Prompts Otimizados para Lovable + Supabase

Documento consolidado com análise, estratégia de uso e prompts prontos para copiar no Lovable.

Conteúdo incluído: análise dos prompts originais, versão mestre otimizada, sequência por etapas, prompts complementares de segurança, RLS, Edge Functions, correção de bugs e melhoria visual.

Recomendação central: não pedir ao Lovable para construir todo o SaaS em um único comando. Use um prompt-base como contexto e depois execute etapas pequenas, validáveis e seguras.

1. Diagnóstico geral

Os prompts originais têm boa visão de produto: SaaS de chat com IA, autenticação, histórico, streaming, admin, planos, segurança e produção. O principal problema é o excesso de escopo em um único comando. Em Lovable, prompts grandes demais tendem a gerar implementação superficial, RLS incompleta, admin protegido só na interface, chamadas sensíveis mal isoladas ou UI bonita sem backend realmente confiável.

A otimização ideal é transformar a especificação em uma sequência: planejamento, schema/RLS, autenticação, chat persistente, Edge Function de IA, configurações/planos, admin e revisão final. Cada etapa deve ter escopo fechado, restrições explícitas e critérios de aceite.

Pontos críticos que devem sempre aparecer nos prompts:

- Usar Supabase Auth, Postgres, RLS e Edge Functions quando o projeto for Lovable + Supabase.
- Nunca expor service_role ou chaves de IA no frontend.
- Validar role admin no frontend, no backend e no banco.
- Implementar RLS com USING e WITH CHECK.
- Fazer chamadas de IA somente por Edge Function.
- Validar JWT, usuário ativo, limite do plano e ownership da conversa.
- Persistir mensagens e logs de forma consistente.
- Usar prompts por etapas, não um comando único gigante.

2. Prompt-base mestre para Lovable + Supabase

```
Crie um SaaS web chamado [NOME_DO_SAAS], um assistente conversacional com IA, usando Lovable + Supabase.

Atue como arquiteto de software senior especialista em SaaS, Supabase, seguranca, RLS, UX/UI e aplicacoes com I
A.

Objetivo:
Construir um MVP real, seguro e funcional de chat com IA, com autenticacao, historico persistente, streaming de
 respostas, configuracoes de conta, painel admin basico e estrutura preparada para planos free/premium.

Stack obrigatoria:
- React
- TypeScript
- Tailwind CSS
- Supabase Auth
- Supabase Postgres
- Supabase Row Level Security
- Supabase Edge Functions
- Variaveis de ambiente seguras
- Componentes reutilizaveis e responsivos

Regras criticas:
- Nunca exponha service_role no frontend.
- O frontend deve usar apenas a chave publica/anonima do Supabase.
- Chaves de provedores de IA devem ficar somente em secrets server-side.
- Toda chamada sensivel a IA deve passar por Supabase Edge Function.
- Toda protecao de dados deve existir no banco com RLS, nao apenas no frontend.
- Rotas privadas devem validar sessao.
- Rotas admin devem validar role no frontend, no backend e no banco.
- O sistema final nao pode depender de mocks para funcionar.
- Nao copie branding, layout proprietario, textos, icones, logos ou identidade visual de nenhum produto existen
te.

Papeis:
1. Visitante: acessa landing page, login, cadastro e recuperacao de senha. Nao acessa area logada, chat, config
uracoes ou admin.
2. Usuario autenticado: acessa apenas seus proprios dados, cria/lista/renomeia/exclui conversas, envia mensagen
s, recebe respostas da IA, edita configuracoes e visualiza plano/uso.
3. Admin: acessa painel admin protegido, visualiza metricas, busca usuarios, bloqueia/desbloqueia usuarios e vi
sualiza logs. No MVP, nao deve acessar conteudo completo das conversas, apenas metadados.

Funcionalidades do MVP:
- cadastro com e-mail e senha
```

```
- login, logout e recuperacao de senha
- sessao persistente
- rotas privadas protegidas
- dashboard logado
- sidebar com historico de conversas
- criar, abrir, renomear e excluir conversas
- envio de mensagem com Enter
- Shift+Enter para quebra de linha
- bloqueio de mensagem vazia
- prevencao de duplo envio
- resposta da IA em streaming
- persistencia de mensagens
- markdown seguro
- blocos de codigo legiveis
- copiar resposta
- regenerar resposta
- titulo automatico de conversa
- configuracoes de conta
- dark mode e light mode
- estrutura de plano free e premium
- limite de uso por plano
- painel admin basico
- logs de uso e erro

Design:
- SaaS moderno, limpo, premium e original.
- Layout responsivo para desktop, tablet e mobile.
- Sidebar no desktop e drawer no mobile.
- Estados loading, empty, error, success e disabled.
- Feedback visual para acoes importantes.
- Confirmacao para acoes destrutivas.
- Acessibilidade basica: contraste adequado, labels, foco visivel e navegacao por teclado.

Antes de implementar, gere um plano tecnico com:
1. arquitetura
2. rotas
3. estrutura de pastas
4. schema do banco
5. politicas RLS
6. fluxos de autenticacao
7. fluxo de chat
8. Edge Functions necessarias
9. plano de implementacao por etapas
10. criterios de aceite por etapa

Nao implemente tudo de uma vez. Trabalhe por etapas pequenas, validaveis e seguras.
```

3. Modelo de dados recomendado

```
Tabelas minimas:

profiles
- id uuid primary key references auth.users(id) on delete cascade
- full_name text
- role text default 'user'
- plan text default 'free'
- is_active boolean default true
- created_at timestamptz default now()
- updated_at timestamptz default now()

conversations
- id uuid primary key default gen_random_uuid()
- user_id uuid references auth.users(id) on delete cascade
- title text not null
- status text default 'active'
- created_at timestamptz default now()
- updated_at timestamptz default now()

messages
- id uuid primary key default gen_random_uuid()
- conversation_id uuid references conversations(id) on delete cascade
- user_id uuid references auth.users(id) on delete cascade
- role text not null
- content text not null
- tokens_input integer
- tokens_output integer
- provider text
```

```
- model text
- status text default 'completed'
- error_message text
- created_at timestamptz default now()

subscriptions
- id uuid primary key default gen_random_uuid()
- user_id uuid references auth.users(id) on delete cascade
- plan text default 'free'
- status text default 'active'
- start_date timestamptz
- end_date timestamptz
- provider_ref text
- created_at timestamptz default now()
- updated_at timestamptz default now()

usage_logs
- id uuid primary key default gen_random_uuid()
- user_id uuid references auth.users(id) on delete cascade
- conversation_id uuid references conversations(id) on delete set null
- event_type text not null
- details jsonb default '{}'::jsonb
- created_at timestamptz default now()

admin_logs
- id uuid primary key default gen_random_uuid()
- admin_user_id uuid references auth.users(id) on delete cascade
- action text not null
- target_type text not null
- target_id text
- metadata jsonb default '{}'::jsonb
- created_at timestamptz default now()

app_errors
- id uuid primary key default gen_random_uuid()
- user_id uuid references auth.users(id) on delete set null
- context text not null
- message text not null
- metadata jsonb default '{}'::jsonb
- created_at timestamptz default now()

Indices:
- conversations(user_id, updated_at desc)
- messages(conversation_id, created_at)
- messages(user_id, created_at desc)
- usage_logs(user_id, created_at desc)
- app_errors(created_at desc)
- profiles(role), profiles(plan), profiles(is_active)

Constraints:
- profiles.role em: user, admin
- profiles.plan em: free, premium
- conversations.status em: active, archived, deleted
- messages.role em: user, assistant, system
- messages.status em: pending, streaming, completed, failed
- subscriptions.status em: active, canceled, past_due, trialing
```

4. Sequencia ideal de prompts por etapa

Etapa 1 - Planejamento e arquitetura

```
Entre em modo de planejamento. Nao implemente codigo ainda.

Com base no contexto do SaaS [NOME_DO_SAAS], usando Lovable + Supabase, gere um plano tecnico para o MVP.

Entregue:
1. Arquitetura geral do app.
2. Rotas publicas, privadas e admin.
3. Estrutura de pastas.
4. Componentes principais.
5. Schema completo do Supabase.
6. Indices, constraints e relacionamentos.
7. Estrategia de RLS tabela por tabela.
8. Fluxo de autenticacao com Supabase Auth.
9. Fluxo seguro do chat via Edge Function.
```

```
10. Estrategia para streaming.
11. Estrategia para logs, erros e rate limit.
12. Ordem de implementacao recomendada.

Restricoes:
- Nao escreva codigo ainda.
- Nao implemente UI ainda.
- Nao crie tabelas ainda.
- Nao use mocks como solucao final.
- Nao exponha secrets.
- Nao deixe autorizacao apenas no frontend.

Ao final, liste os principais riscos tecnicos e como serao mitigados.
```

Etapa 2 - Schema, migrations e RLS

```
Agora implemente somente a estrutura do Supabase: schema, migrations, indices, constraints, triggers e RLS.

Crie as tabelas: profiles, conversations, messages, subscriptions, usage_logs, admin_logs e app_errors.

Requisitos:
- Usar auth.users como fonte de usuarios.
- profiles.id deve referenciar auth.users(id) com on delete cascade.
- Criar trigger para gerar profile automaticamente apos signup.
- Criar trigger para atualizar updated_at.
- Criar indices para consultas principais.
- Criar constraints para role, plan, status e message role.
- Ativar RLS em todas as tabelas sensiveis.
- Criar politicas RLS com USING e WITH CHECK.
- Usuario comum acessa apenas os proprios dados.
- Admin acessa apenas dados administrativos necessarios.
- Criar funcao auxiliar segura para verificar role admin, evitando recursao de politicas.
- Nao usar service_role no frontend.
- Nao criar interface nesta etapa.

Criterios de aceite:
- RLS esta ativado em todas as tabelas sensiveis.
- Nao existe tabela sensivel sem politica.
- Um usuario nao consegue acessar dados de outro.
- Admin tem acesso administrativo controlado.
- Schema esta pronto para autenticacao, chat, IA, logs e planos.
- Migrations estao organizadas.
```

Etapa 3 - Autenticacao e rotas protegidas

```
Implemente autenticacao real com Supabase Auth e conecte ao schema ja criado.

Escopo:
- Cadastro com e-mail e senha.
- Login.
- Logout.
- Recuperacao de senha.
- Persistencia de sessao.
- Protecao de rotas privadas.
- Redirecionamento correto apos login/logout.
- Criacao automatica de profile apos cadastro.
- Bloqueio de acesso para usuario com is_active = false.
- Validacao de role para rota /admin.
- Tela publica de login, cadastro e recuperacao.
- Area logada base.
- Estado de usuario carregando.
- Mensagens de erro amigaveis.

Nao implementar ainda: chat real, IA, admin completo, pagamentos ou refatoracoes visuais grandes.

Criterios de aceite:
- Usuario cadastra, entra e sai.
- Sessao persiste apos refresh.
- Usuario nao autenticado nao acessa area privada.
- Usuario comum nao acessa /admin.
- Usuario bloqueado nao usa area logada.
- Profile e criado corretamente.
- Nenhuma chave sensivel aparece no frontend.
```

Etapa 4 - Layout logado e chat sem IA real

```
Implemente a area logada e o chat com persistencia, ainda sem chamar IA real.

Escopo:
- Dashboard principal.
- Sidebar com conversas do usuario autenticado.
- Botao Nova conversa.
- Criar, listar, abrir, renomear e excluir conversa com confirmacao.
- Area central de chat e input fixo no rodape.
- Enviar com Enter e Shift+Enter para quebra de linha.
- Impedir mensagem vazia, limitar tamanho do input e prevenir duplo envio.
- Salvar mensagem do usuario.
- Exibir mensagens persistidas.
- Criar resposta temporaria server-side apenas para validar o fluxo.
- Salvar resposta temporaria como mensagem do assistente.
- Gerar titulo automatico a partir da primeira mensagem.
- Loading, empty e error states.
- Auto-scroll inteligente.

Importante:
- A resposta temporaria deve passar pelo mesmo fluxo server-side que depois sera usado pela IA.
- Nao use mock solto no frontend.
- Toda query deve respeitar usuario autenticado e RLS.

Criterios de aceite:
- Usuario cria conversa.
- Mensagem e resposta temporaria sao salvas.
- Historico e reaberto corretamente.
- Renomear e excluir funcionam.
- Usuario nao acessa conversa de outro usuario.
- UI funciona em desktop e mobile.
```

Etapa 5 - Edge Function de IA com streaming

```
Substitua a resposta temporaria por integracao real com IA via Supabase Edge Function.

Escopo:
- Criar Edge Function para chat com IA.
- Validar Authorization Bearer JWT.
- Validar usuario ativo.
- Validar se a conversa pertence ao usuario.
- Validar limite do plano antes da chamada.
- Validar tamanho e conteudo da mensagem.
- Buscar contexto recente da conversa.
- Chamar provedor de IA usando secret server-side.
- Transmitir resposta em streaming para o frontend.
- Exibir resposta em tempo real na UI.
- Persistir resposta final no banco.
- Atualizar status da mensagem para completed ou failed.
- Registrar uso em usage_logs.
- Registrar falhas em app_errors.
- Implementar timeout e fallback amigavel.
- Criar camada de abstracao para trocar o provedor no futuro.
- Adicionar copiar resposta, regenerar resposta, markdown seguro e blocos de codigo legiveis.

Regras:
- Nunca chamar provedor de IA diretamente do frontend.
- Nunca expor chave de IA.
- Nao usar service_role no cliente.
- Nao permitir chamadas para conversa de outro usuario.
- Nao permitir envio simultaneo duplicado.
- Nao gravar resposta parcial como final se o streaming falhar.

Criterios de aceite:
- Resposta aparece em streaming.
- Resposta final fica salva.
- Erros nao quebram a conversa.
- Limite de plano e respeitado.
- Logs de uso sao criados.
- Chaves nao aparecem no frontend.
- Regenerar e copiar funcionam.
```

Etapa 6 - Configuracoes, planos e limites

```
Implemente a tela de configuracoes da conta e a estrutura inicial de planos.

Escopo:
- Tela /settings protegida.
```

```
- Editar nome do usuario.
- Exibir e-mail da conta vindo do Supabase Auth.
- Alterar senha ou enviar link de redefinicao conforme suporte do Supabase Auth.
- Preferencia de tema: claro, escuro e sistema.
- Preparar campo de preferencia de idioma para uso futuro.
- Excluir conta com confirmacao forte.
- Exibir plano atual, limite de uso e uso restante.
- Aplicar limite do plano free antes da chamada de IA.
- Criar placeholder de upgrade sem cobranca real.

Regras:
- Nao implementar Stripe nesta etapa.
- Usuario nao pode alterar seu proprio plano manualmente.
- Plano so pode ser alterado por admin ou integracao futura.
- Acao destrutiva exige confirmacao.

Criterios de aceite:
- Usuario edita nome.
- Tema e salvo.
- Uso aparece corretamente.
- Limite bloqueia novas mensagens quando atingido.
- Usuario nao consegue manipular o proprio plano.
- Excluir conta exige confirmacao.
```

Etapa 7 - Admin basico

```
Implemente o painel admin basico com acesso restrito por role.

Escopo:
- Criar rota /admin.
- Validar role admin antes de renderizar.
- Revalidar permissoes nas queries e funcoes.
- Dashboard com total de usuarios, usuarios ativos, usuarios bloqueados, total de conversas, total de mensagens
, uso por periodo e erros recentes.
- Lista de usuarios com busca.
- Detalhes basicos do usuario: nome, e-mail quando disponivel de forma segura, plano, status, data de criacao,
quantidade de conversas e mensagens.
- Bloquear/desbloquear usuario.
- Registrar acoes em admin_logs.
- Visualizar app_errors e usage_logs agregados.

Regras:
- Usuario comum nunca acessa /admin.
- Admin nao deve ver conteudo completo das conversas no MVP.
- Consultas administrativas devem ser seguras.
- Toda acao administrativa deve registrar admin_user_id.
- Bloqueio de usuario deve impedir uso do chat.

Criterios de aceite:
- Admin acessa painel.
- Usuario comum e bloqueado ou redirecionado.
- Metricas carregam.
- Busca funciona.
- Bloquear/desbloquear funciona.
- Acoes sao registradas em admin_logs.
```

Etapa 8 - Revisao final de seguranca e producao

```
Revise o projeto inteiro como especialista senior em Supabase, SaaS, seguranca, RLS, Edge Functions, UX e confi
abilidade.

Primeiro faca uma auditoria. Depois corrija apenas o necessario.

Audite:
- Supabase Auth e criacao automatica de profiles.
- Protecao de rotas privadas e /admin.
- RLS em todas as tabelas sensiveis.
- Politicas USING e WITH CHECK.
- Risco de acesso cruzado entre usuarios.
- Uso indevido de service_role.
- Exposicao de secrets.
- Edge Functions, validacao de JWT, plano e limite.
- Usuario bloqueado.
- Chat streaming, persistencia, falhas e envio duplicado.
- Sanitizacao de markdown e XSS.
- Validacao client-side e server-side.
```

```
- Logs de uso, erro e admin.
- Performance de queries e paginacao do historico.
- Estados loading, empty e error.
- Responsividade mobile e acessibilidade basica.

Corrija:
- Tabelas sem RLS.
- Politicas permissivas demais.
- Queries sem filtro seguro.
- Rotas sensiveis sem protecao.
- Chaves expostas.
- Chamadas a IA pelo frontend.
- Inputs sem validacao.
- Falhas de tratamento de erro.
- Fluxos quebrados para usuario bloqueado.
- Falhas de limite por plano.
- Problemas obvios de UX ou acessibilidade.

Nao faca refatoracao ampla sem necessidade, mudanca visual grande sem motivo, remocao de funcionalidades existe
ntes ou relaxamento de regras de seguranca.

Ao final, entregue problemas encontrados, correcoes aplicadas, riscos restantes e checklist final de prontidao
para producao.
```

5. Versao curta otimizada para colar direto no Lovable

```
Crie um SaaS web chamado [NOME_DO_SAAS], um assistente conversacional com IA usando Lovable + Supabase.

Atue como arquiteto senior especialista em SaaS, Supabase, RLS, Edge Functions, seguranca, UX/UI e aplicacoes c
om IA.

Stack obrigatoria:
- React
- TypeScript
- Tailwind CSS
- Supabase Auth
- Supabase Postgres
- Supabase RLS
- Supabase Edge Functions
- variaveis de ambiente seguras
- componentes reutilizaveis e responsivos

Objetivo:
Criar um MVP real e funcional de chat com IA, com autenticacao, area logada, historico persistente, streaming d
e respostas, configuracoes de conta, painel admin basico e estrutura preparada para planos free/premium.

Funcionalidades:
- cadastro, login, logout e recuperacao de senha
- sessao persistente
- rotas privadas protegidas
- dashboard logado
- sidebar com historico de conversas
- criar, abrir, renomear e excluir conversas
- envio de mensagem com Enter
- Shift+Enter para quebra de linha
- prevencao de mensagem vazia e duplo envio
- resposta da IA em streaming via Edge Function
- persistencia de mensagens no Supabase
- markdown seguro
- blocos de codigo legiveis
- copiar resposta
- regenerar resposta
- titulo automatico de conversa
- configuracoes de conta
- dark/light mode
- plano free com limite de uso
- estrutura premium preparada
- admin basico com metricas, busca de usuarios, bloqueio/desbloqueio e logs

Banco:
Crie as tabelas: profiles, conversations, messages, subscriptions, usage_logs, admin_logs e app_errors.

Seguranca obrigatoria:
- usar auth.users + profiles
- ativar RLS em todas as tabelas sensiveis
- criar politicas USING e WITH CHECK
- usuario so acessa seus proprios dados
- admin so acessa area admin se role = admin
- validar admin tambem no banco e em Edge Functions
- nunca expor service_role no frontend
- frontend usa apenas chave publica/anonima do Supabase
- chaves de IA ficam apenas em secrets server-side
- toda integracao com IA passa por Edge Function
- validar JWT dentro das Edge Functions
- validar usuario ativo
- validar limite de plano
- impedir acesso a conversa de outro usuario
- registrar uso em usage_logs
- registrar erros em app_errors
- registrar acoes administrativas em admin_logs
- validar inputs no cliente e no servidor
- proteger contra XSS e vazamento de dados

Fluxo de IA:
- frontend chama Edge Function com conversation_id e content
- Edge Function valida usuario, plano, conversa e input
- Edge Function busca contexto recente
- Edge Function chama provedor de IA usando secret
- resposta volta em streaming
- frontend exibe em tempo real
```

```
- resposta final e persistida no banco
- falhas ficam registradas sem quebrar a conversa

UX/UI:
- interface moderna, limpa, premium e original
- nao copiar branding ou identidade visual de terceiros
- desktop, tablet e mobile
- sidebar no desktop
- drawer no mobile
- loading, empty, error, success e disabled states
- acoes destrutivas com confirmacao
- acessibilidade basica

Antes de implementar, gere um plano tecnico com arquitetura, rotas, estrutura de pastas, schema do banco, polit
icas RLS, fluxo de autenticacao, fluxo do chat, Edge Functions necessarias, ordem de implementacao e criterios
de aceite.

Nao implemente tudo de uma vez. Entregue por etapas: arquitetura/schema, migrations/RLS, autenticacao, chat per
sistente, Edge Function com streaming, configuracoes/planos, admin e revisao final de seguranca.
```

6. Prompts complementares

Use estes prompts quando o projeto ja existir e voce quiser endurecer partes especificas sem quebrar o restante.

6.1 Auditoria exclusiva de RLS

```
Audite e corrija exclusivamente a seguranca RLS do Supabase.

Verifique:
- Todas as tabelas sensiveis tem RLS ativado.
- Nao existe politica permissiva demais.
- profiles protege leitura e update por usuario.
- conversations protege select, insert, update e delete por user_id.
- messages protege acesso por ownership da conversa.
- usage_logs nao vaza dados entre usuarios.
- admin_logs e acessivel apenas por admins.
- app_errors e acessivel apenas por admins ou por funcoes seguras.
- Politicas de INSERT usam WITH CHECK.
- Politicas de UPDATE usam USING e WITH CHECK.
- Usuario comum nao consegue criar dados com user_id de outro usuario.
- Admin e validado de forma segura, sem depender apenas do frontend.
- Nao ha uso de service_role no cliente.

Depois:
1. Liste os problemas encontrados.
2. Corrija as politicas necessarias.
3. Explique as politicas finais tabela por tabela.
4. Informe como testar com dois usuarios diferentes.
```

6.2 Auditoria da Edge Function de IA

```
Audite e fortaleza a Edge Function responsavel pelo chat com IA.

A funcao deve:
- Validar Authorization Bearer JWT.
- Rejeitar usuario nao autenticado.
- Rejeitar usuario bloqueado.
- Validar se conversation_id pertence ao usuario.
- Validar limite do plano antes da chamada ao provedor.
- Validar tamanho da mensagem.
- Impedir mensagem vazia.
- Impedir chamadas simultaneas duplicadas.
- Buscar apenas contexto da conversa do usuario.
- Usar secret server-side para a chave da IA.
- Nunca expor chave no frontend.
- Aplicar timeout.
- Fazer streaming da resposta.
- Persistir a resposta final.
- Marcar mensagem como failed em caso de erro.
- Registrar usage_logs.
- Registrar app_errors sem dados sensiveis.
- Retornar erro amigavel ao frontend.
```

```
Nao altere o design da interface nesta etapa.
Nao refatore componentes nao relacionados.
Nao mova chaves para o frontend.
```

6.3 Correcao de bugs sem quebrar o projeto

```
Investigue o problema antes de alterar codigo.

Problema observado:
[DESCREVA O ERRO, COLE A MENSAGEM OU ANEXE PRINT]

Comportamento esperado:
[DESCREVA O QUE DEVERIA ACONTECER]

Comportamento atual:
[DESCREVA O QUE ESTA ACONTECENDO]

Regras:
- Nao faca refatoracao ampla.
- Nao altere telas ou componentes nao relacionados.
- Nao mude schema do banco sem explicar o motivo.
- Nao remova funcionalidades existentes.
- Preserve autenticacao, politicas de acesso e isolamento por usuario.
- Se houver risco de quebrar algo, proponha o plano antes de implementar.

Faca:
1. Identifique a causa provavel.
2. Liste os arquivos ou areas afetadas.
3. Explique a correcao minima.
4. Implemente somente a correcao necessaria.
5. Teste mentalmente os fluxos relacionados.
6. Informe o que foi alterado.
```

6.4 Melhorar UI sem afetar backend

```
Melhore apenas a interface e a experiencia visual desta area:

Area:
[EXEMPLO: /app chat, /settings, /admin, sidebar, tela de login]

Objetivo:
Deixar a interface mais premium, responsiva, clara e consistente com o design system atual.

Pode alterar:
- espacamentos
- hierarquia visual
- estados de loading, empty e error
- responsividade
- contraste
- microinteracoes
- organizacao dos componentes
- textos de apoio

Nao pode alterar:
- autenticacao
- banco de dados
- chamadas de API
- politicas de acesso
- logica de permissoes
- funcoes server-side/edge
- integracao com IA
- schema
- nomes de tabelas ou campos

Criterios de aceite:
- Nenhuma funcionalidade existente quebra.
- Layout melhora em desktop e mobile.
- Acessibilidade basica e preservada.
- Estados vazios e erros ficam mais claros.
- A identidade visual continua original e nao copia nenhum produto existente.
```

7. Recomendacao final de uso

Use o prompt-base como contexto permanente do projeto. Em seguida, rode as etapas na ordem indicada. A ordem mais segura para Lovable + Supabase e: schema/RLS primeiro, autenticacao depois, chat persistente em seguida, IA via Edge Function so depois que ownership e limites estiverem funcionando, e por ultimo admin e hardening final.

Esse fluxo reduz muito o risco de gerar um SaaS visualmente bonito, mas inseguro ou incompleto por baixo.