa seja acessada por ambos, independentemente da ordem em que foi iniciada.
- **Metadados do Chat:** Informações adicionais sobre o chat, como os UIDs dos participantes e, opcionalmente, o ID e nome do imóvel relacionado à conversa, são salvas como metadados. Isso ajuda a contextualizar a conversa para os usuários.
- **Exibição de Mensagens:** As mensagens são carregadas em tempo real usando um StreamBuilder que ouve as atualizações no Realtime Database. Elas são exibidas em um ListView.builder, com cada mensagem formatada como uma "bolha" de chat.
- ChatMessageTile**:** Um widget dedicado (ChatMessageTile) é responsável por renderizar cada bolha de mensagem, diferenciando visualmente as mensagens enviadas pelo usuário atual (bolhas coloridas à direita) das mensagens recebidas (bolhas cinzas à esquerda).
- **Rolagem Automática:** O chat rola automaticamente para a mensagem mais recente sempre que novas mensagens são carregadas ou enviadas, garantindo que o usuário veja o conteúdo mais novo.

- **Envio de Mensagens:** Um campo de texto na parte inferior da tela permite que o usuário digite e envie mensagens. Ao enviar, a mensagem é salva no Realtime Database e a lista de conversas de ambos os usuários é atualizada com a última mensagem.
- **Tratamento de Usuário Não Autenticado:** Se um usuário tentar acessar o chat sem estar logado, uma mensagem informativa é exibida, incentivando-o a fazer login.
- **Histórico de Mensagens:** Em ConversationsScreen fica armazenado as últimas conversas entre o usuário logado e quem o contatou a cerca do imóvel ou os locadores que entrou em contato, apresentando o nome do usuário e como assunto o nome do imóvel para conceder uma experiencia agradável ao cliente da aplicação.

8. CheckoutPage (lib/screens/checkout_page.dart)

CheckoutPage é a interface onde o usuário finaliza o processo de aluguel de um imóvel.

Propósito

A CheckoutPage tem como objetivo principal guiar o usuário através das etapas finais para alugar um imóvel, permitindo a seleção de datas de aluguel, a visualização do valor total e a escolha de um método de pagamento (Pix ou Cartão de Crédito, atualmente simulados).

Funcionamento Geral

1. **Recebimento do Imóvel:**
    - A CheckoutPage é inicializada recebendo um objeto House completo como parâmetro. Isso garante que todas as informações do imóvel selecionado estejam disponíveis para o processo de checkout.
2. **Seleção de Período de Aluguel:**
    - O usuário pode selecionar a "Data de Início" e a "Data de Término" do aluguel através de seletores de data (showDatePicker).
    - Há validação para garantir que a data de término seja posterior à data de início.
3. **Cálculo do Valor Total:**
    - O valor total do aluguel é calculado dinamicamente com base no preço diário do imóvel (assumindo que house.price é o preço por dia) e na duração do período de aluguel selecionado.
    - O campo de "Valor Total" é atualizado automaticamente após a seleção das datas.
4. **Seleção do Método de Pagamento:**
    - Um SegmentedButton permite ao usuário alternar entre "Pix" e "Cartão de Crédito" como métodos de pagamento.

- A interface se adapta para exibir os campos relevantes para o método de pagamento escolhido.

5. **Pagamento via Pix (Simulado):**
    - Quando "Pix" é selecionado, um botão "Gerar código Pix" é exibido.
    - Ao clicar, um código Pix simulado é gerado e exibido em um SelectableText, permitindo que o usuário o copie.
    - Um botão "Confirmar pagamento" é disponibilizado. Ao ser clicado, ele simula a confirmação do pagamento e muda seu estado para "Pagamento confirmado!", ficando desabilitado.
6. **Pagamento via Cartão de Crédito (Simulado):**
    - Quando "Cartão de Crédito" é selecionado, campos de entrada para "Número do Cartão", "Nome no Cartão", "Data de Validade" e "CVV" são exibidos.
    - Há um botão "Pagar com Cartão". Ao ser clicado, ele também chama a mesma função de _confirmarPagamento simulada, indicando que o pagamento foi "processado".
    - **Importante:** Os campos de cartão de crédito são apenas para entrada de dados e não realizam nenhuma validação real ou processamento de pagamento.
7. **Confirmação e Feedback:**
    - Após a "confirmação" do pagamento (seja Pix ou Cartão), um AlertDialog é exibido, informando ao usuário que o pagamento foi bem-sucedido e agradecendo.
    - O estado _pagamentoConfirmado controla a exibição do botão de confirmação e a mensagem de sucesso.

Limitações e Considerações Atuais

- **Pagamentos Simulados:** A funcionalidade de pagamento é inteiramente simulada. Não há integração real com gateways de pagamento (como Stripe, PagSeguro, Cielo, etc.) ou com APIs Pix.
- **Validação de Cartão:** Os campos de cartão de crédito não possuem validação de formato (ex: número de cartão válido, data de validade futura).
- **Armazenamento de Dados:** Após a "confirmação" do pagamento, não há persistência dos dados do aluguel (quem alugou, qual imóvel, por quanto tempo, etc.) em um banco de dados real (Firestore, por exemplo).

- **Tratamento de Erros:** Não há tratamento de erros robusto para falhas de pagamento ou problemas na seleção de datas (além da validação de data de término).
- **Interface de Usuário:** Embora funcional, a interface pode ser aprimorada com ícones de loading mais visíveis para a geração do Pix e feedback mais claro durante o processo.

9. HouseDetailScreen (lib/screens/details_screen.dart)
HouseDetailScreen é a interface onde o usuário visualiza os detalhes específicos de um imóvel mais claramente.

Propósito
A HouseDetailScreen tem como objetivo principal fornecer ao usuário uma visão aprofundada de um imóvel específico, exibindo todas as informações relevantes e permitindo que ele inicie o processo de aluguel.

Funcionamento Geral

1. **Navegação para a Tela de Detalhes:**
    - Quando o usuário clica no botão "Ver Detalhes" em um card de imóvel na HouseScreen (ou em qualquer outra lista de imóveis), ele é direcionado para a HouseDetailScreen.
    - A HouseDetailScreen recebe o objeto House completo como parâmetro, garantindo que todos os dados do imóvel estejam disponíveis para exibição.
2. **Exibição de Imagens do Imóvel:**
    - A tela exibe uma galeria de imagens do imóvel, idealmente em um carrossel deslizável, permitindo que o usuário visualize diferentes ângulos e aspectos da propriedade.
    - As imagens são carregadas dinamicamente, utilizando a funcionalidade fetchHousePhoto do HousesProvider para obter as fotos associadas ao houseId.
3. **Informações Detalhadas do Imóvel:**
    - **Nome e Endereço:** O nome do imóvel e seu endereço completo (incluindo número, complemento e CEP) seriam exibidos de forma clara.
    - **Tipo de Imóvel:** O tipo da propriedade (ex: "Casa", "Apartamento") é destacado.
    - **Preço Diário:** O preço por dia do aluguel é exibido de forma proeminente.
    - **Descrição Completa:** Uma seção dedicada à descrição detalhada do imóvel, permitindo ao proprietário fornecer todas as informações relevantes sobre a propriedade, suas características e comodidades.

4. **Ações do Usuário:**
    - **Botão "Tenho Interesse em Alugar":** Um botão de ação principal que, ao ser clicado, navegará o usuário para a CheckoutPage, passando o objeto House para que o processo de aluguel possa ser iniciado.
    - **Botão "Entrar em Contato com o Proprietário" :** Um botão para iniciar um chat ou outro meio de comunicação com o proprietário do imóvel, utilizando o ownerUid do objeto House.

Possíveis Melhorias Futuras caso fosse transformada em aplicação de fato:

Nesta parte será descrito apenas algumas partes.

Profile Screen:

1. **Implementar Edição de Perfil:** Desenvolver a funcionalidade para o usuário poder editar seu nome, telefone e, principalmente, fazer upload e gerenciar sua foto de perfil (integrando com Firebase Storage para armazenamento de imagens).
2. **Funcionalidades de Ação:** Implementar as lógicas para "Minhas Chaves de Acesso", "Histórico de Aluguéis" e "Configurações".
3. **Dados Dinâmicos para Estatísticas:** Conectar as estatísticas ("Imóveis Alugados", "Favoritos", "Avaliações") a dados reais do Firestore ou de outra fonte, em vez de usar valores estáticos.
4. **Segurança da Foto de Perfil:** Se a foto de perfil for armazenada em Base64 no Firestore, considere os limites de tamanho de documento do Firestore e a eficiência. Para um aplicativo em larga escala, o Firebase Storage é a solução recomendada para o armazenamento de arquivos grandes como imagens.
5. **Validação de Entrada:** Ao implementar a edição de perfil, adicione validação de entrada para garantir que os dados inseridos pelo usuário sejam válidos.
6. ***Gerenciamento de Senha e E-mail: Adicionar opções para o usuário mudar sua senha e e-mail de forma segura.***

ChatPage:

1. **Notificações Push:** Implementar notificações push (FCM) para alertar os usuários sobre novas mensagens de chat, mesmo quando o aplicativo está em segundo plano.

2. **Funcionalidades de Edição de Perfil:** Desenvolver a lógica para permitir que os usuários atualizem seu nome, telefone e foto de perfil.
3. **Gestão de Chaves de Acesso:** Criar uma tela dedicada para o usuário visualizar e gerenciar suas chaves de acesso aos imóveis alugados.
4. **Histórico de Aluguéis:** Implementar a visualização do histórico de aluguéis do usuário.
5. **Configurações:** Adicionar opções de configuração do aplicativo, como preferências de notificação, privacidade, etc.
6. **Otimização de Imagens de Perfil:** Se as fotos de perfil forem armazenadas em Base64 no Firestore, considere otimizar o tamanho e a qualidade das imagens para melhorar o desempenho. Para um aplicativo em larga escala, o Firebase Storage seria mais adequado para o armazenamento de arquivos.
7. **Pesquisa de Chats:** Para usuários com muitas conversas, uma funcionalidade de pesquisa na ConversationsScreen seria útil.

Check-out:

1. **Integração Real de Pagamento:** Este é o passo mais crítico. Integrar com um gateway de pagamento (ex: Stripe, PagSeguro, Mercado Pago) para processar transações Pix e de cartão de crédito de forma segura e real.
2. **Validação de Entrada:** Implementar validações robustas para os campos de cartão de crédito (número, validade, CVV).
3. **Persistência de Dados do Aluguel:** Após um pagamento bem-sucedido, registrar os detalhes do aluguel (usuário, imóvel, datas, valor, status do pagamento) em um banco de dados (Firestore) para fins de histórico e gerenciamento.
4. **Tratamento de Erros de Pagamento:** Implementar tratamento de erros para cenários de falha de pagamento (cartão recusado, Pix expirado, etc.) e fornecer feedback claro ao usuário.
5. **Confirmação de Reserva:** Após o pagamento, enviar uma confirmação de reserva ao usuário e ao proprietário do imóvel (via e-mail, notificação no app, etc.).
6. **Disponibilidade do Imóvel:** Integrar com um sistema de calendário para verificar a disponibilidade do imóvel nas datas selecionadas antes de permitir o checkout.
7. **Taxas e Descontos:** Adicionar lógica para aplicar taxas de serviço, impostos ou descontos ao valor total.
8. **Melhorias na UI/UX:**

9. Adicionar um ícone de "loading" mais visualmente atraente durante a geração do Pix.
10. Melhorar o feedback visual para campos de entrada de cartão (ex: ícones de bandeira do cartão).