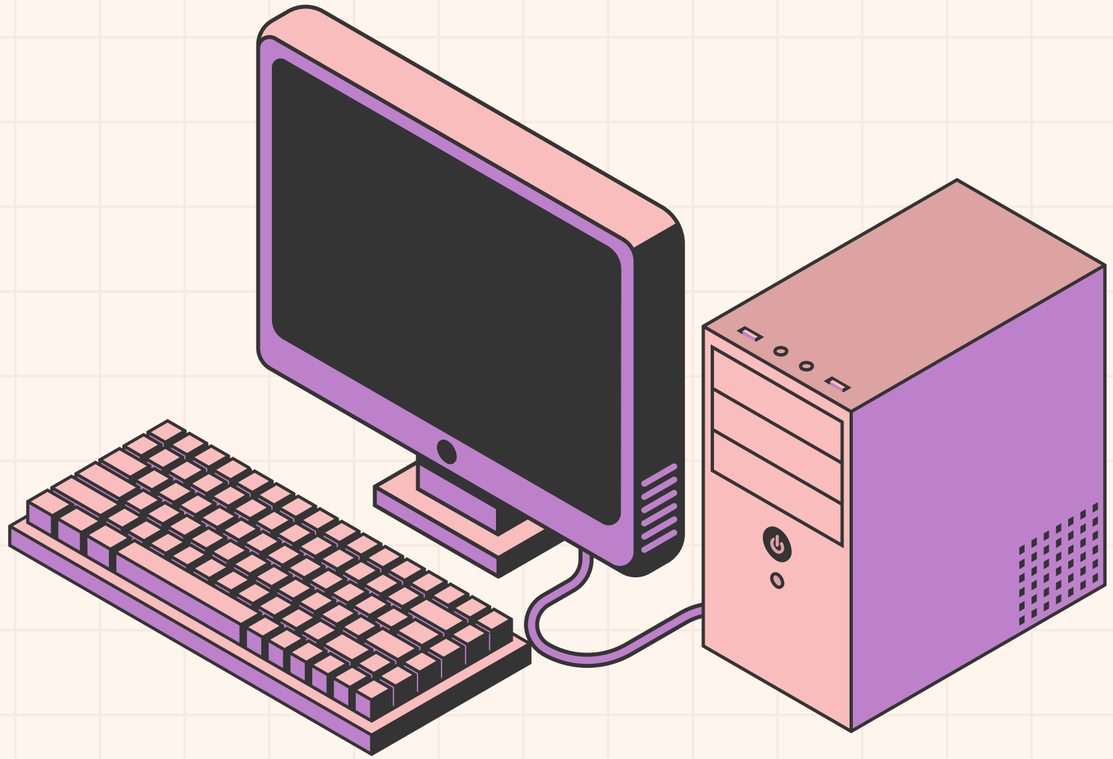
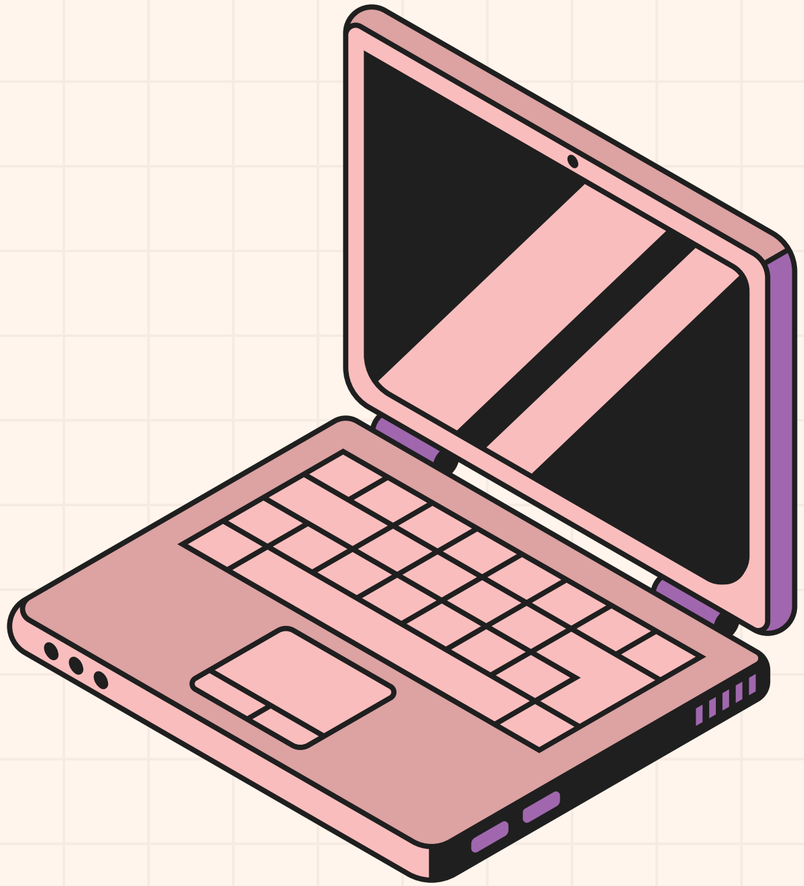
INSECURE DESIGN

(OWASP A04:2021)



O QUE É INSECURE DESIGN

- **Insecure Design**: falhas de arquitetura — segurança não prevista nos fluxos.
- **Bug de implementação**: erro de código (ex: esquecer validação).
- **Segurança** deve nascer nos requisitos e modelagem, não ser “remendo” depois.

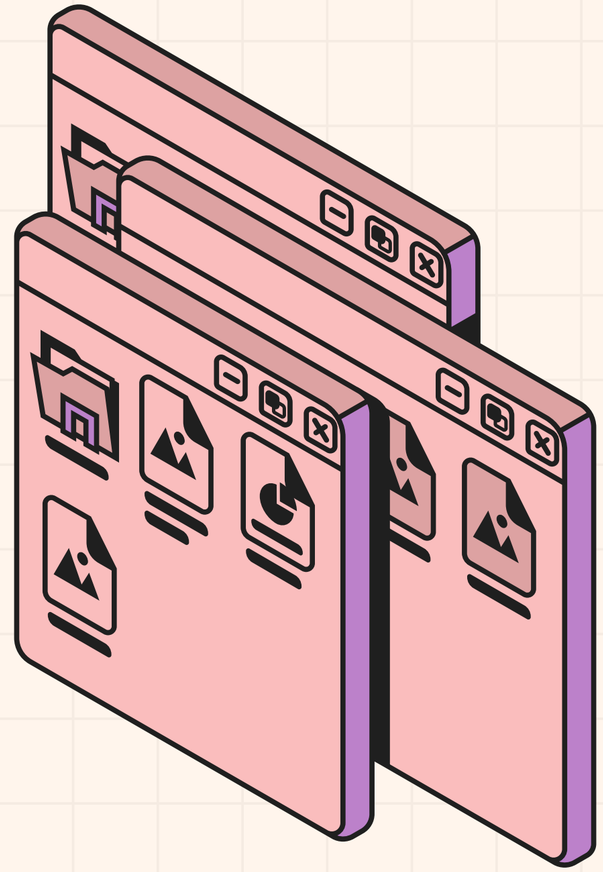


COMO ACONTECE ?

- **Requisitos sem segurança:** histórias de usuário não exigem proteção.
- **Threat Modeling** (mapear ameaças antes de codar) não é feito.
- **Pressa por features** → MVP eterno (sistema provisório que vira definitivo).
- **Sem padrões de projeto**:
  - **Least Privilege**: dar ao usuário só o mínimo de permissão necessária.
  - **Deny-by-default**: bloquear tudo por padrão e liberar só o necessário.
- **Complexidade sem fronteiras** → serviços expostos demais.

SINAIS DE ALERTA

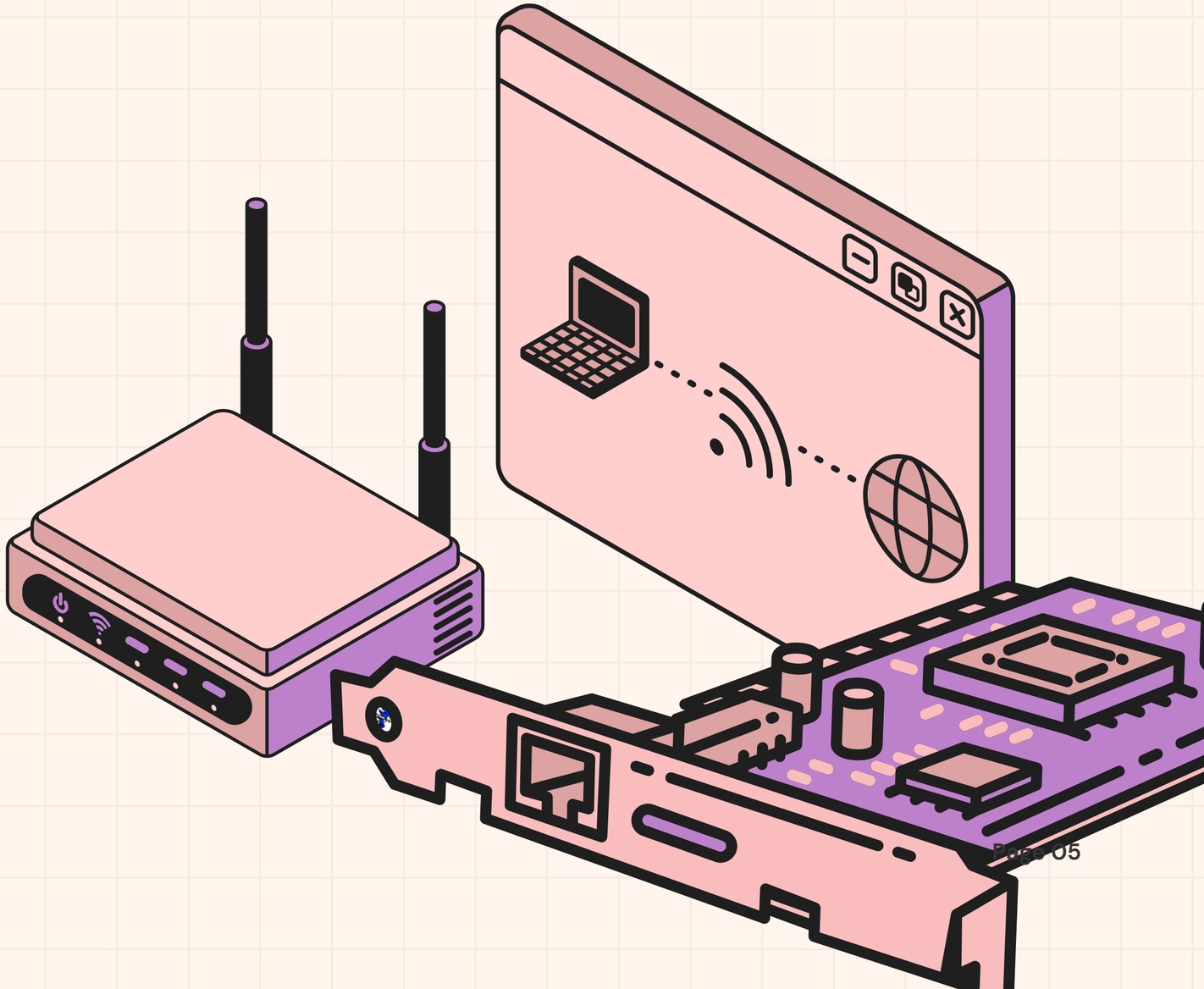
- **Sem Rate Limiting**: sistema não limita tentativas por minuto/IP.
- **Autorização só no front-end** (fácil de burlar).
- **IDOR** (Insecure Direct Object Reference): usar IDs previsíveis para acessar dados de outros.
- **Fluxos críticos** (reset de senha, transferências) sem dupla confirmação.
- **Regras de negócio** não documentadas nem testadas.



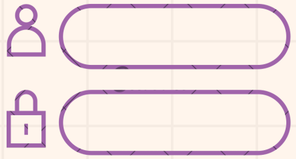
HOME SERVICE ABOUT US CONTACT US

EXEMPLOS PRÁTICOS

- **Login sem rate limiting** → ataque de força bruta.
- **IDOR** em ***/users/{id}*** → usuário A acessa dados do B.
- **Reset de senha frágil** → token fraco ou infinito.
- **Pagamento sem antifraude** → abuso de reembolsos.



EXEMPLOS PRÁTICOS



LOGIN SEM RATE LIMITING

Inseguro: ilimitadas tentativas, erro detalha email errado.

Seguro: 5 tentativas por minuto, atraso progressivo, erro genérico, opcional MFA (autenticação de dois fatores).

IDOR

Inseguro: qualquer logado acessa /users/{id}.

Seguro: o back-end valida se o usuário é dono do recurso ou admin.

RESET DE SENHA

Inseguro: token = email + timestamp, nunca expira, pode ser reutilizado.

Seguro: token aleatório de 32 bytes, expira em 15 min, uso único, invalida anteriores, envia notificação ao usuário.

EXEMPLOS EM CÓDIGO

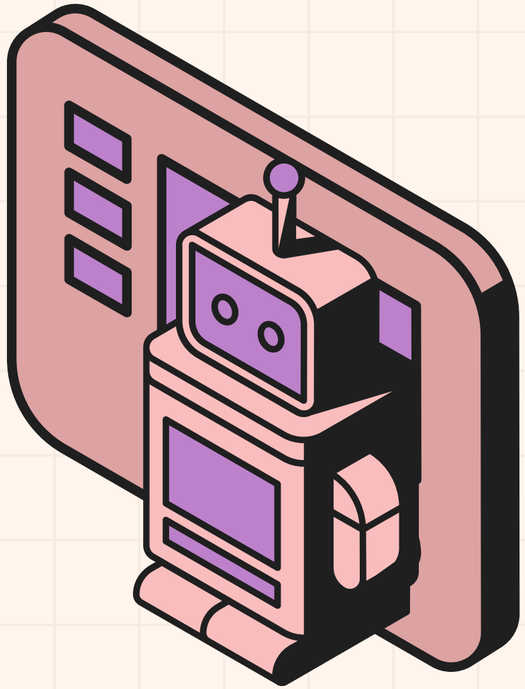
ERRADO

```
    // Simulação de endpoint para redefinir senha
public class PasswordReset {
    public static void resetPassword(String email, String newPassword) {
        // Basta informar o e-mail e pronto: troca a senha
        System.out.println("Senha do usuário " + email + " alterada para: " + newPassword);
    }

    Run | Debug
    public static void main(String[] args) {
        // Hacker só precisa do e-mail da vítima
        resetPassword(email:"victima@exemplo.com", newPassword:"123456");
    }
}
Ctrl+L to chat, Ctrl+K to generate
```

CERTO

```
import java.util.Scanner;
import java.util.Random;

public class PasswordResetSecure {
    private static String generatedToken;

    // Envia token por e-mail (simulado aqui)
    public static void sendResetToken(String email) {
        generatedToken = String.valueOf(new Random().nextInt(999999));
        Run | Debug
        System.out.println("Token enviado para " + email + ": " + generatedToken);
    }

    // Reseta senha somente com token válido
    public static void resetPassword(String email, String token, String newPassword) {
        if (token.equals(generatedToken)) {
            System.out.println("Senha do usuário " + email + " alterada com sucesso!");
        } else {
            System.out.println("Token inválido. Redefinição de senha negada.");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String email = "victima@exemplo.com";
        sendResetToken(email);

        System.out.print("Digite o token recebido: ");
        String token = sc.nextLine();

        resetPassword(email, token, newPassword:"NovaSenhaForte123!");
        sc.close();
    }
```

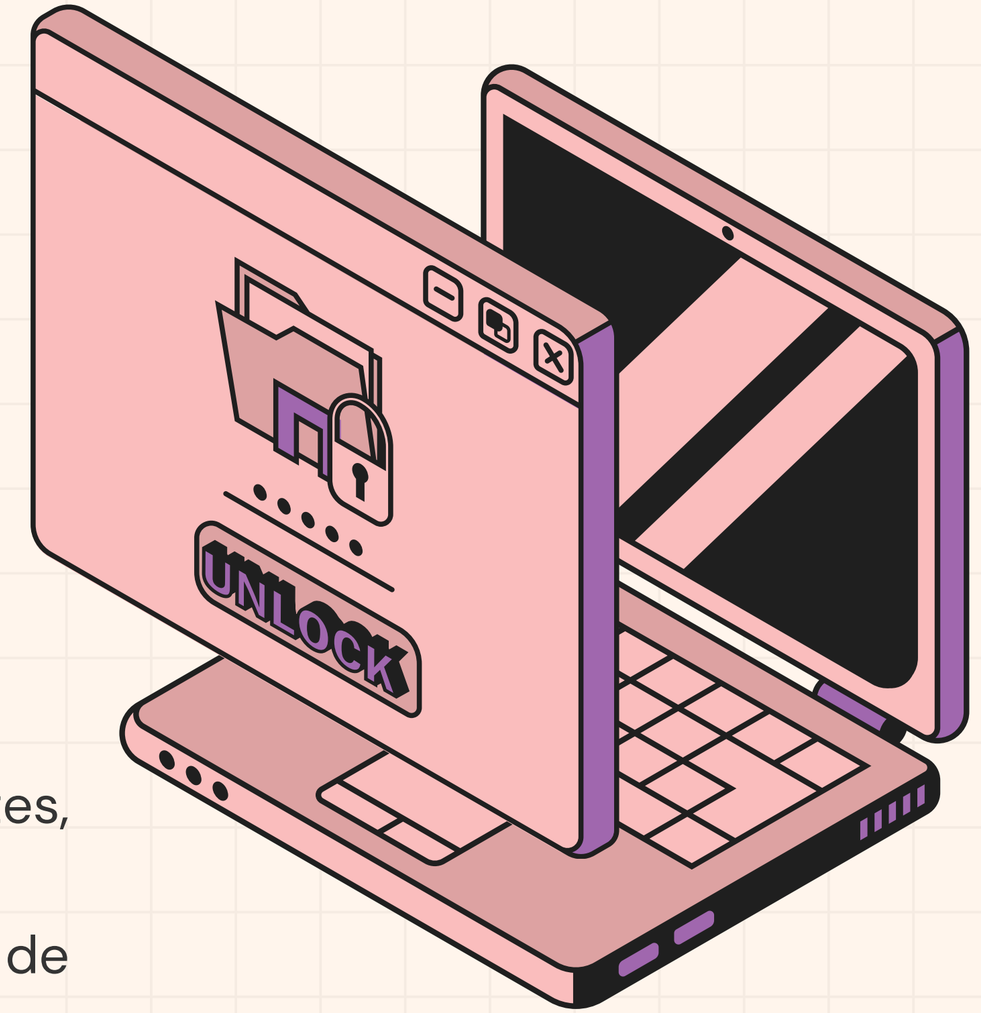
COMO LIDAR
(CHECKLIST DE DESIGN SEGURO)



- [x] **Fazer Threat Modeling** → pensar em ataques antes do código.
- [x] **Criar critérios de aceite** de segurança em cada história.
- [x] **Autorização** no **back-end**, least privilege e deny-by-default.
- [x] **Autenticação forte**: MFA, sessões curtas, logout forçado quando necessário.
- [ ] **Rate limiting** em logins, resets e APIs pesadas.
- [ ] **Tokens fortes**, com expiração e rotação periódica.
- [ ] **Criptografia em trânsito** (HTTPS) e em repouso (DB).
- [ ] **Auditoria de eventos críticos** (reset, pagamento, acesso a dados).

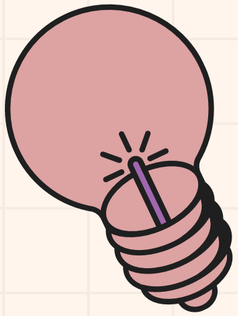
TESTES
E OBSERVABILIDADE

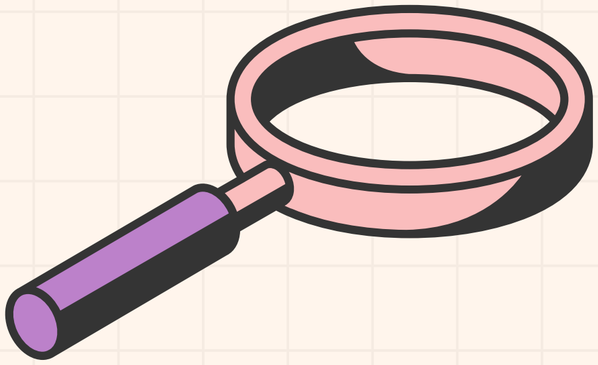
- **Critérios de aceite**:
  - Após 5 falhas → atraso ≥ 2s.
  - Usuário não acessa dados de outro.
  - Token expira em 15 min, uso único.
- **Testes automatizados**: autorização, limites, misuse cases (ex: abusar de reembolso).
- **Observabilidade**: métricas de login, alertas de padrão suspeito, dashboard de erros 403/429.



CONCLUSÃO

Projeto seguro é estrutura, não detalhe extra. Desde o levantamento de requisitos siga princípios sólidos: **least privilege, deny-by-default, tokens fortes e limites**. Essas boas práticas precisam ser adotadas desde o início do projeto para evitar problemas maiores no futuro.





OBRIGADO

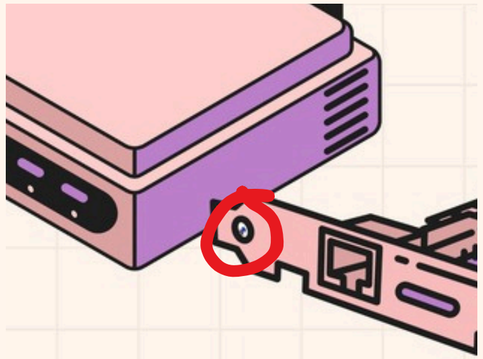
REFERÊNCIAS

OWASP TOP 10: PRINCIPAIS RISCOS DE SEGURANÇA EM APLICAÇÕES.

OWASP ASVS: LISTA DE REQUISITOS PARA VERIFICAR SEGURANÇA DE APPS

OWASP SAMM: MODELO DE MATURIDADE PARA SEGURANÇA NO DESENVOLVIMENTO.

CONSEGUIU
ENCONTRAR ?



e 02