RAIO-X DE PRODUÇÃO

Raio-X de Produção — Brightside Salons — app de demonstração (empresa fictícia, código aberto)

04/11/2026

Tales Ferraz · engenheiro independente · Curitiba

Escopo: o app web de agendamento (Next.js, TypeScript, PostgreSQL): cadastro e login, agendamento, cancelamento, pagamento do sinal e o webhook do provedor de pagamento, lido a partir do tag vibe-coded do repositório
Fora do escopo: a hospedagem, o provedor de pagamento em si (PayDemo é um provedor fictício simulado dentro do app), envio de e-mail, carga e desempenho

Relatório de exemplo — app de demonstração, empresa fictícia (Brightside Salons). O código revisado é público: https://github.com/tales-andre/ai-built-app-rescue. Os erros foram escritos de propósito para mostrar o método; nenhum cliente real aparece aqui.

Resumo para o dono

O app funciona para quem o usa como o esperado. Os problemas aparecem quando alguém não usa: um cliente curioso, um script, ou o provedor de pagamento reenviando uma notificação. Foram encontrados 9 achados: 3 críticos, 4 de gravidade alta, 2 médios. Os itens críticos e de gravidade alta precisam ser corrigidos antes de o app receber clientes e pagamentos reais. 9 correções já estão aplicadas em pull request, cada uma com um teste que falha no código antigo e passa no novo. A tabela abaixo resume cada achado em linguagem de negócio; a parte técnica de cada um vem na seção seguinte.

Gravidade	O que pode acontecer com o seu negócio	Corrigido?
Crítico	A chave da sua conta de pagamento fica visível para qualquer visitante	Sim*
Crítico	Qualquer pessoa podia ver e cancelar os agendamentos de outros clientes	Sim*
Crítico	Uma mensagem de pagamento falsa podia cobrar cartões, e uma repetida cobrou duas vezes	Sim*
Alto	Os agendamentos eram gravados no horário errado	Sim*
Alto	Os clientes podiam definir o próprio preço	Sim*
Alto	Tentativas ilimitadas de adivinhar a senha	Sim
Alto	A chave que assina o login tinha um valor de fábrica, e o cookie de sessão podia trafegar sem HTTPS	Sim*
Médio	Erros eram escondidos em vez de tratados	Sim*
Médio	Uma pessoa podia acabar com duas contas	Sim*

Gravidade: **crítico** = perda de dinheiro ou de dados de clientes ao alcance de qualquer visitante; **alto** = exige esforço pequeno ou condição comum; **médio** = efeito limitado ou dependente de outra falha; **baixo** = melhoria recomendada. ***** = além da correção no código, há uma ação sua fora do código (seção seguinte).

Três coisas a fazer agora, fora do código

1. Trocar a chave secreta do provedor de pagamento. A antiga foi entregue aos navegadores e deve ser tratada como pública.
2. No painel do provedor de pagamento, comparar os sinais cobrados com os agendamentos e estornar as duplicatas.
3. Corrigir os horários dos agendamentos já gravados (estão deslocados em horas) e unificar as contas duplicadas antes de rodar a nova migração do banco.

Achados, em ordem de gravidade

1. A chave da sua conta de pagamento fica visível para qualquer visitante CRÍTICO

O QUE PODE ACONTECER

Qualquer pessoa que abra o site consegue copiar a chave secreta da conta de pagamento a partir do código da página. Com ela, age em nome do negócio junto ao provedor de pagamento: cria cobranças, emite estornos, lê dados de pagamento dos clientes.

O QUE ESTAVA ERRADO

Chave secreta do provedor de pagamento numa variável NEXT_PUBLIC_, usada em componente de cliente. Onde: `src/app/bookings/pay-deposit-button.tsx, .env.example`.

CORREÇÃO

O checkout passou para uma rota do servidor (/api/checkout) que confere a sessão; a chave vive só numa variável do servidor. Os módulos que guardam segredos importam `server-only`, então uma importação acidental no lado do cliente derruba o build em vez de publicar a chave.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Trocar (rotacionar) a chave no provedor de pagamento. Corrigir o código não desfaz o vazamento de uma chave que já foi entregue aos navegadores.

EVIDÊNCIA

`src/app/bookings/pay-deposit-button.tsx, .env.example`

```
"use client";
// src/app/bookings/pay-deposit-button.tsx
await fetch("/api/paydemo-sim/checkout", {
  method: "POST",
  headers: {
    Authorization: `Bearer ${process.env.NEXT_PUBLIC_PAYDEMO_SECRET_KEY}`,
  },
  body: JSON.stringify({ bookingId }),
});
// Next.js copia toda variável NEXT_PUBLIC_* para o JavaScript entregue ao navegador.
```

TESTE

`tests/findings/01-secret-in-browser.test.ts · e2e/bundle.spec.ts` — falha no código antigo e passa no corrigido.

Referência: A-01 · item 1 do checklist · origem: scan_frontend:NX1 · commit: Fix 1: keep the payment secret key on the server

---

2. Qualquer pessoa podia ver e cancelar os agendamentos de outros clientes CRÍTICO

O QUE PODE ACONTECER

Um cliente logado troca um número numa requisição e vê os agendamentos e as anotações privadas de outro cliente, ou os cancela. Os números de agendamento são sequenciais, então todos os agendamentos ficam ao alcance.

O QUE ESTAVA ERRADO

IDOR: a API confia no userId vindo da query string e do corpo; o cancelamento não confere o dono. Onde: `src/app/api/bookings/route.ts, src/app/api/bookings/[id]/cancel/route.ts`.

CORREÇÃO

O usuário passa a vir do cookie de sessão assinado. O cancelamento só atinge um agendamento do próprio solicitante e responde 404 nos demais casos. Chamadas sem login recebem 401.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Conferir nos registros do servidor, se existirem, cancelamentos que não partiram do dono do agendamento.

EVIDÊNCIA

`src/app/api/bookings/route.ts, src/app/api/bookings/[id]/cancel/route.ts`

```
// src/app/api/bookings/route.ts
export async function GET(req: Request) {
  const userId = Number(new URL(req.url).searchParams.get("userId"));
  ...
    .where(eq(bookings.userId, userId))
// O id do cliente vem da URL (?userId=), não da sessão assinada.
```

TESTE

`tests/findings/02-other-customers-bookings.test.ts · e2e/other-customers-bookings.spec.ts` — falha no código antigo e passa no corrigido.

Referência: A-02 · item 2 do checklist · origem: scan_app:AP1 · commit: Fix 2: take the customer from the session, not from the request

---

3. Uma mensagem de pagamento falsa podia cobrar cartões, e uma repetida cobrou duas vezes CRÍTICO

O QUE PODE ACONTECER

O endereço que recebe as notificações de pagamento aceitava mensagem de qualquer origem. Provedores de pagamento também reenviam notificações quando não têm certeza de que chegaram; cada reenvio cobrava o sinal do cliente de novo. Cobrança em dobro significa estornos, tarifas de chargeback e reclamações públicas.

O QUE ESTAVA ERRADO

Webhook sem verificação de assinatura nem idempotência. Onde: `src/app/api/payments/webhook/route.ts`.

CORREÇÃO

Assinatura HMAC e carimbo de tempo conferidos sobre o corpo bruto; o evento é validado com zod (um evento malformado recebe 400, não um 500 que o provedor reenviaria para sempre); cada id de evento é registrado uma única vez (processed_events); a linha do agendamento é travada e só é cobrada enquanto aguarda o sinal; a cobrança leva uma chave de idempotência.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Comparar os sinais no painel do provedor com os agendamentos; estornar as duplicatas.

EVIDÊNCIA

src/app/api/payments/webhook/route.ts

```
// src/app/api/payments/webhook/route.ts
export async function POST(req: Request) {
  try {
    const event = await req.json();   // sem assinatura, sem validação
    if (event.type === "checkout.completed") {
      ...
      const deposit = await charge({ amountCents: row.depositCents, ... }); // cobra a cada chegada
    }
  } catch (err) { /* ignore */ }
  return Response.json({ received: true });
```

TESTE

`tests/findings/03-payment-webhook.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-03 · item 4 do checklist · origem: scan_webhooks:WH1 + WH2 · commit: Fix 3: verify webhook signatures and charge each deposit once (+ follow-up)

---

4. Os agendamentos eram gravados no horário errado ALTO

O QUE PODE ACONTECER

O servidor lê o horário escolhido pelo cliente no fuso do próprio servidor (UTC na maioria das hospedagens), não no fuso do salão. Um agendamento às 10:00 em Chicago vira 4:00 da manhã. Equipe e clientes veem horários diferentes, e o horário de funcionamento é conferido contra o relógio errado.

O QUE ESTAVA ERRADO

new Date(`${date}T${time}`) usa o fuso horário do servidor. Onde: `src/lib/booking-time.ts`.

CORREÇÃO

Os horários são convertidos a partir de America/Chicago, com horário de verão tratado; o horário de funcionamento usa o relógio do salão; o app exibe o horário do salão.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Os agendamentos já gravados estão deslocados em horas. Corrigir com um script único, depois de conferir alguns à mão.

EVIDÊNCIA

src/lib/booking-time.ts

```
// src/lib/booking-time.ts
export function toAppointmentDate(date: string, time: string): Date {
  return new Date(`${date}T${time}`);   // interpretado no fuso do servidor
}
export function isOpen(start: Date): boolean {
  const hour = start.getHours();         // idem
  ...
```

TESTE

`tests/findings/05-time-zone.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-04 · item 8 do checklist · origem: scan_app:AP5 · commit: Fix 5: read appointment times in the salon's time zone

---

5. Os clientes podiam definir o próprio preço ALTO

O QUE PODE ACONTECER

O formulário de agendamento gravava o que o navegador mandasse. Qualquer pessoa com as ferramentas de desenvolvedor do navegador podia agendar um corte de US$ 55 por US$ 0 ou marcá-lo como já confirmado. Entrada malformada derrubava a requisição em vez de mostrar uma mensagem.

O QUE ESTAVA ERRADO

Atribuição em massa (...body no insert) e nenhuma validação de entrada. Onde: `src/app/api/bookings/route.ts`.

CORREÇÃO

Esquemas zod em todo corpo de requisição (chaves desconhecidas descartadas); um 400 que nomeia o campo errado; preço e status definidos pelo servidor.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Procurar agendamentos cujo preço difere do preço do serviço.

EVIDÊNCIA

`src/app/api/bookings/route.ts`

```
// src/app/api/bookings/route.ts
const body = await req.json();
...
const [booking] = await db.insert(bookings).values({
  priceCents: service?.priceCents,
  status: "pending_deposit",
  ...body,          // o corpo da requisição sobrescreve preço e status
  startsAt,
}).returning();
```

TESTE

`tests/findings/06-input-validation.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-05 · item 5 do checklist · origem: scan_webhooks:WH3 · commit: Fix 6: validate every request body and let the server set prices

---

6. Tentativas ilimitadas de adivinhar a senha ALTO

O QUE PODE ACONTECER

Um script podia testar senhas contra a conta de um cliente na velocidade em que o servidor responde. Senhas reutilizadas ou fracas caem rápido.

O QUE ESTAVA ERRADO

Sem limite de tentativas no login por e-mail e senha. Onde: `src/app/api/auth/login/route.ts`.

CORREÇÃO

Após 5 falhas em 15 minutos, o login responde 429 com Retry-After. A verificação e o registro da tentativa rodam numa única transação sob um advisory lock por conta, antes de conferir a senha, então uma rajada de requisições em paralelo é contada uma a uma.

EVIDÊNCIA

`src/app/api/auth/login/route.ts`

```
// src/app/api/auth/login/route.ts
export async function POST(req: Request) {
  const { email, password } = await req.json();
  const [user] = await db.select().from(users).where(eq(users.email, email));
  if (!user || !(await verifyPassword(password, user.passwordHash))) {
    return Response.json({ error: "Invalid email or password" }, { status: 401 });
  }
// Nenhuma contagem de tentativas antes ou depois da verificação.
```

TESTE

`tests/findings/07-login-rate-limit.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-06 · item 6 do checklist · origem: scan_app:AP3 · commit: Fix 7: pause login after repeated wrong passwords (+ follow-up)

---

7. A chave que assina o login tinha um valor de fábrica, e o cookie de sessão podia trafegar sem HTTPS ALTO

O QUE PODE ACONTECER

Se o servidor fosse publicado sem a variável SESSION_SECRET, a chave que assina as sessões seria um valor escrito no código, igual para qualquer cópia do app. Quem lesse o código conseguiria forjar o cookie de login de qualquer cliente. Além disso, o cookie de sessão não exigia HTTPS (sem o atributo Secure), então podia ser enviado em claro numa rede aberta. Nesse cenário o efeito é crítico; com a variável definida, o risco fica no cookie sem Secure.

O QUE ESTAVA ERRADO

Cookie de sessão: chave de assinatura com fallback fixo no código e sem o atributo Secure. Onde: `src/lib/session.ts`.

CORREÇÃO

SESSION_SECRET passa a ser obrigatória em produção, como os demais segredos (o servidor se recusa a criar ou ler sessões sem ela), e o cookie recebe o atributo Secure em produção. HttpOnly e SameSite=Lax já estavam presentes.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Conferir no painel da hospedagem se SESSION_SECRET está definida com um valor próprio e longo. Se não estava, definir e invalidar as sessões ativas.

EVIDÊNCIA

`src/lib/session.ts`

```
// src/lib/session.ts (antes)
function secret() {
  return process.env.SESSION_SECRET ?? "dev-only-session-secret-change-me";
}
export function sessionCookie(userId: number): string {
  ...
  return `${SESSION_COOKIE}=${value}.${sign(value)}; Path=/; HttpOnly; SameSite=Lax; Max-
Age=${MAX_AGE_SECONDS}`;
  // sem Secure
}
```

TESTE

`tests/session.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-09 · item 10 do checklist · origem: scan_app:AP7 · commit: Require SESSION_SECRET in production (found while fixing 1)

---

8. Erros eram escondidos em vez de tratados MÉDIO

O QUE PODE ACONTECER

O build foi configurado para ignorar erros de código, e um deles era um bug real: o salão aceitava agendamentos aos domingos, quando está fechado. Quando a cobrança do sinal falhava, o app dizia ao provedor de pagamento que estava tudo certo, então a cobrança nunca era refeita e o agendamento ficava sem pagamento.

O QUE ESTAVA ERRADO

ignoreBuildErrors: true e catch {} vazio nas rotas de webhook e de cancelamento. Onde: `next.config.ts, src/lib/booking-time.ts, src/app/api/payments/webhook/route.ts`.

CORREÇÃO

A checagem de tipos voltou ao build e o bug do domingo foi corrigido. O webhook responde 500 quando falha, para que o provedor reenvie (seguro agora que o A-03 está corrigido). O cancelamento deixou de informar sucesso quando a atualização falha.

AÇÃO IMEDIATA, FORA DO CÓDIGO

Procurar agendamentos em domingos e agendamentos parados em "aguardando sinal".

EVIDÊNCIA

`next.config.ts, src/lib/booking-time.ts, src/app/api/payments/webhook/route.ts`

```
// next.config.ts
typescript: {
  // build was failing on some type errors, skip them for now
  ignoreBuildErrors: true,
},
// src/lib/booking-time.ts — o erro de tipo escondido era este:
if (SALON.closedDays.includes(day.toString())) return false; // closedDays é number[]
// src/app/api/payments/webhook/route.ts
} catch (err) { // ignore }
return Response.json({ received: true });
```

TESTE

`tests/findings/08-hidden-errors.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-07 · item 9 do checklist · origem: scan_app:AP6 · commit: Fix 8: stop hiding errors

---

9. Uma pessoa podia acabar com duas contas MÉDIO

O QUE PODE ACONTECER

Dois cadastros no mesmo instante, ou o mesmo e-mail digitado com letra maiúscula, criavam duas contas. Os agendamentos se dividem entre elas e a redefinição de senha vai para a conta errada.

O QUE ESTAVA ERRADO

Corrida entre verificar e inserir; sem índice único; e-mails não normalizados. Onde: `src/app/api/auth/signup/route.ts, src/db/schema.ts`.

CORREÇÃO

E-mails gravados em minúsculas; um índice único sobre lower(email) decide; a requisição perdedora recebe 409.

AÇÃO IMEDIATA, FORA DO CÓDIGO

A migração para se já houver duplicatas. Unificar essas contas primeiro.

EVIDÊNCIA

`src/app/api/auth/signup/route.ts, src/db/schema.ts`

```
// src/app/api/auth/signup/route.ts
const existing = await db.select().from(users).where(eq(users.email, body.email));
if (existing.length > 0) {
  return Response.json({ error: "Email already registered" }, { status: 400 });
}
// ... entre a consulta e o insert outro pedido pode passar; e "Ana@x.com" != "ana@x.com"
const [user] = await db.insert(users).values({ email: body.email, ... }).returning();
```

TESTE

`tests/findings/04-duplicate-signup.test.ts` — falha no código antigo e passa no corrigido.

Referência: A-08 · item 7 do checklist · origem: scan_app:AP4 · commit: Fix 4: one account per email, enforced by the database

---

Correção aplicada

Neste app de demonstração todos os achados foram corrigidos, um por commit, no branch `main` do repositório público. Num Raio-X real, a Fase 2 inclui a correção mais crítica (prioridade: chave no bundle, depois cliente vendo dados de outro, depois webhook, depois permissões no banco); as demais entram na Fase 3.

Pull request: https://github.com/tales-andre/ai-built-app-rescue

Correções aplicadas, uma por commit:

- **Chave secreta do provedor de pagamento numa variável NEXT_PUBLIC_, usada em componente de cliente** (commit Fix 1: keep the payment secret key on the server) — teste: `tests/findings/01-secret-in-`

`browser.test.ts · e2e/bundle.spec.ts`

- **IDOR: a API confia no userId vindo da query string e do corpo; o cancelamento não confere o dono** (commit Fix 2: take the customer from the session, not from the request) — teste: `tests/findings/02-other-customers-bookings.test.ts · e2e/other-customers-bookings.spec.ts`
- **Webhook sem verificação de assinatura nem idempotência** (commit Fix 3: verify webhook signatures and charge each deposit once (+ follow-up)) — teste: `tests/findings/03-payment-webhook.test.ts`
- **new Date(`${date}T${time}`) usa o fuso horário do servidor** (commit Fix 5: read appointment times in the salon's time zone) — teste: `tests/findings/05-time-zone.test.ts`
- **Atribuição em massa (...body no insert) e nenhuma validação de entrada** (commit Fix 6: validate every request body and let the server set prices) — teste: `tests/findings/06-input-validation.test.ts`
- **Sem limite de tentativas no login por e-mail e senha** (commit Fix 7: pause login after repeated wrong passwords (+ follow-up)) — teste: `tests/findings/07-login-rate-limit.test.ts`
- **Cookie de sessão: chave de assinatura com fallback fixo no código e sem o atributo Secure** (commit Require SESSION_SECRET in production (found while fixing 1)) — teste: `tests/session.test.ts`
- **ignoreBuildErrors: true e catch {} vazio nas rotas de webhook e de cancelamento** (commit Fix 8: stop hiding errors) — teste: `tests/findings/08-hidden-errors.test.ts`
- **Corrida entre verificar e inserir; sem índice único; e-mails não normalizados** (commit Fix 4: one account per email, enforced by the database) — teste: `tests/findings/04-duplicate-signup.test.ts`

Cada teste falha no código anterior à correção e passa depois dela; é isso que prova que o problema era real e que ficou corrigido.

Como rodar:

```
npm ci
npm test                # código corrigido: todos os testes passam
npm run test:before     # os mesmos testes contra o tag vibe-coded: os testes dos achados falham
git log --oneline vibe-coded..main   # um commit por correção, cada um nomeando o seu teste
```

Checklist dos 15 itens

#	Item verificado	Situação	Observação
1	Chave secreta visível no código entregue ao navegador (bundle, variáveis NEXT_PUBLIC_* / VITE_*)	achado	ver A-01
2	Um cliente conseguir ver ou alterar dados de outro (id do usuário vindo da requisição, não da sessão)	achado	ver A-02
3	Permissões no banco (Supabase RLS): tabelas sem RLS, USING (true), políticas que só checam "está logado", SECURITY DEFINER, user_metadata	fora do escopo	o app é o único cliente do banco; o acesso é controlado na API, não por RLS. Para apps em que o navegador fala direto com o banco (Supabase), ver github.com/tales-andre/supabase-rls-tenant-audit
4	Webhook de pagamento sem verificação de assinatura; cobrança duplicada em reenvio	achado	ver A-03

#	Item verificado	Situação	Observação
5	Preço ou valor definido pelo cliente; entrada sem validação	achado	ver A-05
6	Tentativas ilimitadas de senha	achado	ver A-06
7	Cadastro duplicado (e-mail em maiúscula/minúscula, condição de corrida)	achado	ver A-08
8	Fuso horário errado em agendamentos e prazos	achado	ver A-04
9	Erros escondidos (build ignorando erros de tipo, catch vazio, webhook engolindo falha)	achado	ver A-07
10	Cookie de sessão sem Secure/HttpOnly/SameSite; configuração de autenticação	achado	ver A-09
11	Cabeçalhos de segurança, CORS e cookies no ambiente público	não verificável sem um endereço público	o app de demonstração não está publicado; não houve varredura externa
12	Segredos no repositório e no histórico do git	verificado, ok	histórico do git lido; .env.example traz apenas valores de demonstração
13	Dependências com vulnerabilidade conhecida	verificado, ok	npm audit aponta um aviso moderado em drizzle-kit (ferramenta de desenvolvimento, via esbuild antigo); não faz parte do app que roda em produção
14	Backup e migrações do banco (existe? está versionado? quem restaura?)	verificado, ok	migrações versionadas em drizzle/; banco embutido (PGlite) na demonstração, então não há backup a restaurar
15	Deploy e acesso: quem tem acesso ao projeto, preview apontando para produção, 2FA nas contas de infraestrutura	não verificável sem resposta do dono	num cliente real, respondido pelo questionário de backup e acesso que acompanha o e-mail de escopo

Legenda: **verificado, ok** = li o código e não encontrei o problema; **achado** = há um item na seção de achados; **não verificável sem…** = depende de algo fora do que recebi (painel do provedor, resposta do dono); **fora do escopo** = combinado antes de começar.

O que não mexi e por quê

- **Os números de agendamento continuam sequenciais.** A correção é a verificação de dono; ids aleatórios só esconderiam os números, e trocá-los muda todo link já enviado aos clientes.
- **A pausa do login é por conta.** Quem souber o e-mail de um cliente consegue pausar o login dele por 15 minutos. Limitar por endereço de rede é tarefa da camada na frente do app (hospedagem ou CDN), e um CAPTCHA após algumas falhas é decisão de produto.

- **Sem verificação de agendamento no passado ou de dois agendamentos no mesmo horário.** São regras de negócio (quantas cadeiras, com quanta antecedência) a combinar com o dono, não erros no código como está escrito.
- **Sem token CSRF.** O cookie de sessão é SameSite=Lax, então os navegadores não o enviam em POST de outro site, e toda ação é um POST. Vale acrescentar um token se o app precisar atender navegadores muito antigos.
- **Redefinição de senha e verificação de e-mail.** Não existem neste app; são funcionalidades, não correções.
- **Um aviso moderado numa ferramenta só de desenvolvimento.** drizzle-kit depende de um esbuild antigo. Não faz parte do app que roda em produção.
- **Row Level Security.** O app é o único cliente do seu banco, então o acesso é controlado na API. Para apps em que o navegador fala direto com o banco (Supabase), ver github.com/tales-andre/supabase-rls-tenant-audit.

Orçamento do conserto (Fase 3)

Itens que entram no conserto (todos os críticos e altos pendentes), cada um em um pull request pequeno com teste:

- Chave do provedor de pagamento fora do código entregue ao navegador (A-01)
- Usuário vindo da sessão em todas as rotas; cancelamento só do próprio agendamento (A-02)
- Webhook com assinatura, validação, registro de evento e chave de idempotência (A-03)
- Horários no fuso do salão, com script de correção dos agendamentos já gravados (A-04)
- Validação de todo corpo de requisição; preço e status definidos pelo servidor (A-05)
- Limite de tentativas de login por conta (A-06)
- SESSION_SECRET obrigatória e cookie Secure em produção (A-09)

Preço fechado	R$ 2.500
Prazo	5 dias úteis
Forma de pagamento	50% ao começar e 50% na entrega, por Pix
Validade desta proposta	15 dias a contar da data deste relatório

O preço cobre apenas os itens listados; o que aparecer de novo durante o conserto é relatado e orçado à parte. Recibo em nome de pessoa física; não emito nota fiscal eletrônica.

Neste exemplo o conserto já está feito no repositório público; a tabela mostra como o orçamento da Fase 3 apareceria num relatório real com 7 itens críticos e altos pendentes.

Como este relatório foi feito e seus limites

- O diagnóstico combinou scanners automáticos (busca por padrões conhecidos no código, nas migrações do banco e nas rotas de pagamento) com leitura do código, arquivo por arquivo, nos pontos que os scanners marcaram para revisão.
- **Um resultado limpo não prova que o app é seguro.** Este relatório diz o que foi verificado e como; o que não foi verificado está marcado como tal no checklist.

- Checagens no ambiente público (o que um navegador comum vê ao abrir o site) só foram feitas com a sua autorização por escrito, e só do tipo que um navegador faz. Nesta entrega não houve varredura do ambiente público.
- Nada foi executado contra dados de produção. Nenhum teste rodou contra o seu ambiente ao vivo; os testes rodaram em cópia local do código.
- Os achados foram ordenados pela gravidade do efeito sobre o negócio, não pela dificuldade da correção.

Dados e confidencialidade

- Acesso somente leitura ao repositório e ao esquema do banco; nenhuma senha de produção nem dado de cliente foi solicitado ou recebido.
- Nada do seu código ou dos seus dados entra em repositório público. Neste exemplo o código é aberto por decisão do autor; num relatório real, nada do cliente é publicado.
- Os arquivos recebidos ficam em máquina própria e são apagados 7 dias após a entrega, salvo pedido seu.
- Esclarecimentos sobre qualquer achado deste relatório estão incluídos por 7 dias, por e-mail.

Relatório de exemplo — app de demonstração, empresa fictícia (Brightside Salons). Um relatório real segue exatamente esta estrutura, com os achados do seu app.