PROMPTS MOTURIAL

Vamos alterar o front end deste app. usei o nome errado da empresa, o correto é MOTURIAL e ela está nascendo, ainda sem lojas, porém com algum trabalho de marketing ja sendo feito no mercado. Não estamos utilizando o azul. Sempre focar em laranja, preto e cimento queimado. O novo logo está em anexo e na frente dele deve aparecer o texto “ASSINATURA DE MOTOS” . Estou anexando algumas imagens, usá-las na confecção do front end. Quanto aos planos de locação, temos PLANO MOTURIAL DIARIO R$ 100,00 / DIA plano MOTURIAL MENSAL R$ 53,00 por dia, mínimo de 30 dias, e plano MOTURIAL SEMESTRAL R$ 47,50 por dia para locações de 6 meses ou mais. O cliente, deve sempre pagar um valor calção conforme a motocicleta que esta escolhendo, sendo assim temos os seguintes valores de calção: Shineray Free 150 = R$ 500,00; SUZUKI DK 160 e JEF 150 = R$ 600,00; Show 175 = R$ 700,00. Estou anexando fotos das motos mas pode buscar as melhores imagens na web. Ao finalizar a escolha do plano o cliente deve ir para uma tela de pagamento, usando a somatória da primeira parcela do plano e o valor calção daquela moto escolhida. O pagamento deve ser feito online usando link de pagamento, pix, ou cartão de crédito, ou ainda boleto bancário orientado ao banco itau. O cliente deve ter a opção de escolher a cor da moto e clicar no nome da cor, alterar a imagem para a cor escolhida. Existe apenas uma unidade aberta neste momento, usar como contato o seguinte : Unidade Rio Preto whatsapp (17) 9 9749 1764 email [riopreto@moturial.com.br](mailto:riopreto@moturial.com.br) e endereço : Avenida Alfredo Antonio de Oliveira, 1781 - Jardim Marajó - são josé do rio preto/sp – CEP 15046-355.

Na topografia dos textos, usar tt nORMS PRO eXPANDED para títulos e TT Norms Pro para textos e subtítulos, sempre nas cores branca ou laranja.

Seguem a seguir algumas informações adicionais sobre o front end e principalmente a arquitetura para criar o backend

Crie um app web completo para **aluguel de motocicletas** no Brasil, com foco em **clientes finais** e operação por **holding → franquias → lojas**. Use:**Front-end**: Next.js/React + TypeScript, páginas responsivas. **Auth/DB/Storage**: **Supabase** (SDK no front). **Server-side**: **Supabase Edge Functions (Deno)** chamadas pelo front (não criar server Node). **Pagamentos**: **Pagar.me v5** (cartão com **token no front**, **PIX** e **boleto**). **2FA**: via **SMS/WhatsApp/Email** usando Edge Function.

**Objetivo do produto**

Usuário final escolhe **loja**, **modelo/cor**, **plano** (dia/mês/ano), faz **cotação**, **reserva**, **paga** (cartão/PIX/boleto) e retira na loja. Pode também **comprar acessórios** e pagar no app.

**Regras de arquitetura**

* + **Front chama apenas**: Supabase SDK para CRUD simples e **Edge Functions** para lógica sensível (cotação, criação de aluguel, fechamento, pagamentos, 2FA). **NUNCA** trafegar dados de cartão pelo backend. O front **tokeniza** com tokenizecard.js (public key) e envia **card\_token** para a Edge Function de pagamento. **RLS** no Postgres garantindo: Cliente vê/edita **apenas seus dados** (rentals/invoices/orders). Staff vê dados da sua **org/franquia/loja**.

1. **Estados**:
   * Moto: available → reserved → rented → available , Aluguel: pending → reserved → active → completed (+ canceled/overdue) , Fatura: pending/paid/failed/refunded/canceled

**Estrutura de dados (tabelas principais)**

* orgs, org\_members, franchises, stores ; profiles(user\_id, full\_name, phone, document\_id, is\_staff) ; bike\_models(brand, model, colors[‘preta’,’vermelha’,’azul’], specs) ; bikes(store\_id, bike\_model\_id, color, plate, vin, status, odometer) ; rental\_plans(org\_id, name, duration\_type: hour/day/month/year, duration\_qty, base\_price\_cents, price\_rules) ;;rentals(user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at, end\_at, status, price\_cents, deposit\_cents, km\_start, km\_end) ; invoices(rental\_id, user\_id, amount\_cents, status, gateway, gateway\_payment\_id, boleto\_url, pix\_qr\_code, pix\_copy\_paste, paid\_at) ; accessories(org\_id, name, price\_cents, stock, image\_url, active) ; accessory\_orders(user\_id, store\_id, status, total\_cents) + accessory\_order\_items(order\_id, accessory\_id, qty, price\_cents) ; twofa\_codes(user\_id, channel, code, expires\_at, consumed)

Se existir mecanismo de import de SQL/migrations, use os arquivos com esses nomes: 001\_init.sql, 002\_rls.sql, 003\_seed.sql (conteúdos conforme descrito acima). Caso o Lovable não suporte migrations, **crie as tabelas via Supabase (UI ou CLI)** e prossiga.

**Edge Functions (criar em /supabase/functions)**

**Utilitários compartilhados (/\_shared)**

* + supa.ts: cria admin client (SUPABASE\_URL, UPABASE\_SERVICE\_ROLE\_KEY) + helper json(). utils.ts: requireEnv(name). pagarme.ts: integração **v5** (base https://api.pagar.me/core/v5), funções: createOrderChargeCardToken({ amount\_cents, description, card\_token, customer, metadata, installments }) createOrderChargePix({ amount\_cents, description, customer, metadata }) createOrderChargeBoleto({ amount\_cents, description, customer, due\_at, metadata })

Autenticação **Basic** com PAGARME\_API\_KEY.

**Funções**

1. 2fa-send: recebe { user\_id, channel: 'sms'|'whatsapp'|'email' }, cria OTP (6 dígitos, 5 min), salva em twofa\_codes e **(TODO)** envia via provedor (log por enquanto). 2fa-verify: { user\_id, code }, valida twofa\_codes, marca consumed. rentals-quote: { rental\_plan\_id } → retorna { price\_cents } usando base\_price\_cents (+futuras price\_rules). rentals-create: { user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at } Checa bike.status === 'available', cria rental(status='reserved'), atualiza moto reserved, cria invoice(pending). Retorna { rental\_id, invoice\_id, amount\_cents }. rentals-close: { rental\_id, km\_end } → fecha aluguel (completed), moto available.
2. payments-create (Pagar.me):
   * Body: { ref\_type:'rental'|'accessory', ref\_id, ethod:'card'|'pix'|'boleto', card\_token?, installments?, due\_at?, customer } Busca invoice (no caso rental), cria ordem/charge no Pagar.me: **Cartão**: payment\_method:"credit\_card", card:{ token: <card\_token> } (auth\_and\_capture) **PIX**: payment\_method:"pix" (retornar pix\_qr\_code e pix\_copy\_paste) **Boleto**: payment\_method:"boleto" (retornar boleto\_url) Atualiza invoices com gateway\_payment\_id, status, campos PIX/boletos. Retorna { invoice\_id, payment\_id, status, pix\_qr\_code, pix\_copy\_paste, boleto\_url }.
3. payments-webhook: recebe eventos do Pagar.me (v5).
   * Ao detectar **paid**: invoices.status='paid', paid\_at=now(). Se tiver rental\_id e rental.status='reserved' → rental='active', bike='rented'. Tratar failed/canceled. **(TODO)** validar assinatura HMAC se habilitada.

**Variáveis de ambiente (Edge)**

SUPABASE\_URL, SUPABASE\_SERVICE\_ROLE\_KEY

PAGARME\_API\_KEY *(obrigatória)*

PAGARME\_API\_BASE = https://api.pagar.me/core/v5 *(default)*

(Opcional) PAGARME\_WEBHOOK\_SECRET para assinar webhooks

(Futuro) credenciais do provedor de SMS/WhatsApp/Email para 2FA

**Front-end (páginas e UX)**

**Home**: hero, CTA “Alugar agora”.

**Lojas**: listar/selecionar loja (cidade/estado, mapa simples opcional).

**Modelos**: listar motos por loja, filtrar por cor/ano; detalhe do modelo (fotos do Storage).

**Planos**: diário, mensal, anual; exibir preço base.

**Cotação/Reserva**: wizard com passos: Loja → Modelo/Cor → Plano → Resumo de preço → **Confirmar** (chama rentals-create).

**Checkout**:

Aba **Cartão**: **incluir** <script src="https://checkout.pagar.me/v1/tokenizecard.js" data-pagarmecheckout-app-id="PUBLIC\_KEY"></script>.

Form com campos do cartão (capturados pelo script), ao sucesso obter card\_token e chamar payments-create com { method:"card", card\_token, installments, customer }.

Aba **PIX**: chamar payments-create com { method:"pix" } e exibir **QR code** + **copia-e-cola**.

Aba **Boleto**: chamar payments-create com { method:"boleto", due\_at } e exibir link do boleto.

**Minha Conta**: dados pessoais, 2FA (enviar/verificar), histórico de aluguéis e faturas.

**Acessórios**: grid de produtos, adicionar ao pedido (para retirada na loja); **(pode ficar com placeholder)**.

**Admin simples (staff)**:

Loja: lista de motos com status, mudar para manutenção, ver reservas ativas.

Relatórios básicos: aluguéis por período, receita por loja.

**Estados e fluxos**

Após payments-create **PIX/Boleto**: exibir instruções; o **webhook** mudará status para paid/active.

Após **cartão aprovado** (resposta paid), ir para tela de confirmação.

A devolução: tela staff para rodar rentals-close com km\_end.

**Componentes/serviços (front)**

supabaseClient.ts (SDK com NEXT\_PUBLIC\_SUPABASE\_URL, NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY).

api/payments.ts: wrappers para supabase.functions.invoke('payments-create') e renderização dos retornos.

useAuth() com listener de sessão; guarda flag is2faVerified após 2fa-verify.

Renderização de pix\_qr\_code (canvas ou <img> se vier em base64) e botão “Copiar PIX”.

**Segurança**

Somente **card\_token** sai do front.

Webhook **idempotente**: atualizar por gateway\_payment\_id.

RLS ativa; profiles somente o próprio usuário.

**Seeds iniciais**

Criar **Org=Holding Motos**, **Franquia SP**, **Loja Paulista**.

bike\_models: **Suzuki DR 160** com cores preta, vermelha, azul.

rental\_plans: Diário (R$ 99,00), Mensal (R$ 1.299,00), Anual (R$ 13.999,00).

**Variáveis de ambiente (Front)**

NEXT\_PUBLIC\_SUPABASE\_URL, NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY.

NEXT\_PUBLIC\_PAGARME\_PUBLIC\_KEY (para data-pagarmecheckout-app-id).

**Aceite (critérios mínimos)**

Usuário consegue **logar**, **ativar 2FA** (simulado), **selecionar loja/modelo/plano**, **criar reserva**, **pagar** (cartão com **token**, PIX, boleto) e ver **status** atualizando no perfil após o webhook.

Staff consegue ver motos e aluguéis da sua loja e **concluir devolução**.

Código organizado, tipado, com serviços dedicados para Edge Functions.

UI limpa, responsiva, com mensagens de erro/sucesso claras.

PROMPT ORIGINAL DO CHATGPT

O aplicativo lovable é capaz de gerar o front end e o Back end do app? Ou tenho que usar outro agente para o backend ?

O **Lovable** é voltado principalmente para **gerar o front-end** do seu aplicativo (UI/UX) de forma rápida, usando IA para transformar ideias ou wireframes em telas funcionais. Ele também consegue criar uma estrutura de **backend básico** (como APIs simples, autenticação, banco de dados conectado), mas não é um backend completo e robusto como os que você desenvolveria manualmente em Node.js, Django, Rails, etc.

Ou seja:

* ✅ **Front-end**: ele cobre muito bem (design + código pronto).
* ⚠️ **Back-end**: gera algo inicial/simples, mas em projetos mais complexos geralmente é preciso **complementar com outro serviço** (ex.: Supabase, Firebase, Hasura, ou até um backend customizado em Node/Java/Python).

👉 Em resumo: você pode começar com o que o Lovable gera, mas para algo profissional e escalável vai precisar integrar ou criar um backend mais robusto com outra tecnologia.

Quer que eu monte um **mapa prático** te mostrando:

1. O que o Lovable cobre no backend.
2. Quais opções extras você pode usar para complementar (ex.: Supabase, Firebase, Backend customizado).
3. Como integrar isso ao front-end gerado pelo Lovable?

Perfeito! Aqui vai um mapa prático — direto ao ponto — para você usar o **Lovable** com (ou sem) um backend extra.

**1) O que o Lovable costuma cobrir no “backend”**

* **CRUD básico** via APIs geradas (REST/GraphQL variam conforme template).
* **Autenticação simples** (login/cadastro) e estados de sessão.
* **Integração inicial com DB** (models/tabelas comuns; seed de dados).
* **Ações server-side leves** (ex.: enviar email, webhooks simples).
* **Deploy “one-click”** em serviços gerenciados do próprio ecossistema.

➡️ Bom para MVP, POCs e apps de baixa complexidade.  
⚠️ Para regras de negócio complexas, filas, jobs, multi-tenant, auditoria, limites de taxa, etc., normalmente você **complementa**.

**2) Opções para complementar o backend (comparativo rápido)**

**A) Supabase** (Postgres gerenciado + Auth + Storage + Edge Functions)

* Quando usar: CRUD rápido, SQL forte, realtime, Auth simples, dashboards.
* Pró: velocidade de entrega, Postgres “de verdade”, fácil integrar com front.
* Contra: funções complexas exigem Edge Functions/Row Level Security bem planejadas.

**B) Firebase** (Auth + Firestore/RTDB + Functions + Storage)

* Quando usar: apps mobile/web com realtime e push; protótipos muito rápidos.
* Pró: escala do Google, SDK excelente.
* Contra: Firestore não é relacional; consultas complexas podem complicar custo.

**C) Hasura (GraphQL sobre Postgres)**

* Quando usar: quer GraphQL “out of the box” performático sobre Postgres.
* Pró: permissões granulares, subscriptions, velocidade.
* Contra: lógica complexa vai para “Actions/Remote Schemas” ou serviços externos.

**D) Backend custom (Node.js/TypeScript – Express/NestJS; Python – Django/FastAPI; Ruby – Rails; Java – Spring)**

* Quando usar: regras de negócio ricas, integrações corporativas, compliance, filas, domínios complexos.
* Pró: total controle, arquiteturas limpas (DDD, Clean), testes e observabilidade.
* Contra: mais tempo e equipe; necessidade de engenharia DevOps.

**E) BaaS/Workflow** (n8n, Inngest, Temporal, Make/Zapier p/ integrações)

* Quando usar: orquestração de rotinas, ETL, jobs assíncronos sem “reinventar roda”.
* Pró: acelera automações e jobs.
* Contra: evite acoplar demais lógica crítica a automações de terceiros.

**3) Como integrar ao front gerado pelo Lovable (passo a passo)**

**Cenário 1 — Lovable + Supabase**

1. **Criar projeto no Supabase** → definir tabelas e RLS (Row Level Security).
2. **Auth**: habilitar provedores (email, Google, etc.).
3. **Camada de Acesso**: no front Lovable, instale o SDK do Supabase e configure SUPABASE\_URL e SUPABASE\_ANON\_KEY via variáveis de ambiente do projeto.
4. **APIs**: use o client (ex.: supabase.from('tabela').select('\*')…) ou crie **Edge Functions** para lógica sensível.
5. **Storage**: uploads de arquivos direto no Storage do Supabase com regras.
6. **Deploy**: manter as chaves seguras; use service\_role **apenas** no server.

**Cenário 2 — Lovable + Hasura (GraphQL)**

1. Conecte o Hasura ao Postgres (Supabase/Neon/RDS).
2. Modele permissões por role (user/admin/anonymous).
3. No front Lovable, use um **GraphQL client** (Apollo/urql) com token JWT.
4. Regras complexas → **Actions** (chamam seu microserviço) e **Remote Schemas**.

**Cenário 3 — Lovable + Backend Custom (NestJS)**

1. Crie API (REST/GraphQL) com rotas versionadas (/v1).
2. Implemente Auth (JWT + refresh), RBAC/ABAC e validação (Zod/Class-Validator).
3. No front Lovable, crie um **API service** (fetch/axios) centralizado.
4. Para uploads, use URLs pré-assinadas (S3/GCS) para não trafegar pelo backend.
5. Observabilidade: logs estruturados, tracing (OpenTelemetry), métricas.

**4) Arquitetura recomendada (pragmática para começar bem)**

* **Fase MVP**: Lovable (front) + **Supabase** (DB/Auth/Storage) + **Edge Functions** p/ lógica sensível.
* **Fase Scale**: Introduza um **serviço backend** (NestJS/Django) para:
  + Regras de negócio, **integrações** (ERP/CRM/pagamentos), **filas** (BullMQ/SQS), **jobs** (Inngest/Temporal), **auditoria** e **webhooks robustos**.
  + Mantenha o Supabase como DB/Auth/Storage (ou migre DB se necessário).

**5) Segurança & boas práticas (essenciais)**

* **Ambientes**: dev, staging, prod separados + variáveis via secrets.
* **Auth**: JWT de curta duração + **refresh tokens**; senhas com bcrypt/argon2.
* **Autorização**: RBAC/ABAC; no Supabase, **RLS** bem testada.
* **Validação**: sempre valide payload no backend (Zod/JOI/Class-Validator).
* **CORS**: restrito aos domínios do app.
* **Logs**: sensíveis ofuscados; retenção e alerta (Datadog/New Relic/Grafana).
* **Backups**: automáticos e testados (restore drill trimestral).

**6) Deploy e CI/CD**

* **Front Lovable**: Vercel/Netlify/Fly (ou o que a plataforma suportar).
* **BaaS**: Supabase/Firebase cuidam do deploy do backend gerenciado.
* **Backend Custom**: Docker + Render/Fly/Heroku/Koyeb (rápido) ou Kubernetes (quando fizer sentido).
* **CI**: GitHub Actions. Pipelines: lint → testes → build → deploy com migrações.

**7) Checklist rápido para você decidir agora**

1. Seu app tem **regras de negócio simples**? → Lovable + Supabase resolvem.
2. Precisa de **multi-tenant, faturamento, workflows, integrações**? → adicione Backend Custom.
3. Precisa de **realtime** e chat/notificações? → Supabase Realtime / Firebase.
4. Quer **GraphQL nativo** e forte controle de permissão em DB relacional? → Hasura.
5. **Crescimento** previsto? → planeje já filas, jobs e observabilidade.

**8) Exemplo de contrato API (para guiar o front do Lovable)**

POST /v1/auth/login

Body: { email, password }

Res: { accessToken, refreshToken, user }

GET /v1/todos?page=1&limit=20

Auth: Bearer <accessToken>

Res: { data: Todo[], total, page, limit }

POST /v1/todos

Body: { title, dueDate }

Res: { id, title, dueDate, completed }

PUT /v1/todos/:id

PATCH /v1/todos/:id

DELETE /v1/todos/:id

No front, mantenha um **client único** que injeta o token e trata 401/403, refresh e retry.

**Recomendação final (opinião prática)**

* **Comece**: Lovable (front) + **Supabase** (Auth/DB/Storage/Edge Functions).
* **Evolua**: quando a lógica ficar “apertada”, introduza um **backend custom** (NestJS ou Django), mantendo o Supabase para o que ele faz muito bem.

Se quiser, me diga o **tipo do seu app** (ex.: marketplace, CRM, portal, e-commerce) e eu já te devolvo uma arquitetura pronta + passos de implementação na ordem certa.

Algumas pessoas usam o lovable para o front end e cursor - extensões vs.code / claude.ai. Como ficaria isso no meu app com o front já feito usando lovable + supabase

Show de bola — dá pra combinar **Lovable (front)** + **Supabase (Auth/DB/Storage/Edge Functions)** com **Cursor/VS Code + Claude.ai** para acelerar o backend e a lógica “server-side”. Aqui vai um guia enxuto, mas completo, para você plugar tudo.

**Visão geral (arquitetura)**

* **Front**: gerado no Lovable (React/Next, por ex.).
* **BaaS**: Supabase (Postgres + Auth + Storage + **Edge Functions**).
* **Backend complementar** (opcional): serviço Node (NestJS/Fastify) ou Python (FastAPI) quando a lógica crescer.
* **Dev AI**: Cursor (IDE com agente) + extensões VS Code; **Claude.ai** para specs, refactors e PR reviews.

**Fluxo de trabalho recomendado**

1. **Repos**

* app-frontend/ (Lovable)
* infra-supa/ (migrations + policies + seeds + edge functions)
* svc-api/ (opcional, backend custom)

1. **Ambientes & .env**

* FRONTEND: NEXT\_PUBLIC\_SUPABASE\_URL, NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY
* EDGE FUNCTIONS / BACKEND: SUPABASE\_SERVICE\_ROLE\_KEY, segredos de terceiros (Stripe, etc.)
* Separe dev, staging, prod.

1. **Desenvolvimento local**

* Instale **Supabase CLI** → supabase start (Postgres local + Studio)
* Rode o **front Lovable** com as chaves NEXT\_PUBLIC\_\*
* Se usar **Edge Functions**: supabase functions serve
* (Opcional) svc-api com docker compose up.

1. **Uso do Cursor/VS Code + Claude**

* No **Cursor**, abra o repo (infra-supa e/ou svc-api) e:
  + Gere **migrations SQL** a partir do schema desejado.
  + Escreva **RLS policies** com base nas regras de negócio.
  + Crie **Edge Functions** (Deno) para lógica sensível.
  + Gere testes E2E (Playwright) e unitários (Vitest/Jest).
* No **Claude.ai**, use como “arquiteto/editor”:
  + Colar contratos de API/erros do front → **gerar endpoints/handlers**.
  + Pedir **refactor seguro** (com diffs) e **revisão de PR**.
  + Produzir **testes e docs** (README, ADRs curtos).

**Integração Front (Lovable) ↔ Supabase (essencial)**

* **SDK** (front):
* import { createClient } from '@supabase/supabase-js'
* export const supabase = createClient(process.env.NEXT\_PUBLIC\_SUPABASE\_URL!, process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!)
* **Auth**: implemente login/logout + listener de sessão; proteja páginas com HOCs/guards.
* **Tipos gerados**: supabase gen types typescript --project-id <id> --schema public > src/types/supabase.ts → evita bug de coluna/campo.

**Banco & RLS (modelo mínimo)**

**Migration (exemplo)**

create table public.todos (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid not null references auth.users(id),

title text not null,

due\_date date,

completed boolean default false,

created\_at timestamptz default now()

);

**RLS**

alter table public.todos enable row level security;

create policy "own read" on public.todos

for select using (auth.uid() = user\_id);

create policy "own write" on public.todos

for insert with check (auth.uid() = user\_id);

create policy "own update" on public.todos

for update using (auth.uid() = user\_id);

create policy "own delete" on public.todos

for delete using (auth.uid() = user\_id);

**Seed (dev)**

insert into public.todos (user\_id, title) values ('00000000-0000-0000-0000-000000000000', 'Primeira tarefa');

**Edge Functions (quando precisa lógica server-side)**

**Ex.: checagem de negócio antes de criar pedido**

// /supabase/functions/create-order/index.ts

import { serve } from 'https://deno.land/std@0.168.0/http/server.ts'

import { createClient } from 'https://esm.sh/@supabase/supabase-js'

serve(async (req) => {

const supabase = createClient(Deno.env.get('SUPABASE\_URL')!, Deno.env.get('SUPABASE\_SERVICE\_ROLE\_KEY')!)

const { user\_id, items } = await req.json()

// validação, cálculo, limites, etc.

// grava em tabelas relacionais…

// retorna payload seguro ao front

return new Response(JSON.stringify({ ok: true }), { headers: { 'content-type': 'application/json' } })

})

Chame do front com **chave anon** via **invoke**:

const { data, error } = await supabase.functions.invoke('create-order', { body: { user\_id, items } })

**Backend custom (quando o Edge não basta)**

* **Quando**: múltiplas integrações (ERP/CRM), filas, workflows complexos, auditoria pesada, multi-tenant, relatórios grandes.
* **Stack** (exemplo):
  + NestJS + Prisma (se Postgres do Supabase) ou TypeORM
  + Auth JWT + refresh; RBAC/ABAC
  + Filas: BullMQ (Redis) / SQS
  + Observabilidade: OpenTelemetry + logs estruturados
* **Contrato de API** (exemplo)
* GET /v1/reports?from=...&to=...
* POST /v1/orders
* O front do Lovable consome via fetch/axios centralizado (um API client).

**Prompts prontos (Cursor/Claude)**

* **Gerar RLS a partir de regra de negócio**

“Gere políticas RLS para a tabela invoices onde o usuário só vê/edita as faturas do seu org\_id, admins veem todas. Considere roles: user, admin. Devolva SQL e testes com cenários de falha.”

* **Criar Edge Function com validação**

“Crie uma Supabase Edge Function issue-refund que valida permissões (apenas admin), consulta payments, chama webhook externo, atualiza refunds, e retorna JSON com status. Inclua tratamento de erros e logs.”

* **Refactor orientado a tipos**

“Refatore este service para TypeScript estrito, crie tipos Result<T, E>, e cubra com testes unitários. Envie diff e riscos.”

**Extensões úteis no VS Code/Cursor**

* **ESLint + Prettier** (qualidade contínua)
* **GitHub Copilot** (complementa Cursor)
* **Supabase** (explorar DB/Policies)
* **REST Client** ou **Thunder Client** (testar APIs)
* **Error Lens** (feedback de linting em linha)

**CI/CD (curto e bom)**

* GitHub Actions:
  + infra-supa: checa SQL, aplica migrações em staging, roda testes de Edge Functions.
  + app-frontend: lint → build → deploy (Vercel/Netlify).
  + svc-api: testes → build Docker → deploy (Render/Fly/Koyeb ou K8s).
* Proteja main com PR + review (use **Claude** para revisar diffs grandes).

**Decisão prática para você agora**

* **Se o front já está no Lovable** e o app é de média complexidade:
  1. Modele o **DB** no Supabase + crie **RLS** (com ajuda do Cursor/Claude).
  2. Escreva **Edge Functions** para a lógica sensível (billing, limites, webhooks).
  3. Só crie um **backend custom** quando sentir dor (integrações/filas/workflows).
  4. Automatize **migrations + seeds + testes** no CI.

Se me disser em 1 linha o **tipo do seu app** (ex.: marketplace B2B, CRM de clínicas, portal de mentoria), eu te entrego um **pacote base** (tabelas, RLS, 2 Edge Functions e as rotas do front) já no formato que você pluga no seu repo.

Estamos falando de um app web que basicamente controla o aluguel de motocicletas. Existe uma holding que vende franquias de lojas de aluguel de motos. O app vai ficar nas mãos dos clientes finais, pessoas comuns que alugam motos. Existem planos de locação por tempo, dias, meses e ano. Existem tipos de motos, iniciarei com a suzuki DR 160 em 3 cores. Mas o sistema deve permitir o cadastro de usuários, autenticação de 2 fatores usando SMS, WhatsApp ou e-mail, cadastro dos planos de locação, cadastro de lojas, cadastro de motos com outras marcas e modelos. Além disso o sistema deve permitir que o usuário pague a conta pelo aplicativo usando cartão de crédito ou débito, pix ou boleto. Ha também um cadastro de acessórios, como calças, capacetes, etc que podem ser comprados e pagos igualmente para retirada na loja física.

Perfeito, Alexandre! Abaixo vai um **pacote-base** para seu app web de **aluguel de motocicletas** usando **Front (Lovable)** + **Supabase (Auth/DB/Storage/Edge Functions)** + (opcional) **serviço backend** quando crescer. Foquei em: multi‐tenant (holding → franquias → lojas), 2FA, planos, frota, pagamentos (cartão, PIX, boleto), e venda de acessórios.

**Arquitetura (visão rápida)**

* **Front** (Lovable): Next/React, consumo via SDK do Supabase + chamadas às Edge Functions.
* **Supabase**:
  + **Auth** (email + OTP) e suporte a 2FA via **Edge Function** com SMS/WhatsApp.
  + **DB Postgres** (RLS ativada).
  + **Storage** (fotos de motos, acessórios, documentos).
  + **Edge Functions** (Deno) p/ lógica: cálculo de preço, criação de aluguel, pagamentos e webhooks.
* **Pagamento (BR)**: escolha 1 gateway com **PIX + cartão + boleto**. Opções comuns: **Pagar.me (Stone)**, **Mercado Pago**, **Iugu**, **Gerencianet/Efí**. (Você pode começar com um e trocar depois via camada de abstração na Edge Function.)

**Modelo de dados (essencial, com multi-tenant)**

Prefixo org\_ para a holding/franquias; cada loja pertence a uma franquia; RLS garante isolamento.

-- ORGANIZAÇÃO / MULTI-TENANT

create table orgs (

id uuid primary key default gen\_random\_uuid(),

name text not null,

created\_at timestamptz default now()

);

create table org\_members (

org\_id uuid references orgs(id) on delete cascade,

user\_id uuid references auth.users(id) on delete cascade,

role text check (role in ('owner','admin','manager','staff','viewer')) not null,

primary key (org\_id, user\_id)

);

create table franchises (

id uuid primary key default gen\_random\_uuid(),

org\_id uuid not null references orgs(id) on delete cascade,

name text not null,

cnpj text,

created\_at timestamptz default now()

);

create table stores (

id uuid primary key default gen\_random\_uuid(),

franchise\_id uuid not null references franchises(id) on delete cascade,

name text not null,

address jsonb, -- {street, number, city, state, zip, lat, lng}

phone text,

opening\_hours jsonb, -- {mon:{open:"08:00", close:"18:00"}, ...}

created\_at timestamptz default now()

);

-- USUÁRIOS (perfil do cliente final e staff)

create table profiles (

user\_id uuid primary key references auth.users(id) on delete cascade,

full\_name text,

phone text,

document\_id text, -- CPF

is\_staff boolean default false,

created\_at timestamptz default now()

);

-- CATÁLOGO DE MOTOS

create table bike\_models (

id uuid primary key default gen\_random\_uuid(),

brand text not null, -- ex: 'Suzuki'

model text not null, -- ex: 'DR 160'

engine\_cc int,

year int,

colors text[] default '{}', -- ex: {'preta','vermelha','azul'}

specs jsonb

);

create table bikes (

id uuid primary key default gen\_random\_uuid(),

store\_id uuid not null references stores(id) on delete cascade,

bike\_model\_id uuid not null references bike\_models(id),

color text not null,

plate text unique,

vin text unique,

status text not null default 'available'

check (status in ('available','reserved','rented','maintenance','inactive')),

odometer int default 0,

created\_at timestamptz default now()

);

-- PLANOS DE LOCAÇÃO

create table rental\_plans (

id uuid primary key default gen\_random\_uuid(),

org\_id uuid not null references orgs(id) on delete cascade,

name text not null, -- ex: "Diário", "Mensal", "Anual"

duration\_type text not null check (duration\_type in ('hour','day','month','year')),

duration\_qty int not null, -- ex: 1, 7, 30, 12

base\_price\_cents int not null, -- preço base

price\_rules jsonb, -- upgrades, franquia km, caução, etc.

active boolean default true

);

-- ALUGUÉIS

create table rentals (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid not null references auth.users(id),

store\_id uuid not null references stores(id),

bike\_id uuid not null references bikes(id),

rental\_plan\_id uuid not null references rental\_plans(id),

start\_at timestamptz not null,

end\_at timestamptz,

status text not null default 'pending'

check (status in ('pending','reserved','active','completed','canceled','overdue')),

price\_cents int,

deposit\_cents int default 0,

km\_start int,

km\_end int,

created\_at timestamptz default now()

);

-- PAGAMENTOS (PIX/cartão/boleto) + faturas

create table invoices (

id uuid primary key default gen\_random\_uuid(),

rental\_id uuid references rentals(id) on delete set null,

user\_id uuid not null references auth.users(id),

amount\_cents int not null,

currency text default 'BRL',

status text not null default 'pending'

check (status in ('pending','paid','failed','refunded','canceled')),

gateway text, -- 'pagarme','mercadopago','iugu','efi'

gateway\_payment\_id text, -- id no PSP

boleto\_url text, -- se aplicável

pix\_qr\_code text, -- se aplicável

pix\_copy\_paste text, -- se aplicável

created\_at timestamptz default now(),

paid\_at timestamptz

);

-- ACESSÓRIOS (venda na retirada)

create table accessories (

id uuid primary key default gen\_random\_uuid(),

org\_id uuid not null references orgs(id) on delete cascade,

name text not null,

description text,

price\_cents int not null,

stock int default 0,

active boolean default true,

image\_url text

);

create table accessory\_orders (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid not null references auth.users(id),

store\_id uuid not null references stores(id),

status text not null default 'pending'

check (status in ('pending','paid','ready','delivered','canceled','refunded')),

total\_cents int not null,

created\_at timestamptz default now()

);

create table accessory\_order\_items (

order\_id uuid references accessory\_orders(id) on delete cascade,

accessory\_id uuid references accessories(id),

qty int not null check (qty > 0),

price\_cents int not null,

primary key (order\_id, accessory\_id)

);

create index on rentals(user\_id);

create index on rentals(store\_id);

create index on bikes(store\_id);

create index on invoices(user\_id);

**RLS (exemplos práticos)**

* **Clientes** só enxergam seus próprios aluguéis/faturas.
* **Staff** enxerga dados da **sua loja/franquia**.
* **Holding** (roles owner/admin) enxerga tudo da organização.

-- Rentals

alter table rentals enable row level security;

create policy "customer\_reads\_own\_rentals"

on rentals for select using (auth.uid() = user\_id);

create policy "customer\_writes\_own\_rentals"

on rentals for insert with check (auth.uid() = user\_id);

-- Exemplo de acesso por staff da loja

create policy "store\_staff\_reads\_store\_rentals"

on rentals for select using (

exists (

select 1

from org\_members om

join stores s on s.id = rentals.store\_id

join franchises f on f.id = s.franchise\_id

where om.user\_id = auth.uid()

and om.org\_id = f.org\_id

and om.role in ('admin','manager','staff')

)

);

Repita a lógica para bikes, invoices, accessory\_orders. Você pode refinar com org\_id e store\_id em cada tabela para simplificar as policies.

**Fluxos críticos**

**1) Autenticação + 2FA (SMS/WhatsApp/email)**

* **Supabase Auth** para login básico (email+link mágico ou email+senha).
* **2FA**: implemente via **Edge Function**:
  1. Usuário loga.
  2. Front chama POST /2fa/send (Edge) informando canal: **sms**, **whatsapp** ou **email**.
  3. Função gera OTP (6 dígitos), salva em tabela twofa\_codes(user\_id, code, channel, expires\_at, consumed), envia via provedor (Zenvia, Twilio, WhatsApp Cloud API, email do Supabase).
  4. Front envia POST /2fa/verify com o código. Se ok, marca consumed=true e sobe um **flag de sessão** (ex.: claim JWT personalizado) p/ liberar rotas sensíveis.

*Tabela simples para 2FA:*

create table twofa\_codes (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid references auth.users(id) on delete cascade,

channel text check (channel in ('sms','whatsapp','email')) not null,

code text not null,

expires\_at timestamptz not null,

consumed boolean default false,

created\_at timestamptz default now()

);

**2) Aluguel (reserva → retirada → devolução)**

1. Cliente escolhe **plano**, **loja**, **moto** (cor, disponibilidade).
2. Front chama **Edge Function rentals/quote** → calcula preço (plano + caução + extras).
3. Cliente confirma → **Edge rentals/create**: cria rental (status reserved), gera invoice pending.
4. Pagamento aprovado (webhook) → rental.status='active', bikes.status='rented'.
5. Devolução → **Edge rentals/close**: registra km\_end, calcula extras (km excedente, multa atraso), gera/atualiza invoice e muda rental p/ completed e bike available.

**3) Pagamentos (PIX, cartão, boleto)**

* **Edge payments/create**: recebe {ref\_type: 'rental'|'accessory', ref\_id, method: 'card'|'pix'|'boleto'}:
  + Cria/atualiza invoice.
  + Chama SDK/HTTP do gateway, retorna client\_secret (cartão), pix (qr/copy&paste), boleto\_url.
* **Edge payments/webhook**: recebe eventos do PSP:
  + paid → invoices.status='paid' + timbra paid\_at.
  + Se for rental → ativa aluguel; se for accessory → order.status='paid'.
  + failed/canceled → atualiza status e notifica.
* **Reembolso**: **Edge payments/refund** (opcional).

**Edge Functions (sugestões de pastas/nomes)**

* functions/2fa-send & functions/2fa-verify
* functions/rentals-quote (regras de preço; lê rental\_plans.price\_rules)
* functions/rentals-create
* functions/rentals-close
* functions/payments-create
* functions/payments-webhook (rota pública)
* functions/inventory-update (manutenção, mudança de status)

*Exemplo mini de função (Deno) para cotação:*

// rentals-quote/index.ts

import { serve } from "https://deno.land/std/http/server.ts";

import { createClient } from "https://esm.sh/@supabase/supabase-js";

serve(async (req) => {

try {

const { rental\_plan\_id, start\_at, duration\_override } = await req.json();

const supabase = createClient(Deno.env.get("SUPABASE\_URL")!, Deno.env.get("SUPABASE\_SERVICE\_ROLE\_KEY")!);

const { data: plan } = await supabase

.from("rental\_plans")

.select("\*")

.eq("id", rental\_plan\_id)

.single();

// cálculo simplificado (ex.: base + extras do price\_rules)

const price = plan.base\_price\_cents;

return new Response(JSON.stringify({ price\_cents: price }), { headers: { "content-type": "application/json" }});

} catch (e) {

return new Response(JSON.stringify({ error: e.message }), { status: 400 });

}

});

**Integração no Front (Lovable)**

* **SDK Supabase** para operações CRUD simples (catálogos, perfil).
* Para ações sensíveis e pagamentos → **chame as Edge Functions**.
* **Sessão com 2FA**: após verify, salve um **estado** (ex.: is2fa=true) em store global (Zustand/Redux) e/ou cookie seguro.

**Pagamentos no Brasil (escolha prática)**

* **Pagar.me (Stone)**: bom para crédito/débito e **PIX**; webhooks estáveis; split de pagamentos (se precisar no futuro).
* **Mercado Pago**: onboarding rápido; PIX forte; boa UX; também boleto.
* **Iugu** / **Efí (Gerencianet)**: tradicionais para **boleto + PIX**.

Estratégia: implemente uma **interface única** na Edge (payments/create) e atrás dela um “driver” por PSP. Assim você troca de gateway sem mexer no front.

**Roadmap de entrega (4 sprints enxutas)**

**Sprint 1 – Fundações**

* Tabelas + RLS (orgs, franchises, stores, bike\_models, bikes, rental\_plans, profiles).
* Auth e perfis; upload de imagens (Storage).
* Listagem/busca de motos por loja.

**Sprint 2 – Aluguel**

* rentals-quote, rentals-create, rentals-close.
* Estados da moto (available→reserved→rented→available).
* Telas: escolher plano/loja/moto, resumo de cotação.

**Sprint 3 – 2FA + Pagamentos**

* 2FA (send/verify).
* payments-create (cartão/PIX/boleto) + payments-webhook.
* Fatura no perfil do usuário.

**Sprint 4 – Acessórios + Operação de Loja**

* Catálogo de acessórios + pedidos.
* Retirada na loja (marcar ready/delivered).
* Relatórios básicos (alugueis por período, receita por loja).

**Variáveis de ambiente (mínimo)**

* NEXT\_PUBLIC\_SUPABASE\_URL
* NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY
* SUPABASE\_URL (Edge)
* SUPABASE\_SERVICE\_ROLE\_KEY (Edge)
* PAYMENTS\_PROVIDER = pagarme|mercadopago|iugu|efi
* PAYMENTS\_SECRET/API\_KEY (do PSP escolhido)
* 2FA\_SMS\_PROVIDER + credenciais (Zenvia/Twilio) ou WhatsApp Cloud API
* APP\_BASE\_URL (para webhooks/links de retorno)

**Recomendação prática para você agora**

1. Suba as **tabelas** e **RLS** acima no Supabase.
2. Implemente as **4 Edge Functions** de base: 2fa-send, 2fa-verify, rentals-quote, rentals-create.
3. Escolha **1 PSP** (eu iria de **Pagar.me** ou **Mercado Pago**) e faça payments-create + payments-webhook.
4. Ligue tudo no front do **Lovable** com telas simples de cotação → pagamento → confirmação.

Se quiser, eu te entrego o **pacote de migrações SQL + skeleton das Edge Functions** já estruturado para colar no seu repositório (nomes de pastas e comandos do Supabase CLI). Quer com **Pagar.me** ou **Mercado Pago** primeiro?

Perfeito — vou te entregar um **pacote-base** já preparado para **Lovable (front)** + **Supabase** + **Pagar.me**.  
Inclui: **migrations SQL**, **seeds da DR 160**, **RLS essenciais**, e **skeleton de Edge Functions** (2FA, cotação/criação/conclusão de aluguel, criação de pagamento e webhook do Pagar.me).

**0) Estrutura de pastas (sugestão)**

/supabase

/migrations

001\_init.sql

002\_rls.sql

003\_seed.sql

/functions

/\_shared

supa.ts

utils.ts

pagarme.ts

/2fa-send

index.ts

/2fa-verify

index.ts

/rentals-quote

index.ts

/rentals-create

index.ts

/rentals-close

index.ts

/payments-create

index.ts

/payments-webhook

index.ts

**1) Migrations (SQL)**

**001\_init.sql**

-- ORGANIZAÇÃO / MULTI-TENANT

create table orgs (

id uuid primary key default gen\_random\_uuid(),

name text not null,

created\_at timestamptz default now()

);

create table org\_members (

org\_id uuid references orgs(id) on delete cascade,

user\_id uuid references auth.users(id) on delete cascade,

role text check (role in ('owner','admin','manager','staff','viewer')) not null,

primary key (org\_id, user\_id)

);

create table franchises (

id uuid primary key default gen\_random\_uuid(),

org\_id uuid not null references orgs(id) on delete cascade,

name text not null,

cnpj text,

created\_at timestamptz default now()

);

create table stores (

id uuid primary key default gen\_random\_uuid(),

franchise\_id uuid not null references franchises(id) on delete cascade,

name text not null,

address jsonb,

phone text,

opening\_hours jsonb,

created\_at timestamptz default now()

);

-- PERFIS (clientes e staff)

create table profiles (

user\_id uuid primary key references auth.users(id) on delete cascade,

full\_name text,

phone text,

document\_id text,

is\_staff boolean default false,

created\_at timestamptz default now()

);

-- CATÁLOGO DE MOTOS

create table bike\_models (

id uuid primary key default gen\_random\_uuid(),

brand text not null,

model text not null,

engine\_cc int,

year int,

colors text[] default '{}',

specs jsonb

);

create table bikes (

id uuid primary key default gen\_random\_uuid(),

store\_id uuid not null references stores(id) on delete cascade,

bike\_model\_id uuid not null references bike\_models(id),

color text not null,

plate text unique,

vin text unique,

status text not null default 'available'

check (status in ('available','reserved','rented','maintenance','inactive')),

odometer int default 0,

created\_at timestamptz default now()

);

-- PLANOS

create table rental\_plans (

id uuid primary key default gen\_random\_uuid(),

org\_id uuid not null references orgs(id) on delete cascade,

name text not null,

duration\_type text not null check (duration\_type in ('hour','day','month','year')),

duration\_qty int not null,

base\_price\_cents int not null,

price\_rules jsonb,

active boolean default true

);

-- ALUGUÉIS

create table rentals (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid not null references auth.users(id),

store\_id uuid not null references stores(id),

bike\_id uuid not null references bikes(id),

rental\_plan\_id uuid not null references rental\_plans(id),

start\_at timestamptz not null,

end\_at timestamptz,

status text not null default 'pending'

check (status in ('pending','reserved','active','completed','canceled','overdue')),

price\_cents int,

deposit\_cents int default 0,

km\_start int,

km\_end int,

created\_at timestamptz default now()

);

-- FATURAS / PAGAMENTOS

create table invoices (

id uuid primary key default gen\_random\_uuid(),

rental\_id uuid references rentals(id) on delete set null,

user\_id uuid not null references auth.users(id),

amount\_cents int not null,

currency text default 'BRL',

status text not null default 'pending'

check (status in ('pending','paid','failed','refunded','canceled')),

gateway text, -- 'pagarme'

gateway\_payment\_id text, -- id no PSP

boleto\_url text,

pix\_qr\_code text,

pix\_copy\_paste text,

created\_at timestamptz default now(),

paid\_at timestamptz

);

-- ACESSÓRIOS

create table accessories (

id uuid primary key default gen\_random\_uuid(),

org\_id uuid not null references orgs(id) on delete cascade,

name text not null,

description text,

price\_cents int not null,

stock int default 0,

active boolean default true,

image\_url text

);

create table accessory\_orders (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid not null references auth.users(id),

store\_id uuid not null references stores(id),

status text not null default 'pending'

check (status in ('pending','paid','ready','delivered','canceled','refunded')),

total\_cents int not null,

created\_at timestamptz default now()

);

create table accessory\_order\_items (

order\_id uuid references accessory\_orders(id) on delete cascade,

accessory\_id uuid references accessories(id),

qty int not null check (qty > 0),

price\_cents int not null,

primary key (order\_id, accessory\_id)

);

-- 2FA

create table twofa\_codes (

id uuid primary key default gen\_random\_uuid(),

user\_id uuid references auth.users(id) on delete cascade,

channel text check (channel in ('sms','whatsapp','email')) not null,

code text not null,

expires\_at timestamptz not null,

consumed boolean default false,

created\_at timestamptz default now()

);

-- Índices úteis

create index on rentals(user\_id);

create index on rentals(store\_id);

create index on bikes(store\_id);

create index on invoices(user\_id);

create index on invoices(gateway\_payment\_id);

**002\_rls.sql**

-- Ativar RLS

alter table rentals enable row level security;

alter table invoices enable row level security;

alter table accessory\_orders enable row level security;

alter table accessory\_order\_items enable row level security;

alter table bikes enable row level security;

alter table stores enable row level security;

alter table rental\_plans enable row level security;

alter table profiles enable row level security;

-- Clientes veem/alteram somente seus próprios dados

create policy "profiles\_self" on profiles

for select using (auth.uid() = user\_id);

create policy "profiles\_self\_upd" on profiles

for update using (auth.uid() = user\_id) with check (auth.uid() = user\_id);

-- Rentals

create policy "rentals\_read\_own" on rentals

for select using (auth.uid() = user\_id);

create policy "rentals\_insert\_own" on rentals

for insert with check (auth.uid() = user\_id);

-- Staff por organização (admin/manager/staff)

-- Exemplo para leitura de rentals por staff de mesma org (via store -> franchise -> org)

create policy "rentals\_staff\_read\_by\_org" on rentals

for select using (

exists (

select 1

from org\_members om

join stores s on s.id = rentals.store\_id

join franchises f on f.id = s.franchise\_id

where om.user\_id = auth.uid()

and om.org\_id = f.org\_id

and om.role in ('owner','admin','manager','staff')

)

);

-- Invoices

create policy "invoices\_read\_own" on invoices

for select using (auth.uid() = user\_id);

-- Orders de acessórios (cliente)

create policy "acc\_orders\_read\_own" on accessory\_orders

for select using (auth.uid() = user\_id);

create policy "acc\_orders\_insert\_own" on accessory\_orders

for insert with check (auth.uid() = user\_id);

-- Catálogo de planos/motos/lojas: leitura aberta (ou por org, se preferir)

create policy "stores\_read\_all" on stores for select using (true);

create policy "bikes\_read\_all" on bikes for select using (true);

create policy "rental\_plans\_read\_all" on rental\_plans for select using (true);

**003\_seed.sql**

-- Org e franquia iniciais (exemplo)

insert into orgs (id, name) values

('00000000-0000-0000-0000-000000000001','Holding Motos');

insert into franchises (id, org\_id, name) values

('00000000-0000-0000-0000-000000000101','00000000-0000-0000-000000000001','Franquia SP');

insert into stores (id, franchise\_id, name, address) values

('00000000-0000-0000-0000-000000000201','00000000-0000-0000-000000000101','Loja Paulista', '{"city":"São Paulo","state":"SP"}'::jsonb);

-- Modelo DR 160 com 3 cores

insert into bike\_models (brand, model, engine\_cc, year, colors, specs)

values ('Suzuki','DR 160',160,2025,ARRAY['preta','vermelha','azul'],'{"abs":"sim"}'::jsonb);

-- Planos base

insert into rental\_plans (org\_id, name, duration\_type, duration\_qty, base\_price\_cents, price\_rules)

values

('00000000-0000-0000-0000-000000000001','Diário','day',1,9900,'{}'::jsonb),

('00000000-0000-0000-0000-000000000001','Mensal','month',1,129900,'{}'::jsonb),

('00000000-0000-0000-0000-000000000001','Anual','year',1,1399900,'{}'::jsonb);

Com o **Supabase CLI**, rode:  
supabase start (local) → supabase db push para aplicar.

**2) Edge Functions — utilitários compartilhados**

**/supabase/functions/\_shared/supa.ts**

import { createClient } from "https://esm.sh/@supabase/supabase-js@2";

export function adminClient() {

const url = Deno.env.get("SUPABASE\_URL")!;

const key = Deno.env.get("SUPABASE\_SERVICE\_ROLE\_KEY")!;

return createClient(url, key, { auth: { persistSession: false } });

}

export function json(data: unknown, status = 200) {

return new Response(JSON.stringify(data), {

status,

headers: { "content-type": "application/json" },

});

}

**/supabase/functions/\_shared/utils.ts**

export function requireEnv(name: string): string {

const v = Deno.env.get(name);

if (!v) throw new Error(`Missing env: ${name}`);

return v;

}

export function assert(condition: any, msg: string) {

if (!condition) throw new Error(msg);

}

**/supabase/functions/\_shared/pagarme.ts**

import { requireEnv } from "./utils.ts";

/\*\*

\* Skeleton de integração com Pagar.me.

\* Ajuste endpoints/payloads conforme o ambiente e versão da API.

\*/

const API\_KEY = requireEnv("PAGARME\_API\_KEY");

const API\_BASE = Deno.env.get("PAGARME\_API\_BASE") || "https://api.pagar.me"; // ajuste se necessário

function authHeaders() {

// Muitos PSPs usam Basic com chave; ajuste conforme docs do Pagar.me

const basic = btoa(`${API\_KEY}:`);

return {

"authorization": `Basic ${basic}`,

"content-type": "application/json",

};

}

export type CreatePaymentInput = {

method: "card" | "pix" | "boleto";

amount\_cents: number;

description?: string;

metadata?: Record<string,string>;

customer?: {

name?: string;

email?: string;

document?: string;

phone?: string;

};

card?: {

token?: string; // tokenização no front (recomendado)

installments?: number;

capture?: boolean;

};

pix?: {

expires\_in\_seconds?: number;

};

boleto?: {

due\_date?: string; // YYYY-MM-DD

};

};

export type CreatePaymentOutput = {

payment\_id: string;

status: "pending" | "paid" | "failed" | "canceled";

boleto\_url?: string;

pix\_qr\_code?: string;

pix\_copy\_paste?: string;

provider\_raw?: unknown;

};

export async function createPayment(input: CreatePaymentInput): Promise<CreatePaymentOutput> {

// TODO: mapear para endpoints reais do Pagar.me:

// - Cartão: criar cobrança/ordem e pagamento com card token

// - PIX: criar pagamento PIX e retornar QR/copy&paste

// - Boleto: criar boleto e retornar URL/código de barras

// Abaixo, um stub ilustrativo (retorna "pending")

const res = await fetch(`${API\_BASE}/stub/payments`, { // <- troque pelo endpoint real

method: "POST",

headers: authHeaders(),

body: JSON.stringify(input),

});

if (!res.ok) {

const err = await res.text();

throw new Error(`Pagar.me error: ${res.status} ${err}`);

}

const data = await res.json();

return {

payment\_id: data.id ?? crypto.randomUUID(),

status: data.status ?? "pending",

boleto\_url: data.boleto?.url,

pix\_qr\_code: data.pix?.qr\_code,

pix\_copy\_paste: data.pix?.copy\_paste,

provider\_raw: data,

};

}

/\*\* Verificação de webhook (HMAC) — ajuste header/algoritmo conforme Pagar.me \*/

export function verifyWebhook(signatureHeader: string | null, bodyRaw: string): boolean {

const secret = requireEnv("PAGARME\_WEBHOOK\_SECRET"); // defina no painel

if (!signatureHeader) return false;

// Exemplo genérico HMAC-SHA256:

const key = new TextEncoder().encode(secret);

const msg = new TextEncoder().encode(bodyRaw);

// @ts-ignore Deno subtle crypto

const hmac = await crypto.subtle.sign("HMAC", await crypto.subtle.importKey(

"raw", key, { name: "HMAC", hash: "SHA-256" }, false, ["sign"]), msg);

const digest = Array.from(new Uint8Array(hmac)).map(b => b.toString(16).padStart(2, "0")).join("");

// Compare com o header (ajuste conforme o padrão do Pagar.me)

return signatureHeader.replace(/^sha256=/, "") === digest;

}

**Importante:** os endpoints/payloads do Pagar.me variam por versão. O arquivo acima é um **skeleton** seguro para você **encaixar a chamada real** (e manter o resto do fluxo idêntico).

**3) Edge Functions — fluxos do app**

**2fa-send/index.ts**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient, json } from "../\_shared/supa.ts";

serve(async (req) => {

try {

const { user\_id, channel } = await req.json();

if (!user\_id || !["sms","whatsapp","email"].includes(channel)) {

return json({ error: "invalid params" }, 400);

}

const code = Math.floor(100000 + Math.random()\*900000).toString();

const expires = new Date(Date.now() + 5\*60\*1000).toISOString(); // 5 min

const sb = adminClient();

await sb.from("twofa\_codes").insert({

user\_id, channel, code, expires\_at: expires

});

// TODO: enviar via provedor (Zenvia/Twilio/WhatsApp Cloud/Email)

// Por ora, apenas log:

console.log(`2FA ${channel} for ${user\_id}: ${code}`);

return json({ ok: true });

} catch (e) {

return json({ error: e.message }, 500);

}

});

**2fa-verify/index.ts**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient, json } from "../\_shared/supa.ts";

serve(async (req) => {

try {

const { user\_id, code } = await req.json();

const sb = adminClient();

const { data, error } = await sb.from("twofa\_codes")

.select("\*").eq("user\_id", user\_id).eq("code", code).eq("consumed", false)

.lte("expires\_at", new Date(Date.now()+1).toISOString()) // só formato; ajusta com lógicas adicionais

.order("created\_at", { ascending: false }).limit(1);

if (error) throw error;

if (!data || !data.length) return json({ ok: false, reason: "invalid\_or\_expired" }, 400);

await sb.from("twofa\_codes").update({ consumed: true }).eq("id", data[0].id);

// Opcional: setar claim/flag de 2FA numa tabela de sessão se você gerenciar isso server-side

return json({ ok: true });

} catch (e) {

return json({ error: e.message }, 500);

}

});

**rentals-quote/index.ts**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient, json } from "../\_shared/supa.ts";

serve(async (req) => {

try {

const { rental\_plan\_id } = await req.json();

const sb = adminClient();

const { data: plan, error } = await sb.from("rental\_plans").select("\*").eq("id", rental\_plan\_id).single();

if (error || !plan) return json({ error: "plan\_not\_found" }, 404);

// TODO: aplicar price\_rules (caução, upgrades, etc.)

const price\_cents = plan.base\_price\_cents;

return json({ price\_cents });

} catch (e) {

return json({ error: e.message }, 500);

}

});

**rentals-create/index.ts**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient, json } from "../\_shared/supa.ts";

serve(async (req) => {

try {

const { user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at } = await req.json();

const sb = adminClient();

// Verifica disponibilidade da moto

const { data: bike } = await sb.from("bikes").select("status").eq("id", bike\_id).single();

if (!bike || bike.status !== "available") return json({ error: "bike\_unavailable" }, 409);

// Cotação simples

const { data: plan } = await sb.from("rental\_plans").select("\*").eq("id", rental\_plan\_id).single();

const price\_cents = plan?.base\_price\_cents ?? 0;

// Cria rental + invoice; muda bike para reserved

const { data: rental, error: rerr } = await sb.from("rentals").insert({

user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at, status: "reserved", price\_cents

}).select("\*").single();

if (rerr) throw rerr;

const { error: berr } = await sb.from("bikes").update({ status: "reserved" }).eq("id", bike\_id);

if (berr) throw berr;

const { data: inv, error: ierr } = await sb.from("invoices").insert({

rental\_id: rental.id, user\_id, amount\_cents: price\_cents, status: "pending", gateway: "pagarme"

}).select("\*").single();

if (ierr) throw ierr;

return json({ rental\_id: rental.id, invoice\_id: inv.id, amount\_cents: price\_cents });

} catch (e) {

return json({ error: e.message }, 500);

}

});

**rentals-close/index.ts**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient, json } from "../\_shared/supa.ts";

serve(async (req) => {

try {

const { rental\_id, km\_end } = await req.json();

const sb = adminClient();

const { data: rental } = await sb.from("rentals").select("\*").eq("id", rental\_id).single();

if (!rental || rental.status !== "active") return json({ error: "invalid\_state" }, 409);

// TODO: calcular excedentes/taxas e, se necessário, gerar invoice adicional

await sb.from("rentals").update({ status: "completed", km\_end, end\_at: new Date().toISOString() }).eq("id", rental\_id);

await sb.from("bikes").update({ status: "available" }).eq("id", rental.bike\_id);

return json({ ok: true });

} catch (e) {

return json({ error: e.message }, 500);

}

});

**payments-create/index.ts (Pagar.me)**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient, json } from "../\_shared/supa.ts";

import { createPayment } from "../\_shared/pagarme.ts";

serve(async (req) => {

try {

const { ref\_type, ref\_id, method, card, pix, boleto } = await req.json();

if (!["rental","accessory"].includes(ref\_type)) return json({ error: "invalid\_ref\_type" }, 400);

if (!["card","pix","boleto"].includes(method)) return json({ error: "invalid\_method" }, 400);

const sb = adminClient();

let invoice;

if (ref\_type === "rental") {

const { data: inv } = await sb.from("invoices").select("\*").eq("rental\_id", ref\_id).order("created\_at", { ascending:false }).limit(1).single();

if (!inv) return json({ error: "invoice\_not\_found" }, 404);

invoice = inv;

} else {

// accessory order -> busque/garanta invoice própria se você decidir criar uma

return json({ error: "accessory\_flow\_not\_implemented" }, 501);

}

const pay = await createPayment({

method,

amount\_cents: invoice.amount\_cents,

description: `Pagamento aluguel ${invoice.rental\_id}`,

metadata: { invoice\_id: invoice.id, rental\_id: invoice.rental\_id ?? "" },

card, pix, boleto

});

await sb.from("invoices").update({

gateway: "pagarme",

gateway\_payment\_id: pay.payment\_id,

pix\_qr\_code: pay.pix\_qr\_code ?? null,

pix\_copy\_paste: pay.pix\_copy\_paste ?? null,

boleto\_url: pay.boleto\_url ?? null,

status: pay.status

}).eq("id", invoice.id);

return json({

invoice\_id: invoice.id,

payment\_id: pay.payment\_id,

status: pay.status,

pix\_qr\_code: pay.pix\_qr\_code,

pix\_copy\_paste: pay.pix\_copy\_paste,

boleto\_url: pay.boleto\_url

});

} catch (e) {

return json({ error: e.message }, 500);

}

});

**payments-webhook/index.ts (Pagar.me)**

import { serve } from "https://deno.land/std/http/server.ts";

import { adminClient } from "../\_shared/supa.ts";

// import { verifyWebhook } from "../\_shared/pagarme.ts"; // ative quando ajustar assinatura

serve(async (req) => {

try {

const raw = await req.text();

// const ok = verifyWebhook(req.headers.get("x-pagarme-signature"), raw); // ajuste nome do header conforme docs

// if (!ok) return new Response("invalid signature", { status: 401 });

const evt = JSON.parse(raw);

const sb = adminClient();

// Ajuste os campos conforme o evento do Pagar.me (status, id do pagamento etc.)

const paymentId = evt?.data?.id ?? evt?.payment\_id;

const status = (evt?.data?.status ?? evt?.status ?? "").toLowerCase();

if (!paymentId || !status) return new Response("bad payload", { status: 400 });

const { data: inv } = await sb.from("invoices").select("\*").eq("gateway\_payment\_id", paymentId).single();

if (!inv) return new Response("invoice not found", { status: 404 });

// Atualiza invoice

const patch: any = { status };

if (status === "paid") patch.paid\_at = new Date().toISOString();

await sb.from("invoices").update(patch).eq("id", inv.id);

// Se for aluguel e pago -> ativa aluguel e marca bike como 'rented'

if (inv.rental\_id && status === "paid") {

const { data: rental } = await sb.from("rentals").select("bike\_id,status").eq("id", inv.rental\_id).single();

if (rental && rental.status === "reserved") {

await sb.from("rentals").update({ status: "active" }).eq("id", inv.rental\_id);

await sb.from("bikes").update({ status: "rented" }).eq("id", rental.bike\_id);

}

}

return new Response("ok");

} catch (e) {

return new Response(e.message, { status: 500 });

}

});

**4) Variáveis de ambiente (Edge Functions)**

* SUPABASE\_URL
* SUPABASE\_SERVICE\_ROLE\_KEY
* PAGARME\_API\_KEY **(obrigatória)**
* PAGARME\_API\_BASE *(opcional; ex.:* [*https://api.pagar.me*](https://api.pagar.me/) *ou URL da versão usada)*
* PAGARME\_WEBHOOK\_SECRET *(para validar assinatura do webhook)*
* Provedor 2FA (se usar SMS/WhatsApp): ex. ZENVIA\_API\_KEY ou TWILIO\_\*

**5) Front (Lovable) — chamadas essenciais**

Cotação:

const { data, error } = await supabase.functions.invoke("rentals-quote", {

body: { rental\_plan\_id }

});

Criar aluguel:

const { data, error } = await supabase.functions.invoke("rentals-create", {

body: { user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at: new Date().toISOString() }

});

Criar pagamento (Pagar.me):

const { data, error } = await supabase.functions.invoke("payments-create", {

body: {

ref\_type: "rental",

ref\_id: rentalId,

method: "pix" // ou "card" (com card.token) / "boleto"

}

});

Webhook (configure no painel do Pagar.me):

POST https://<seu-dominio-ou-url-do-supabase>/functions/v1/payments-webhook

**6) Passos imediatos (checklist)**

1. Rodar supabase db push com os 3 arquivos de migration.
2. Publicar as Edge Functions:
   * supabase functions deploy 2fa-send
   * supabase functions deploy 2fa-verify
   * supabase functions deploy rentals-quote
   * supabase functions deploy rentals-create
   * supabase functions deploy rentals-close
   * supabase functions deploy payments-create
   * supabase functions deploy payments-webhook
3. Definir variáveis de ambiente no Supabase (Secrets) e no Vercel/host do front.
4. No Pagar.me: criar **chave de API**, ativar **webhook** apontando para payments-webhook.
5. Ajustar **\_shared/pagarme.ts** com os **endpoints reais** da versão da API que você usar (PIX/cartão/boleto).
6. (Opcional) Ativar envio real de **2FA** (SMS/WhatsApp/Email) no 2fa-send.

Se quiser, posso **mapear os endpoints específicos do Pagar.me** da versão que você pretende usar (PIX + cartão + boleto) e já devolver o pagarme.ts preenchido com os payloads corretos. Basta me dizer **qual versão/SDK** do Pagar.me você vai adotar (vinculado à Stone/PSP) e o **fluxo de tokenização de cartão** que prefere (token no front ou card hash).

Pronto, Alexandre! Montei um pacote **plug-and-play** com:

* Migrations (001\_init.sql, 002\_rls.sql, 003\_seed.sql)
* Edge Functions prontas (2FA, cotação, criação/fechamento de aluguel, pagamento e webhook)
* Integração **Pagar.me v5** com **token no front** (cartão), **PIX** e **boleto**
* Snippet de front para chamar as funções

👉 **Baixe aqui:** [lovable-supabase-pagarme-starter.zip](about:blank)

**Como usar (checklist rápido)**

1. **Local**
   * supabase start
   * supabase db push (aplica 001..003)
2. **Secrets das Edge Functions** (no Supabase → Project Settings → Secrets)
   * SUPABASE\_URL
   * SUPABASE\_SERVICE\_ROLE\_KEY
   * PAGARME\_API\_KEY *(secret da API)*
   * PAGARME\_API\_BASE = https://api.pagar.me/core/v5 *(opcional, já padrão)*
   * (Opcional) PAGARME\_WEBHOOK\_SECRET *(se validar assinatura)*
3. **Deploy das Functions**
4. supabase functions deploy 2fa-send
5. supabase functions deploy 2fa-verify
6. supabase functions deploy rentals-quote
7. supabase functions deploy rentals-create
8. supabase functions deploy rentals-close
9. supabase functions deploy payments-create
10. supabase functions deploy payments-webhook
11. **Webhook no Pagar.me**
    * Aponte para:  
      POST https://<seu-projeto>.functions.supabase.co/payments-webhook
    * Trate os eventos (paid/failed). O arquivo já atualiza invoice, rental e bike.
12. **Front (Lovable)**
    * Inclua o script de tokenização no checkout:
    * <script src="https://checkout.pagar.me/v1/tokenizecard.js"
    * data-pagarmecheckout-app-id="SUA\_PUBLIC\_KEY"></script>
    * Ao **gerar o token**, chame a função payments-create com { method: "card", card\_token }.
    * Para **PIX** e **boleto**, chame com { method: "pix" } ou { method: "boleto", due\_at }.
    * No pacote há um snippet em frontend-snippets/pagarme-client.ts.

**Observações importantes**

* **Nunca** envie dados de cartão ao backend; use apenas o **card\_token**.
* O **customer** no payload do Pagar.me é obrigatório (ou customer\_id). Preencha com dados do profiles.
* O webhook está genérico o bastante para lidar com order/charge — ajuste mapeamentos conforme os **eventos** que você ativar no dashboard.
* A função payments-create já salva pix\_qr\_code, pix\_copy\_paste e boleto\_url quando retornados.

Quer que eu gere também uma **página de checkout** (React/Next) já com o form de cartão, PIX e boleto, usando o estilo do Lovable e consumindo essas funções?

Escrever num único prompt a ser copiado no lovable para gerar front e Back de acordo com as dicas acima

Aqui vai um **único prompt** para você colar no Lovable e ele gerar **Front + “Backend” (Edge Functions no Supabase)** conforme todo o desenho que combinamos (aluguel de motos + Pagar.me v5 com token no front + PIX + boleto + 2FA).

**PROMPT ÚNICO PARA O LOVABLE**

Crie um app web completo para **aluguel de motocicletas** no Brasil, com foco em **clientes finais** e operação por **holding → franquias → lojas**. Use:

* **Front-end**: Next.js/React + TypeScript, páginas responsivas.
* **Auth/DB/Storage**: **Supabase** (SDK no front).
* **Server-side**: **Supabase Edge Functions (Deno)** chamadas pelo front (não criar server Node).
* **Pagamentos**: **Pagar.me v5** (cartão com **token no front**, **PIX** e **boleto**).
* **2FA**: via **SMS/WhatsApp/Email** usando Edge Function.

**Objetivo do produto**

Usuário final escolhe **loja**, **modelo/cor**, **plano** (dia/mês/ano), faz **cotação**, **reserva**, **paga** (cartão/PIX/boleto) e retira na loja. Pode também **comprar acessórios** e pagar no app.

**Regras de arquitetura**

1. **Front chama apenas**: Supabase SDK para CRUD simples e **Edge Functions** para lógica sensível (cotação, criação de aluguel, fechamento, pagamentos, 2FA).
2. **NUNCA** trafegar dados de cartão pelo backend. O front **tokeniza** com tokenizecard.js (public key) e envia **card\_token** para a Edge Function de pagamento.
3. **RLS** no Postgres garantindo:
   * Cliente vê/edita **apenas seus dados** (rentals/invoices/orders).
   * Staff vê dados da sua **org/franquia/loja**.
4. **Estados**:
   * Moto: available → reserved → rented → available
   * Aluguel: pending → reserved → active → completed (+ canceled/overdue)
   * Fatura: pending/paid/failed/refunded/canceled

**Estrutura de dados (tabelas principais)**

* orgs, org\_members, franchises, stores
* profiles(user\_id, full\_name, phone, document\_id, is\_staff)
* bike\_models(brand, model, colors[‘preta’,’vermelha’,’azul’], specs)
* bikes(store\_id, bike\_model\_id, color, plate, vin, status, odometer)
* rental\_plans(org\_id, name, duration\_type: hour/day/month/year, duration\_qty, base\_price\_cents, price\_rules)
* rentals(user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at, end\_at, status, price\_cents, deposit\_cents, km\_start, km\_end)
* invoices(rental\_id, user\_id, amount\_cents, status, gateway, gateway\_payment\_id, boleto\_url, pix\_qr\_code, pix\_copy\_paste, paid\_at)
* accessories(org\_id, name, price\_cents, stock, image\_url, active)
* accessory\_orders(user\_id, store\_id, status, total\_cents) + accessory\_order\_items(order\_id, accessory\_id, qty, price\_cents)
* twofa\_codes(user\_id, channel, code, expires\_at, consumed)

Se existir mecanismo de import de SQL/migrations, use os arquivos com esses nomes: 001\_init.sql, 002\_rls.sql, 003\_seed.sql (conteúdos conforme descrito acima). Caso o Lovable não suporte migrations, **crie as tabelas via Supabase (UI ou CLI)** e prossiga.

**Edge Functions (criar em /supabase/functions)**

**Utilitários compartilhados (/\_shared)**

* supa.ts: cria admin client (SUPABASE\_URL, SUPABASE\_SERVICE\_ROLE\_KEY) + helper json().
* utils.ts: requireEnv(name).
* pagarme.ts: integração **v5** (base https://api.pagar.me/core/v5), funções:
  + createOrderChargeCardToken({ amount\_cents, description, card\_token, customer, metadata, installments })
  + createOrderChargePix({ amount\_cents, description, customer, metadata })
  + createOrderChargeBoleto({ amount\_cents, description, customer, due\_at, metadata })

Autenticação **Basic** com PAGARME\_API\_KEY.

**Funções**

1. 2fa-send: recebe { user\_id, channel: 'sms'|'whatsapp'|'email' }, cria OTP (6 dígitos, 5 min), salva em twofa\_codes e **(TODO)** envia via provedor (log por enquanto).
2. 2fa-verify: { user\_id, code }, valida twofa\_codes, marca consumed.
3. rentals-quote: { rental\_plan\_id } → retorna { price\_cents } usando base\_price\_cents (+futuras price\_rules).
4. rentals-create: { user\_id, store\_id, bike\_id, rental\_plan\_id, start\_at }
   * Checa bike.status === 'available', cria rental(status='reserved'), atualiza moto reserved, cria invoice(pending).
   * Retorna { rental\_id, invoice\_id, amount\_cents }.
5. rentals-close: { rental\_id, km\_end } → fecha aluguel (completed), moto available.
6. payments-create (Pagar.me):
   * Body: { ref\_type:'rental'|'accessory', ref\_id, method:'card'|'pix'|'boleto', card\_token?, installments?, due\_at?, customer }
   * Busca invoice (no caso rental), cria ordem/charge no Pagar.me:
     + **Cartão**: payment\_method:"credit\_card", card:{ token: <card\_token> } (auth\_and\_capture)
     + **PIX**: payment\_method:"pix" (retornar pix\_qr\_code e pix\_copy\_paste)
     + **Boleto**: payment\_method:"boleto" (retornar boleto\_url)
   * Atualiza invoices com gateway\_payment\_id, status, campos PIX/boletos.
   * Retorna { invoice\_id, payment\_id, status, pix\_qr\_code, pix\_copy\_paste, boleto\_url }.
7. payments-webhook: recebe eventos do Pagar.me (v5).
   * Ao detectar **paid**: invoices.status='paid', paid\_at=now(). Se tiver rental\_id e rental.status='reserved' → rental='active', bike='rented'.
   * Tratar failed/canceled.
   * **(TODO)** validar assinatura HMAC se habilitada.

**Variáveis de ambiente (Edge)**

* SUPABASE\_URL, SUPABASE\_SERVICE\_ROLE\_KEY
* PAGARME\_API\_KEY *(obrigatória)*
* PAGARME\_API\_BASE = https://api.pagar.me/core/v5 *(default)*
* (Opcional) PAGARME\_WEBHOOK\_SECRET para assinar webhooks
* (Futuro) credenciais do provedor de SMS/WhatsApp/Email para 2FA

**Front-end (páginas e UX)**

* **Home**: hero, CTA “Alugar agora”.
* **Lojas**: listar/selecionar loja (cidade/estado, mapa simples opcional).
* **Modelos**: listar motos por loja, filtrar por cor/ano; detalhe do modelo (fotos do Storage).
* **Planos**: diário, mensal, anual; exibir preço base.
* **Cotação/Reserva**: wizard com passos: Loja → Modelo/Cor → Plano → Resumo de preço → **Confirmar** (chama rentals-create).
* **Checkout**:
  + Aba **Cartão**: **incluir** <script src="https://checkout.pagar.me/v1/tokenizecard.js" data-pagarmecheckout-app-id="PUBLIC\_KEY"></script>.
    - Form com campos do cartão (capturados pelo script), ao sucesso obter card\_token e chamar payments-create com { method:"card", card\_token, installments, customer }.
  + Aba **PIX**: chamar payments-create com { method:"pix" } e exibir **QR code** + **copia-e-cola**.
  + Aba **Boleto**: chamar payments-create com { method:"boleto", due\_at } e exibir link do boleto.
* **Minha Conta**: dados pessoais, 2FA (enviar/verificar), histórico de aluguéis e faturas.
* **Acessórios**: grid de produtos, adicionar ao pedido (para retirada na loja); **(pode ficar com placeholder)**.
* **Admin simples (staff)**:
  + Loja: lista de motos com status, mudar para manutenção, ver reservas ativas.
  + Relatórios básicos: aluguéis por período, receita por loja.

**Estados e fluxos**

* Após payments-create **PIX/Boleto**: exibir instruções; o **webhook** mudará status para paid/active.
* Após **cartão aprovado** (resposta paid), ir para tela de confirmação.
* A devolução: tela staff para rodar rentals-close com km\_end.

**Componentes/serviços (front)**

* supabaseClient.ts (SDK com NEXT\_PUBLIC\_SUPABASE\_URL, NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY).
* api/payments.ts: wrappers para supabase.functions.invoke('payments-create') e renderização dos retornos.
* useAuth() com listener de sessão; guarda flag is2faVerified após 2fa-verify.
* Renderização de pix\_qr\_code (canvas ou <img> se vier em base64) e botão “Copiar PIX”.

**Segurança**

* Somente **card\_token** sai do front.
* Webhook **idempotente**: atualizar por gateway\_payment\_id.
* RLS ativa; profiles somente o próprio usuário.

**Seeds iniciais**

* Criar **Org=Holding Motos**, **Franquia SP**, **Loja Paulista**.
* bike\_models: **Suzuki DR 160** com cores preta, vermelha, azul.
* rental\_plans: Diário (R$ 99,00), Mensal (R$ 1.299,00), Anual (R$ 13.999,00).

**Variáveis de ambiente (Front)**

* NEXT\_PUBLIC\_SUPABASE\_URL, NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY.
* NEXT\_PUBLIC\_PAGARME\_PUBLIC\_KEY (para data-pagarmecheckout-app-id).

**Aceite (critérios mínimos)**

* Usuário consegue **logar**, **ativar 2FA** (simulado), **selecionar loja/modelo/plano**, **criar reserva**, **pagar** (cartão com **token**, PIX, boleto) e ver **status** atualizando no perfil após o webhook.
* Staff consegue ver motos e aluguéis da sua loja e **concluir devolução**.
* Código organizado, tipado, com serviços dedicados para Edge Functions.
* UI limpa, responsiva, com mensagens de erro/sucesso claras.

Gere o projeto com essa estrutura, componha as páginas e componentes, crie as Edge Functions descritas (código em Deno), integre o **tokenizecard.js** para cartão e deixe pontos marcados com **TODO** onde for necessária a chave/URL real (Pagar.me, provedor 2FA).