Mosai
SECURITY

CHECKLIST DE SEGURANÇA PARA SAAS CRIADO COM IA

12 pontos para revisar antes de colocar usuários reais no sistema

UM PRESENTE DA MOSAI SECURITY

Antes de começar

A IA mudou a velocidade com que sistemas são criados. Hoje, uma pessoa sozinha consegue montar login, painel, área de cliente, pagamento, upload, banco de dados e colocar um produto no ar em poucos dias.

Isso é incrível.

Mas fazer funcionar e estar pronto para receber usuários reais são coisas diferentes.

Um sistema pode ter uma tela bonita, login funcionando e deploy no ar, mas ainda assim:

- mostrar dados demais no navegador
- permitir que um usuário acesse informações de outro
- expor chaves privadas
- deixar um painel administrativo praticamente sem proteção
- guardar informação sensível nos logs
- publicar arquivos que deveriam ser privados
- confiar no frontend para decisões que deveriam acontecer no servidor

O problema é que quase nada disso aparece na interface. A página carrega, o botão funciona, o login entra, o dashboard abre. E mesmo assim o risco pode estar ali.

Este checklist foi feito para te ajudar a olhar para esses pontos antes de colocar usuários reais dentro do sistema.

Não é uma auditoria completa. Não é uma promessa de segurança. Não substitui uma análise técnica mais profunda. É uma pausa antes de publicar.

Como usar este checklist

Escolha uma parte importante do seu sistema. Pode ser o login, o cadastro, o painel administrativo, a área de cliente, o upload de arquivos, a integração com pagamento, a API, ou qualquer fluxo que lide com dados reais.

Depois passe pelos 12 pontos. Para cada um, responda:

- Isso existe no meu sistema?
- Eu sei como foi implementado?
- Eu consigo testar isso?
- A IA explicou a decisão ou apenas fez funcionar?
- Se estiver errado, o que pode acontecer?

Você não precisa saber tudo de segurança para começar. Mas precisa reconhecer quando algo importante foi publicado sem que você entenda como está protegido.

Se em algum ponto a resposta for "não sei", não ignore. Esse é exatamente o lugar onde vale parar e conferir.

1 Abra o DevTools e veja o que o navegador recebeu

Antes de pensar em ataques complexos, abra as ferramentas do próprio navegador. No Chrome ou Edge: clique com o botão direito em qualquer lugar da página, escolha inspecionar e olhe as abas Network, Application e Console.

O navegador mostra muito mais do que aparece na tela. Uma página pode exibir apenas o nome de um cliente, enquanto a resposta da API trouxe também email, telefone, identificador interno, plano contratado e observações administrativas que não precisariam estar ali.

Se o navegador recebeu, o usuário pode ver. Mesmo que a interface não mostre.

O que observar

- Na aba Network, clique nas requisições e veja o que veio na resposta.
- Na aba Application, confira se há dados salvos em localStorage, sessionStorage ou cookies.
- Na aba console, veja se o sistema exibe erros técnicos ou informações internas.

Perguntas para revisar

- A API está retornando mais dados do que a tela precisa?
- Existem emails, telefones ou informações pessoais nas respostas?
- Algum token aparece no navegador?
- Há dados de usuários diferentes na mesma resposta?
- O console exibe erros técnicos detalhados?
- Existe informação sensível salva no localStorage?

Sinal de alerta: Se a API envia todos os dados e o frontend apenas esconde parte deles, o dado já foi entregue ao navegador. Esconder na tela não é o mesmo que proteger.

O que marcar

* Abri o DevTools no meu sistema.
* Revisei as respostas das principais APIs.
* Verifiquei localStorage, sessionStorage e cookies.
* Não encontrei dados sensíveis entregues sem necessidade.

2 Nunca trate segredo como dado de frontend

É comum pedir para a IA integrar pagamentos, email, banco de dados ou algum serviço externo. E é justamente nessa hora que muitos projetos acabam colocando uma chave privada no lugar errado.

Exemplos de informações que não deveriam chegar ao navegador:

- chave privada de API
- token administrativo
- chave de serviço do banco
- segredo de webhook
- credencial de envio de email
- senha de banco de dados
- token com permissão elevada

Em projetos frontend, variáveis com prefixos como NEXT_PUBLIC_ ou VITE_ são automaticamente expostas para o navegador. Isso é útil para configurações públicas, mas não é o lugar certo para segredos.

Regra simples: se está no JavaScript entregue ao navegador, trate como público.

Perguntas para revisar

- Alguma chave privada está em arquivo do frontend?
- Existe token sensível em variável pública?
- O navegador chama diretamente uma API usando credencial privada?
- Alguma chave aparece no código, no DevTools ou no histórico do projeto?
- O backend é responsável pelas chamadas realmente sensíveis?

Sinal de alerta: Se uma chave encontrada no navegador permite consultar dados privados, enviar email, administrar banco ou alterar informações, ela está no lugar errado.

O que marcar

* Verifiquei variáveis públicas do projeto.

* Não existem chaves sensíveis no frontend.

* Integrações críticas passam pelo backend.

* Nenhum segredo aparece no DevTools ou no código entregue ao navegador.

3 Login funcionando não significa que o acesso está correto

Um dos enganos mais comuns é achar que, se o usuário fez login, o sistema já está protegido.

Login responde apenas uma pergunta: quem é essa pessoa?

Ainda falta responder outra: essa pessoa pode acessar exatamente este dado ou executar esta ação?

Um usuário pode estar corretamente autenticado e ainda assim conseguir acessar informação que pertence a outra conta.

Exemplo simples

Seu sistema tem uma página como /clientes/123. Um usuário legítimo acessa os dados dele nessa página. Mas o que acontece se ele trocar manualmente o número na URL para /clientes/124?

Se aparecer informação de outra pessoa, o login estava funcionando, mas a permissão falhou.

O mesmo vale para APIs

Não adianta esconder um botão na interface se o endpoint continua aceitando a ação. Não adianta remover o link do painel se a rota abre quando alguém digita o endereço. A validação precisa acontecer no servidor.

Perguntas para revisar

- Cada rota protegida valida o usuário no servidor?
- A API verifica se o registro pertence àquele usuário?
- Usuário comum consegue chamar alguma ação administrativa?
- Trocar um ID na URL muda os dados exibidos?
- Esconder um botão é a única proteção existente?

Sinal de alerta: Se o sistema confia no frontend para decidir quem pode acessar o quê, essa proteção pode ser contornada por qualquer pessoa que abra o DevTools.

O que marcar

* Testei URLs com IDs diferentes.

* A API valida acesso ao recurso solicitado.

* O servidor controla as permissões.

* Usuário comum não acessa funções administrativas.

4 Entregue ao navegador apenas o necessário

Muitos vazamentos não acontecem porque alguém invadiu o sistema. Acontecem porque o próprio sistema enviou informação demais.

Imagine uma tela que precisa mostrar apenas nome, status e data de criação. Mas a resposta da API traz também email, telefone, documento, endereço, observações internas, dados de pagamento, informações de outros usuários e permissões administrativas.

Talvez nada disso apareça visualmente. Mas está no navegador.

O erro mais comum

Pedir ao backend o objeto completo e deixar o frontend decidir o que mostrar. Isso é cômodo durante o desenvolvimento, mas pode expor informações desnecessárias.

Cada tela deve receber somente os dados necessários para cumprir a sua função.

Perguntas para revisar

- A tela recebe campos que nunca utiliza?
- A API retorna objetos completos por comodidade?
- Informações internas aparecem na resposta, mesmo escondidas na interface?
- Dados pessoais estão sendo enviados sem necessidade?
- Existe diferença entre o que a tela mostra e o que o navegador recebe?

Sinal de alerta: Se o dado não deveria ser visto pelo usuário, ele não deveria chegar ao navegador. Filtrar na tela não resolve o problema.

O que marcar

* Revisei os campos retornados pelas APIs.
* Removi informações que a tela não precisa.
* Dados internos não são enviados por padrão.
* A resposta do backend é mínima para cada fluxo.

5 Em um SaaS, cliente nenhum deveria enxergar o outro

Se o seu sistema atende mais de uma pessoa, empresa, organização ou workspace, o isolamento entre contas precisa ser tratado com muita atenção.

A regra é simples: cada cliente deve acessar apenas os próprios dados. Isso precisa valer para telas, APIs, relatórios, arquivos, dashboards, convites, pagamentos, histórico de atividades, tudo.

Um erro perigoso

Buscar um registro apenas pelo ID sem confirmar a quem ele pertence. Em muitos sistemas, isso não é suficiente. A consulta também precisa confirmar que aquele registro pertence ao usuário ou à organização correta.

Perguntas para revisar

- Toda consulta sensível confirma a conta ou organização?
- O ID da organização vem de informação confiável do servidor?
- Um usuário consegue alterar filtros e ver dados de outro cliente?
- Convites levam a pessoa para a organização correta?
- Arquivos ficam separados por cliente?

Sinal de alerta: Se a consulta encontra o registro certo mas não confirma quem tem permissão para vê-lo, existe risco real de vazamento entre clientes.

O que marcar

* Testei acesso com dois usuários de contas diferentes.
* Cada conta vê apenas seus próprios dados.
* APIs validam organização ou proprietário.
* Convites e arquivos respeitam o isolamento entre clientes.

6 Se você usa Supabase, não ignore as regras do banco

O Supabase facilita muito a vida de quem está construindo rápido. Autenticação, banco de dados, storage, integração com o frontend, tudo junto.

Mas essa facilidade não elimina a necessidade de regras de acesso. Uma das perguntas mais importantes para quem usa Supabase é: as tabelas sensíveis têm RLS configurado corretamente?

RLS significa Row Level Security. São regras no banco que controlam quais linhas cada usuário pode ler, criar, editar ou apagar.

Por que isso importa

Sem regras bem configuradas, um usuário autenticado pode acabar acessando dados que não pertencem a ele. Mesmo que a tela pareça correta, mesmo que o login funcione, mesmo que o frontend só mostre os dados esperados.

Tabelas que merecem atenção

- perfis de usuários
- clientes e organizações
- pedidos e pagamentos
- documentos e arquivos
- mensagens e assinaturas
- qualquer registro que deveria ser privado

Perguntas para revisar

- RLS está ativa nas tabelas importantes?
- Existem regras para leitura, criação, edição e exclusão?
- A regra confirma o usuário ou organização correta?
- A chave administrativa do Supabase ficou somente no backend?
- O acesso foi testado com contas diferentes?

> **Sinal de alerta:** Se você usa Supabase e não sabe dizer se as tabelas privadas têm RLS ativa, vale parar antes de publicar e conferir.

O que marcar

* Identifiquei as tabelas sensíveis.

* Confirmei que RLS está ativa quando necessária.

* Testei com usuários diferentes.

* A chave administrativa não está no frontend.

7 Painel administrativo não pode depender de segredo na URL

Todo produto tem, em algum momento, uma área administrativa. Ela pode estar em caminhos como /admin, /painel, /backoffice, /gestao ou /internal.

O erro é imaginar que ninguém vai encontrar essa página porque o link não aparece no menu. Endereço escondido não é proteção.

Se existe uma rota administrativa, o sistema precisa verificar quem está tentando acessá-la, e isso vale também para as APIs usadas por essa rota.

O que realmente precisa existir

- autenticação
- verificação de permissão administrativa
- bloqueio no servidor, não só no frontend
- sessão protegida
- cuidados extras para ações críticas

Perguntas para revisar

- Digitar o endereço do admin abre alguma coisa sem autorização?
- Usuário comum consegue acessar a rota?
- Usuário comum consegue chamar endpoints administrativos?
- A proteção existe no servidor ou só no frontend?
- A área admin aparece em lugares públicos sem necessidade?

Sinal de alerta: Se a única defesa do painel é "ninguém sabe o endereço", ele não está protegido.

O que marcar

* Testei a rota admin sem estar logado.

* Testei com usuário comum.

* O servidor bloqueia acesso indevido.

* As APIs administrativas validam permissão.

8 Upload de arquivos merece mais atenção do que parece

Adicionar upload costuma parecer simples: o usuário envia um PDF ou uma imagem e você salva. Mas arquivos envolvem várias decisões que passam despercebidas.

Quem pode enviar, quem pode abrir, onde o arquivo fica salvo, se o link é público, quais tipos são aceitos, qual o tamanho máximo, se o arquivo pode ser substituído, se o nome revela alguma informação.

Exemplos de problema

- documento privado acessível por link público
- usuário baixando arquivo de outro cliente
- bucket inteiro exposto
- upload aceitando qualquer tipo de arquivo
- sem limite de tamanho
- validação feita apenas no navegador
- documento sensível indexado por engano

Perguntas para revisar

- O upload aceita apenas os tipos realmente necessários?
- Existe limite de tamanho?
- Arquivos privados exigem autorização para serem baixados?
- Um usuário consegue alterar uma URL e abrir arquivo de outro?
- O storage está público sem necessidade?
- A validação também ocorre no servidor?

Sinal de alerta: Se o arquivo deveria ser privado mas qualquer pessoa com o link consegue abrir, revise antes de publicar.

O que marcar

* Defini tipos permitidos de arquivo.

* Defini limite de tamanho.

* Arquivos privados exigem acesso autorizado.

* Usuários não acessam arquivos de outras contas.

9 Logs podem guardar justamente o que você queria proteger

Logs ajudam a descobrir por que algo quebrou. O problema é quando eles registram informação demais.

Durante o desenvolvimento, é comum deixar registros de tudo: requisição completa, resposta completa, token, cookie, senha, email, telefone, dados de pagamento, chave de API. Em produção, isso pode virar um vazamento silencioso.

O que revisar com atenção

Olhe os logs de login, cadastro, pagamento, recuperação de senha, uploads, integrações externas, webhooks e chamadas para IA.

Perguntas para revisar

- O sistema salva o corpo inteiro de requisições?
- Tokens aparecem em logs?
- Erros registram dados pessoais?
- Quem consegue acessar os logs?
- Informações sensíveis são mascaradas?
- Logs antigos ficam guardados por quanto tempo?

Sinal de alerta: Se uma busca simples nos logs encontra senha, token, cookie ou dados pessoais completos, existe um problema que precisa ser resolvido antes de ir a produção.

O que marcar

* Revisei logs dos fluxos importantes.
* Tokens e senhas não são gravados.
* Dados pessoais são evitados ou mascarados.
* Acesso aos logs é restrito.

10 Mensagem de erro não precisa entregar os bastidores

Quando algo falha, o usuário precisa saber que houve um problema. Mas ele não precisa receber os detalhes internos do sistema.

Erros mal tratados podem revelar caminho de arquivos, versão de bibliotecas, nome de tabela, query do banco, estrutura da API, informações do servidor, stack trace ou variáveis de ambiente.

Um exemplo simples

Em vez de exibir uma mensagem técnica como "Falha na query da tabela users ao validar JWT em /srv/app/auth.ts", a tela poderia simplesmente mostrar: "Não foi possível concluir esta ação. Tente novamente."

Os detalhes técnicos ficam registrados internamente, em local seguro, para análise.

Atenção especial ao login

Mensagens como "Este email existe, mas a senha está errada" podem ajudar alguém a descobrir quais usuários estão cadastrados. Em muitos casos, é melhor usar uma mensagem genérica como "Email ou senha inválidos".

Perguntas para revisar

- Stack trace aparece para o usuário?
- O sistema revela nomes internos de tabela ou arquivos?
- Erros de login informam demais?
- Chaves ou tokens aparecem em mensagens de erro?
- As APIs retornam detalhes técnicos em produção?

Sinal de alerta: Se o erro ajuda alguém a entender a estrutura interna do sistema, ele está mostrando mais do que deveria.

O que marcar

* Erros públicos são claros, mas não revelam detalhes internos.

* Stack traces não aparecem em produção.

* Login não confirma desnecessariamente quais usuários existem.

* Detalhes técnicos ficam apenas em logs protegidos.

11 CORS aberto demais pode ser um atalho perigoso

CORS costuma aparecer quando o frontend tenta chamar uma API e o navegador bloqueia. É comum a IA resolver rapidamente adicionando uma configuração ampla, como permitir qualquer origem.

Em alguns casos, isso pode ser aceitável. Em outros, especialmente quando existem sessão, cookies, dados privados ou ações sensíveis, precisa ser revisado com cuidado.

Às vezes a configuração foi feita apenas para parar de dar erro, sem pensar em quais sites deveriam poder conversar com a API.

Perguntas para revisar

- Quais domínios estão autorizados a chamar a API?
- O ambiente de produção ainda permite localhost?
- Existe permissão para qualquer origem sem necessidade?
- A API trabalha com cookies ou credenciais?
- Configurações de desenvolvimento foram levadas para produção?

Uma observação importante: CORS não substitui autenticação e autorização. Mesmo com CORS bem configurado, o servidor ainda precisa verificar quem está fazendo a ação e quais dados essa pessoa pode acessar.

> **Sinal de alerta:** Se a solução encontrada foi simplesmente liberar tudo em produção, vale parar e revisar antes de receber usuários reais.

O que marcar

* Revisei origens permitidas em produção.
* localhost não ficou liberado sem necessidade.
* APIs sensíveis não estão abertas de forma genérica.
* Autenticação e autorização continuam sendo validadas no servidor.

12 Faça uma revisão de 30 minutos antes de publicar

Você não precisa começar com uma auditoria enorme para evitar erros óbvios. Antes de colocar uma funcionalidade importante no ar, separe 30 minutos e responda:

- O que essa tela mostra?
- O que a API entrega ao navegador?
- Quem pode abrir essa página?
- Quem pode chamar essa ação?
- Um usuário consegue acessar dados de outro?
- Existe alguma chave exposta?
- O painel admin está realmente bloqueado?
- Arquivos privados estão realmente privados?
- Logs guardam informações sensíveis?
- Erros revelam detalhes internos?
- Configurações de teste ficaram em produção?
- Eu colocaria dados reais de usuários aqui hoje?

Se alguma resposta for incerta, anote e revise antes de publicar.

Não é sobre travar o lançamento. É sobre não descobrir depois, da pior forma, algo que estava visível antes.

O que marcar

* Reservei tempo para revisar antes de publicar.
* Testei com mais de um usuário.
* Revisei APIs e dados enviados ao navegador.
* Confirmei que não existem secrets expostos.
* Só publicarei dados reais depois de revisar os pontos críticos.

Checklist final de publicação

Use esta lista como revisão rápida antes de colocar usuários reais dentro do sistema.

Navegador e frontend

O que marcar
* Abri o DevTools e revisei Network, Application e Console.
* As APIs retornam somente os dados necessários.
* Não existem secrets ou tokens privados no frontend.
* localStorage e sessionStorage não guardam dados sensíveis sem necessidade.
* Nenhum erro técnico importante aparece para usuários.

Login e permissões

O que marcar
* Rotas protegidas validam sessão no servidor.
* Cada API confirma se o usuário pode acessar aquele dado.
* Trocar IDs na URL não permite ver dados de outra pessoa.
* Usuário comum não acessa funções administrativas.
* A proteção não depende apenas de esconder botões.

Clientes, organizações e banco de dados

O que marcar
* Uma conta não acessa dados de outra conta.
* Consultas ao banco filtram corretamente usuário ou organização.
* Convites levam o usuário para a organização correta.
* Arquivos e relatórios respeitam o isolamento entre clientes.
* Se uso Supabase, revisei RLS nas tabelas sensíveis.

Upload, logs e erros

O que marcar
* Upload aceita apenas tipos realmente necessários. * Existe limite de tamanho para arquivos. * Arquivos privados exigem autorização. * Logs não armazenam senha, token, cookie ou dados sensíveis completos. * Mensagens de erro não revelam detalhes internos.

Produção e exposição

O que marcar
* CORS foi revisado para produção. * localhost e configurações de teste não ficaram liberados sem necessidade. * Painel administrativo exige permissão real. * Variáveis de ambiente foram conferidas antes do deploy. * Fiz uma revisão final antes de aceitar usuários reais.

5 prompts para revisar seu sistema com IA

A IA pode ter ajudado a construir seu sistema. Ela também pode ajudar a revisar decisões importantes, desde que você peça da forma certa.

A seguir estão cinco prompts que você pode copiar e colar na ferramenta que está usando. A recomendação é simples: peça primeiro o diagnóstico. Só depois autorize mudanças no código.

Prompt 1: Descobrir dados expostos no frontend

```
Revise este fluxo como um especialista em segurança de aplicações SaaS.

Quero identificar se o frontend está recebendo dados além do necessário
ou armazenando informações sensíveis no navegador.

Analise:
- respostas das APIs
- campos enviados ao navegador
- dados salvos em localStorage, sessionStorage e cookies
- tokens ou identificadores presentes no frontend
- objetos retornados pelo backend
- erros exibidos no Console

Para cada problema encontrado, me diga:
1. o que está exposto
2. onde isso acontece
3. por que isso representa risco
4. qual seria o impacto em um sistema real
5. como corrigir retornando apenas o necessário
6. como testar manualmente se a correção funcionou

Não altere o código ainda.
Primeiro me entregue apenas o diagnóstico, priorizado por gravidade.
```

Prompt 2: Revisar permissões e acesso entre usuários

```
Revise este sistema procurando falhas de autorização e acesso indevido
entre usuários, clientes, organizações ou workspaces.

Não assuma que login funcionando significa que o sistema está protegido.

Analise:
- rotas protegidas
- endpoints de API
- parâmetros de URL
- IDs enviados pelo frontend
- consultas ao banco
- funções administrativas
- páginas que exibem dados privados

Quero descobrir se um usuário autenticado consegue acessar ou alterar
dados de outro usuário ou organização.

Para cada risco encontrado, me diga:
1. onde está a falha
2. qual cenário de abuso seria possível
3. qual validação está faltando no servidor
4. qual correção recomenda
5. como testar usando dois usuários diferentes

Não faça mudanças automáticas ainda. Primeiro entregue o diagnóstico.
```

Prompt 3: Procurar secrets e chaves expostas

```
Revise este projeto procurando exposição de secrets, tokens,
chaves privadas e variáveis sensíveis.

Analise:
- arquivos .env
- variáveis com prefixo público (NEXT_PUBLIC_, VITE_)
- código frontend
- chamadas para APIs externas
- tokens em logs
- credenciais em arquivos
- service role keys
- chaves presentes no histórico ou em exemplos de configuração

Classifique cada item como: público, sensível ou crítico.

Para cada item sensível ou crítico, explique:
1. onde ele está
2. por que não deveria estar ali
3. que ação alguém poderia executar com acesso a esse segredo
4. para onde esse segredo deveria ser movido
5. como corrigir
6. como validar que ele não aparece mais no bundle ou no DevTools

Não altere o código ainda. Primeiro me entregue o diagnóstico.
```

Prompt 4: Revisar Supabase e RLS

```
Revise a segurança do Supabase utilizado neste projeto.

Quero confirmar se tabelas sensíveis têm Row Level Security
configurada corretamente e se cada usuário acessa apenas os dados permitidos.

Analise:
- tabelas com dados de usuários
- tabelas com dados de organizações, clientes ou workspaces
- tabelas com pagamentos, arquivos, mensagens ou documentos
- políticas de select, insert, update e delete
- uso de auth.uid()
- campos user_id, owner_id, organization_id ou tenant_id
- chamadas feitas diretamente pelo frontend
- uso da service role key
- storage e arquivos privados

Para cada tabela sensível, me entregue:
1. se RLS está ativa
2. quais operações têm políticas
3. quem pode ler, criar, editar e excluir
4. qual risco existe atualmente
5. qual política deveria ser adotada
6. como testar usando dois usuários diferentes

Não aplique mudanças automaticamente. Primeiro entregue o diagnóstico.
```

Prompt 5: Revisão final antes do deploy

```
Faça uma revisão de segurança antes do deploy deste sistema.

O objetivo é encontrar riscos evidentes antes de colocar usuários reais
dentro do produto.

Verifique:
- autenticação e autorização
- dados expostos no frontend
- secrets e variáveis de ambiente
- painel administrativo
- uploads, logs e mensagens de erro
- CORS e banco de dados
- isolamento entre usuários ou organizações
- integrações externas
- configurações de produção
- arquivos e documentos privados

Me entregue uma tabela com:
1. risco encontrado
2. onde está
3. impacto provável
4. gravidade: baixa, média, alta ou crítica
5. correção recomendada
6. como testar manualmente
7. se deve ser resolvido antes do deploy

Não faça mudanças automáticas. Primeiro o diagnóstico priorizado.
```

Uma última orientação

Você não precisa usar todos esses prompts ao mesmo tempo.

Comece pelo que mais importa no seu sistema agora.

- Se você tem login e área de cliente, revise permissões.
- Se você tem dados pessoais, revise o que chega ao navegador.
- Se você tem Supabase, revise RLS.
- Se você tem upload, revise arquivos privados.
- Se você integrou pagamento, email ou IA, revise secrets.
- Se você já vai publicar, rode a revisão final antes do deploy.

A ideia é simples: evitar que a pressa faça você publicar algo importante sem olhar para os pontos certos.

Quem preparou este material

Meu nome é Eduardo Monteiro e eu criei a Mosai Security.

Trabalho com tecnologia há mais de 20 anos e acompanhei de perto sistemas sendo construídos, publicados, mantidos e corrigidos depois que algum problema já tinha aparecido.

Com a chegada das ferramentas de IA, a velocidade mudou completamente. Hoje, uma pessoa consegue tirar uma ideia do papel, criar telas, integrar serviços, publicar e testar com usuários em poucos dias. Eu acho isso fantástico.

Mas essa velocidade também trouxe um problema: sistemas reais estão sendo publicados sem que seus criadores tenham clareza sobre algumas decisões invisíveis.

- O usuário consegue ver dados que não deveria?
- A autenticação realmente separa uma conta da outra?
- Alguma chave privada foi parar no navegador?
- O painel administrativo está protegido de verdade?
- O upload é privado quando deveria ser?
- Os logs estão guardando informação demais?

Essas dúvidas não significam que criar com IA é errado. Significam que construir rápido também exige revisar melhor.

Preparei este checklist porque acredito que segurança não deveria aparecer apenas quando algo dá errado. Ela pode fazer parte do processo de um jeito simples e possível para quem está criando.

Se este checklist te fizer remover uma chave do frontend, revisar uma permissão, testar o acesso entre dois usuários ou perceber um dado exposto antes de publicar, ele já cumpriu o papel dele.

Eduardo Monteiro

Fundador da Mosai Security | mosai.com.br

Quer ver como seu sistema está de verdade?

Este checklist te ajuda a saber onde olhar. O Mosai Scanner faz a varredura por você.

São 78 verificações automáticas que analisam seu sistema em minutos e entregam um relatório com os problemas encontrados, o nível de gravidade de cada um e o que fazer para corrigir.

O que o scanner verifica:

- Headers de segurança HTTP
- Exposição de informações sensíveis
- Configurações de cookies e sessão
- Proteção contra ataques comuns
- CORS e políticas de acesso
- Certificado SSL e configurações TLS

Acesse agora:

scan.mosai.com.br