Tutorial Completo: Sistema de Autenticação e Autorização

Objetivo: Compreender conceitos e implementação do sistema de autenticação/autorização usado nos projetos integradores.

Índice

---

1. Fundamentos Teóricos

1.1 Autenticação vs Autorização

Autenticação é o processo de verificar **quem** você é:

- "Você é realmente João Silva?"
- Exemplo: Login com email e senha
- Responde: "Este usuário é válido?"

Autorização é o processo de verificar **o que** você pode fazer:

- "João Silva tem permissão para acessar a área administrativa?"
- Exemplo: Verificar se o usuário tem perfil de ADMIN
- Responde: "Este usuário pode acessar este recurso?"

```
┌─────────────────────────────────────────────────┐
│ FLUXO TÍPICO DE AUTENTICAÇÃO E AUTORIZAÇÃO      │
├─────────────────────────────────────────────────┤
│                                                 │
│ 1. Usuário fornece credenciais (email + senha)  │
│                 ↓                               │
│ 2. Sistema verifica se as credenciais são válidas │
│    [AUTENTICAÇÃO]                               │
│                 ↓                               │
│ 3. Se válido, usuário é autenticado             │
│    Sistema cria uma SESSÃO                      │
│                 ↓                               │
│ 4. Usuário tenta acessar um recurso protegido   │
│                 ↓                               │
│ 5. Sistema verifica se o usuário tem permissão  │
│    [AUTORIZAÇÃO]                                │
│                 ↓                               │
│ 6. Acesso concedido ou negado                   │
│                                                 │
└─────────────────────────────────────────────────┘
```

1.2 Hashing de Senhas (bcrypt)

Por que não armazenar senhas em texto plano?

Se um atacante obtiver acesso ao banco de dados:

- ❌ **Texto plano**: `senha: "123456"` → Todas as contas comprometidas imediatamente
- ✅ **Hash**: `senha: "$2b$12$KIX..."` → Senhas não podem ser recuperadas

O que é hashing?

Hashing é uma função matemática de **mão única**:

- Entrada: `"minha_senha"`
- Saída: `"$2b$12$KIXmPGCDHx3uR7..."` (hash)
- **Impossível** reverter: do hash para a senha original

Características importantes:

1. **Determinístico**: mesma senha sempre gera o mesmo hash

2. **Irreversível**: não há como voltar do hash para a senha
3. **Resistente a colisões**: senhas diferentes geram hashes diferentes

bcrypt - O algoritmo escolhido

O bcrypt é especialmente bom para senhas porque:

- **Lento por design**: dificulta ataques de força bruta
- **Salt automático**: cada hash é único mesmo para senhas iguais
- **Ajustável**: pode aumentar a complexidade com o tempo

```
# Exemplo de uso do bcrypt
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# Criar hash de uma senha
senha = "minha_senha_secreta"
hash_senha = pwd_context.hash(senha)
# Resultado: "$2b$12$KIXmPGCDHx3uR7..."

# Verificar se uma senha corresponde ao hash
senha_digitada = "minha_senha_secreta"
eh_valida = pwd_context.verify(senha_digitada, hash_senha)
# Resultado: True

senha_errada = "senha_incorreta"
eh_valida = pwd_context.verify(senha_errada, hash_senha)
# Resultado: False
```

O que é o Salt?

Salt é um valor aleatório adicionado à senha antes do hash:

```
Sem salt:
  hash("senha123") → sempre o mesmo resultado

Com salt (bcrypt):
  hash("senha123" + "xK9mP2") → resultado único
  hash("senha123" + "aB7cD1") → resultado diferente
```

Isso significa que dois usuários com a mesma senha terão hashes diferentes no banco!

1.3 Sessões HTTP

O Problema: HTTP é Stateless

HTTP não mantém estado entre requisições:

```
Requisição 1: Login com sucesso ✅
Requisição 2: Acessar perfil → Quem é você? ❓
```

A Solução: Sessões

Sessões permitem que o servidor "lembre" do usuário entre requisições:

```
┌─────────────────────────────────────────────────┐
│ FUNCIONAMENTO DE SESSÕES                        │
├─────────────────────────────────────────────────┤
│                                                 │
│ 1. Usuário faz login com sucesso                │
│                 ↓                               │
│ 2. Servidor cria uma SESSÃO no servidor         │
│    session_id: "abc123"                         │
│    dados: {usuario_id: 42, nome: "João", ...}   │
│                 ↓                               │
│ 3. Servidor envia COOKIE para o navegador       │
│    Set-Cookie: session=abc123                   │
│                 ↓                               │
│ 4. Navegador armazena o cookie                  │
│                 ↓                               │
│ 5. Próximas requisições incluem o cookie        │
│    Cookie: session=abc123                       │
│                 ↓                               │
│ 6. Servidor reconhece o usuário pela sessão     │
│                                                 │
└─────────────────────────────────────────────────┘
```

Componentes de uma Sessão:

1. **Session ID**: identificador único da sessão (armazenado no cookie)
2. **Dados da sessão**: informações do usuário (armazenados no servidor)
3. **Expiração**: tempo de validade da sessão

Exemplo prático:

```
 # No servidor (FastAPI + SessionMiddleware)
from starlette.middleware.sessions import SessionMiddleware

app.add_middleware(
    SessionMiddleware,
    secret_key="chave-secreta-muito-segura",
    max_age=3600,  # Sessão expira em 1 hora
)

# Em uma rota de login
@app.post("/login")
async def login(request: Request, email: str, senha: str):
    usuario = verificar_credenciais(email, senha)
    if usuario:
        # Criar sessão
        request.session['usuario'] = {
            'id': usuario.id,
            'nome': usuario.nome,
            'perfil': 'ADMIN'
        }
        return {"mensagem": "Login bem-sucedido"}

# Em outra rota
@app.get("/perfil")
async def perfil(request: Request):
    # Recuperar dados da sessão
    usuario = request.session.get('usuario')
    if usuario:
        return {"nome": usuario['nome']}
    else:
        return {"erro": "Não autenticado"}
```

1.4 Tokens de Segurança

Tokens são strings aleatórias usadas para operações temporárias e sensíveis.

Uso comum: Recuperação de Senha

```
┌─────────────────────────────────────────────────────┐
│ FLUXO DE RECUPERAÇÃO DE SENHA COM TOKEN             │
├─────────────────────────────────────────────────────┤
│                                                     │
│ 1. Usuário esquece a senha                          │
│    Solicita recuperação                             │
│                 ↓                                   │
│ 2. Sistema gera TOKEN aleatório                     │
│    token: "xK9mP2aB7cD1..."                         │
│    expira_em: 24 horas                              │
│                 ↓                                   │
│ 3. Sistema salva token no banco                     │
│    usuario.token_redefinicao = "xK9mP2..."          │
│                 ↓                                   │
│ 4. Sistema envia email com link                     │
│    Link: /reset-senha?token=xK9mP2...               │
│                 ↓                                   │
│ 5. Usuário clica no link                            │
│                 ↓                                   │
│ 6. Sistema valida o token:                          │
│    - Token existe no banco?                         │
│    - Token não expirou?                             │
│                 ↓                                   │
│ 7. Se válido, permite definir nova senha            │
│                 ↓                                   │
│ 8. Token é invalidado após uso                      │
│                                                     │
└─────────────────────────────────────────────────────┘
```

Geração de Token Seguro:

```
import secrets
import string

def gerar_token(tamanho: int = 32) -> str:
    """Gera um token aleatório criptograficamente seguro"""
    caracteres = string.ascii_letters + string.digits
    return ''.join(secrets.choice(caracteres) for _ in range(tamanho))

# Resultado: "xK9mP2aB7cD1fG3hI5jK8lM0nO4pQ6rS"
```

Características de um bom token:

- **Aleatório**: usa `secrets` (não `random`!)
- **Longo**: pelo menos 32 caracteres
- **Único**: cada operação gera um novo token
- **Temporário**: expira após um período definido
- **Uso único**: invalidado após ser usado

1.5 Perfis/Roles de Usuário

Perfis (ou roles) definem o que cada tipo de usuário pode fazer no sistema.

No CaseBem existem 3 perfis:

```
class TipoUsuario(Enum):
    ADMIN = "ADMIN"           # Administrador do sistema
    NOIVO = "NOIVO"           # Cliente (casal de noivos)
    FORNECEDOR = "FORNECEDOR" # Fornecedor de produtos/serviços
```

Permissões típicas:

Ação	ADMIN	NOIVO	FORNECEDOR
Ver lista pública de			

Ação	ADMIN	NOIVO	FORNECEDOR
itens	✅	✅	✅
Cadastrar item	✅	✅	✅
Ver todos os usuários	✅	✅	✅
Criar administrador	✅	✅	✅
Criar orçamento	✅	✅	✅
Aprovar fornecedor	✅	✅	✅

Implementação com Enum:

```
 # Definição do modelo
from enum import Enum

class TipoUsuario(Enum):
    ADMIN = "ADMIN"
    NOIVO = "NOIVO"
    FORNECEDOR = "FORNECEDOR"

@dataclass
class Usuario:
    id: int
    nome: str
    email: str
    senha: str
    perfil: TipoUsuario  # ← Tipo é o Enum

# Uso
usuario = Usuario(
    id=1,
    nome="João",
    email="joao@email.com",
    senha="hash...",
    perfil=TipoUsuario.ADMIN  # ← Valor do enum
)

# Verificação
if usuario.perfil == TipoUsuario.ADMIN:
    print("É administrador")

# Converter para string
perfil_string = usuario.perfil.value  # "ADMIN"
```

1.6 Decorators Python

Decorators são funções que modificam o comportamento de outras funções.

Exemplo básico:

```
def meu_decorator(funcao):
    def wrapper():
        print("Antes da função")
        funcao()
        print("Depois da função")
    return wrapper

@meu_decorator
def ola():
    print("Olá!")

ola()
# Saída:
# Antes da função
# Olá!
# Depois da função
```

Por que usar decorators para autenticação?

Sem decorator (repetitivo):

```
@app.get("/admin/dashboard")
async def dashboard(request: Request):
    # Repetir em cada rota protegida
    usuario = request.session.get('usuario')
    if not usuario:
        return RedirectResponse('/login')
    if usuario['perfil'] != 'ADMIN':
        raise HTTPException(403, "Sem permissão")

    # Lógica da rota
    return {"data": "dashboard"}

@app.get("/admin/usuarios")
async def usuarios(request: Request):
    # Repetir tudo de novo...
    usuario = request.session.get('usuario')
    if not usuario:
        return RedirectResponse('/login')
    if usuario['perfil'] != 'ADMIN':
        raise HTTPException(403, "Sem permissão")

    # Lógica da rota
    return {"data": "usuarios"}
```

Com decorator (elegante):

```
@app.get("/admin/dashboard")
@requer_autenticacao(['ADMIN'])
async def dashboard(request: Request, usuario_logado: dict):
    # Lógica da rota diretamente
    return {"data": "dashboard"}

@app.get("/admin/usuarios")
@requer_autenticacao(['ADMIN'])
async def usuarios(request: Request, usuario_logado: dict):
    # Lógica da rota diretamente
    return {"data": "usuarios"}
```

Vantagens:

- ✅ Código limpo e legível
- ✅ Reutilização de lógica
- ✅ Fácil manutenção
- ✅ Menos erros

---

2. Fluxos Completos

2.1 Fluxo de Cadastro de Usuário

```
┌─────────────────────────────────────────────────────────────┐
│ FLUXO DE CADASTRO                                           │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│ 1. Usuário acessa /cadastro-fornecedor                      │
│                 ↓                                           │
│ 2. Preenche formulário:                                     │
│    - Nome                                                   │
│    - Email                                                  │
│    - Senha                                                  │
│    - CPF (opcional)                                         │
│    - Telefone                                               │
│                 ↓                                           │
│ 3. Submit do formulário → POST /cadastro-fornecedor         │
│                 ↓                                           │
│ 4. Backend valida dados:                                    │
│    ✓ Email já existe?                                       │
│    ✓ Senha forte o suficiente?                              │
│    ✓ CPF válido?                                            │
│    ✓ Campos obrigatórios preenchidos?                       │
│                 ↓                                           │
│ 5. Se validação falhar:                                     │
│    → Retorna formulário com mensagem de erro                │
│                                                             │
│ 6. Se validação passar:                                     │
│    a) Criar hash da senha                                   │
│       senha_hash = criar_hash_senha(senha)                  │
│                 ↓                                           │
│    b) Criar objeto Fornecedor                               │
│       fornecedor = Fornecedor(                              │
│         nome=nome,                                          │
│         email=email,                                        │
│         senha=senha_hash,  ← Hash, não texto plano!         │
│         perfil=TipoUsuario.FORNECEDOR                       │
│       )                                                     │
│                 ↓                                           │
│    c) Salvar no banco de dados                              │
│       fornecedor_id = fornecedor_repo.inserir(fornecedor)   │
│                 ↓                                           │
│    d) Enviar email de boas-vindas (opcional)                │
│       enviar_email_boas_vindas(email, nome)                 │
│                 ↓                                           │
│ 7. Redirecionar para /login com mensagem de sucesso         │
│    "Cadastro realizado! Faça login para continuar"          │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Código resumido:

```
@router.post("/cadastro-fornecedor")
async def post_cadastro_fornecedor(
    request: Request,
    nome: str = Form(...),
    email: str = Form(...),
    senha: str = Form(...),
    cpf: str = Form(None),
    telefone: str = Form(...),
):
    # 1. Validar dados
    if usuario_repo.obter_usuario_por_email(email):
        return {"erro": "E-mail já cadastrado"}

    valida, erro = validar_forca_senha(senha)
    if not valida:
        return {"erro": erro}

    # 2. Criar hash da senha
    senha_hash = criar_hash_senha(senha)

    # 3. Criar objeto usuário
    fornecedor = Fornecedor(
        id=0,
        nome=nome,
        email=email,
        senha=senha_hash,  # Hash!
        perfil=TipoUsuario.FORNECEDOR,
        cpf=cpf,
        telefone=telefone,
        # ... outros campos
    )

    # 4. Salvar no banco
    fornecedor_id = fornecedor_repo.inserir(fornecedor)

    # 5. Redirecionar
    informar_sucesso(request, "Cadastro realizado com sucesso!")
    return RedirectResponse("/login", status.HTTP_303_SEE_OTHER)
```

2.2 Fluxo de Login

```
┌─────────────────────────────────────────────────────────────┐
│ FLUXO DE LOGIN                                              │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│ 1. Usuário acessa /login                                    │
│                 ↓                                           │
│ 2. Preenche formulário:                                     │
│    - Email                                                  │
│    - Senha                                                  │
│                 ↓                                           │
│ 3. Submit do formulário → POST /login                       │
│                 ↓                                           │
│ 4. Backend busca usuário por email:                         │
│    usuario = usuario_repo.obter_usuario_por_email(email)    │
│                 ↓                                           │
│ 5. Verifica se usuário existe                               │
│    if not usuario:                                          │
│        return "Email ou senha inválidos"                    │
│                 ↓                                           │
│ 6. Verifica senha com bcrypt:                               │
│    senha_correta = verificar_senha(                         │
│        senha_digitada,                                      │
│        usuario.senha  # Hash do banco                       │
│    )                                                        │
│                 ↓                                           │
│ 7. Se senha incorreta:                                      │
│    → return "Email ou senha inválidos"                      │
│    → Log de tentativa falhada                               │
│                                                             │
│ 8. Se senha correta:                                        │
│    a) Converter usuário para dicionário                     │
│       usuario_dict = usuario_para_sessao(usuario)           │
│       # Remove senha, converte Enum                         │
│                 ↓                                           │
│    b) Criar sessão                                          │
│       criar_sessao(request, usuario_dict)                   │
│       # Armazena dados no session cookie                    │
│                 ↓                                           │
│    c) Log de sucesso                                        │
│       logger.info("Login bem-sucedido", usuario_id)         │
│                 ↓                                           │
│    d) Redirecionar baseado no perfil:                       │
│       - ADMIN → /admin/dashboard                            │
│       - FORNECEDOR → /fornecedor/dashboard                  │
│       - NOIVO → /noivo/dashboard                            │
│                                                             │
│ 9. Navegador recebe Set-Cookie com session_id               │
│                 ↓                                           │
│ 10. Próximas requisições incluem cookie automaticamente     │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Código resumido:

```
@router.post("/login")
async def post_login(
    request: Request,
    email: str = Form(...),
    senha: str = Form(...),
):
    # 1. Buscar usuário
    usuario = usuario_repo.obter_usuario_por_email(email)

    # 2. Verificar existência e senha
    if not usuario or not verificar_senha(senha, usuario.senha):
        return {"erro": "Email ou senha inválidos"}

    # 3. Criar sessão
    usuario_dict = usuario_para_sessao(usuario)
    criar_sessao(request, usuario_dict)

    # 4. Redirecionar
    if usuario.perfil == TipoUsuario.ADMIN:
        return RedirectResponse("/admin/dashboard")
    elif usuario.perfil == TipoUsuario.FORNECEDOR:
        return RedirectResponse("/fornecedor/dashboard")
    else:
        return RedirectResponse("/noivo/dashboard")
```

2.3 Fluxo de Acesso a Rota Protegida

```
┌─────────────────────────────────────────────────────────────┐
│ FLUXO DE ACESSO A ROTA PROTEGIDA                            │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│ 1. Usuário tenta acessar /admin/usuarios                    │
│                 ↓                                           │
│ 2. Requisição chega na rota:                                │
│                                                             │
│    @router.get("/admin/usuarios")                           │
│    @requer_autenticacao(['ADMIN'])                          │
│    async def listar_usuarios(request, usuario_logado):      │
│        ...                                                  │
│                 ↓                                           │
│ 3. Decorator @requer_autenticacao é executado ANTES         │
│    da função listar_usuarios                                │
│                 ↓                                           │
│ 4. Decorator verifica sessão:                               │
│    usuario = request.session.get('usuario')                 │
│                 ↓                                           │
│ 5. CASO 1: Usuário não está logado (sessão vazia)           │
│    if not usuario:                                          │
│        return RedirectResponse(                             │
│            '/login?redirect=/admin/usuarios')               │
│    → Redireciona para login                                 │
│    → Após login, volta para /admin/usuarios                 │
│                                                             │
│ 6. CASO 2: Usuário está logado mas sem permissão            │
│    (Exemplo: perfil='FORNECEDOR')                           │
│                 ↓                                           │
│    if usuario['perfil'] not in ['ADMIN']:                   │
│        raise HTTPException(403, "Sem permissão")            │
│    → Retorna erro 403 Forbidden                             │
│                                                             │
│ 7. CASO 3: Usuário está logado E tem permissão              │
│    (perfil='ADMIN')                                         │
│                 ↓                                           │
│    kwargs['usuario_logado'] = usuario                       │
│    return await func(*args, **kwargs)                       │
│    → Injeta usuario_logado como parâmetro                   │
│    → Executa a função listar_usuarios                       │
│                 ↓                                           │
│ 8. Função listar_usuarios é executada:                      │
│                                                             │
│    async def listar_usuarios(request, usuario_logado):      │
│        # usuario_logado já está disponível                  │
│        admin_id = usuario_logado['id']                      │
│        usuarios = usuario_repo.obter_todos()                │
│        return render_template(...)                          │
│                 ↓                                           │
│ 9. Resposta é retornada ao usuário                          │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Comparação dos 3 casos:

```
 # CASO 1: Não logado
request.session = {}
# Resultado: Redirect para /login

# CASO 2: Logado mas sem permissão
request.session = {
    'usuario': {
        'id': 42,
        'nome': 'João Fornecedor',
        'perfil': 'FORNECEDOR'  # ← Não é ADMIN
    }
}
# Resultado: HTTPException 403

# CASO 3: Logado com permissão
request.session = {
    'usuario': {
        'id': 1,
        'nome': 'Admin',
        'perfil': 'ADMIN'  # ← É ADMIN
    }
}
# Resultado: Função executada normalmente
```

2.4 Fluxo de Recuperação de Senha

```
┌─────────────────────────────────────────────────────────┐
│ FLUXO COMPLETO DE RECUPERAÇÃO DE SENHA                  │
├─────────────────────────────────────────────────────────┤
│                                                         │
│ ETAPA 1: SOLICITAR RECUPERAÇÃO                          │
│                                                         │
│ 1. Usuário acessa /esqueci-senha                        │
│                  ↓                                      │
│ 2. Preenche formulário com email                        │
│                  ↓                                      │
│ 3. POST /esqueci-senha                                  │
│                  ↓                                      │
│ 4. Backend busca usuário por email                      │
│    usuario = usuario_repo.obter_usuario_por_email(email)    │
│                  ↓                                      │
│ 5. Se usuário não existe:                               │
│    → AINDA ASSIM mostra mensagem de sucesso             │
│    → Evita revelar emails cadastrados (segurança)       │
│                                                         │
│ 6. Se usuário existe:                                   │
│    a) Gerar token aleatório                             │
│       token = "xK9mP2aB7cD1..."  # 32 caracteres        │
│                  ↓                                      │
│    b) Calcular data de expiração                        │
│       expira_em = agora + 24 horas                      │
│                  ↓                                      │
│    c) Salvar no banco                                   │
│       usuario.token_redefinicao = token                 │
│       usuario.data_token = expira_em                    │
│       usuario_repo.atualizar(usuario)                   │
│                  ↓                                      │
│    d) Enviar email com link                             │
│       Link: /reset-senha?token=xK9mP2aB7cD1...          │
│                  ↓                                      │
│ 7. Mostrar mensagem: "Se o email estiver cadastrado..." │
│                                                         │
├─────────────────────────────────────────────────────────┤
│                                                         │
```

```
| ETAPA 2: CLICAR NO LINK DO EMAIL                                  |
|                                                                   |
| 8. Usuário clica no link do email                                 |
|    GET /reset-senha?token=xK9mP2aB7cD1...                         |
|                 ↓                                                 |
| 9. Backend valida o token:                                        |
|                                                                   |
|    a) Token foi fornecido?                                        |
|       if not token:                                               |
|           return "Token não fornecido"                            |
|                 ↓                                                 |
|    b) Token existe no banco?                                      |
|       usuario = usuario_repo.obter_usuario_por_token(token)       |
|       if not usuario:                                             |
|           return "Token inválido"                                 |
|                 ↓                                                 |
|    c) Token não expirou?                                          |
|       agora = datetime.now()                                      |
|       expira_em = datetime.fromisoformat(usuario.data_token)      |
|       if agora > expira_em:                                       |
|           return "Token expirado"                                 |
|                 ↓                                                 |
| 10. Se token válido:                                              |
|     → Mostrar formulário de nova senha                            |
|                                                                   |
|-------------------------------------------------------------------|
|                                                                   |
| ETAPA 3: DEFINIR NOVA SENHA                                       |
|                                                                   |
| 11. Usuário preenche:                                             |
|     - Nova senha                                                  |
|     - Confirmar senha                                             |
|                 ↓                                                 |
| 12. POST /reset-senha                                             |
|                 ↓                                                 |
| 13. Backend valida:                                               |
|                                                                   |
|     a) Senhas coincidem?                                          |
|        if senha != confirmar_senha:                               |
|            return "Senhas não coincidem"                          |
|                 ↓                                                 |
|     b) Senha forte o suficiente?                                  |
|        valida, erro = validar_forca_senha(senha)                  |
|        if not valida:                                             |
|            return erro                                            |
|                 ↓                                                 |
|     c) Token ainda válido?                                        |
|        (Repetir validações da etapa 9)                            |
|                 ↓                                                 |
| 14. Se tudo válido:                                               |
|                                                                   |
|     a) Criar hash da nova senha                                   |
|        senha_hash = criar_hash_senha(senha)                       |
|                 ↓                                                 |
|     b) Atualizar usuário                                          |
|        usuario.senha = senha_hash                                 |
|        usuario.token_redefinicao = None  # Invalidar              |
|        usuario.data_token = None                                  |
|        usuario_repo.atualizar(usuario)                            |
|                 ↓                                                 |
|     c) Redirecionar para login                                    |
|        "Senha redefinida! Faça login com sua nova senha"          |
|                                                                   |
| 15. Token é invalidado e não pode ser usado novamente             |
|                                                                   |
|                                                                   |
```

```
└──────────────────────────────────────────────────────────┘
```

Por que sempre mostrar "Se o email estiver cadastrado..."?

```
# ❌ MAU: Revela se email existe
if not usuario:
    return "Email não encontrado"
else:
    return "Email enviado com sucesso"

# Atacante pode descobrir emails cadastrados:
# "teste@email.com" → "Email não encontrado"
# "joao@email.com" → "Email enviado com sucesso"  ← Email existe!

# ✅ BOM: Mesma mensagem sempre
return "Se o email estiver cadastrado, você receberá instruções"

# Atacante não consegue saber se o email existe
```

2.5 Fluxo de Logout

```
┌──────────────────────────────────────────────────────────┐
│ FLUXO DE LOGOUT                                          │
├──────────────────────────────────────────────────────────┤
│                                                          │
│ 1. Usuário clica em "Sair" / "Logout"                    │
│                  ↓                                       │
│ 2. GET /logout                                           │
│                  ↓                                       │
│ 3. Backend limpa a sessão:                               │
│    request.session.clear()                               │
│    # Remove todos os dados da sessão                     │
│                  ↓                                       │
│ 4. Cookie de sessão é invalidado                         │
│                  ↓                                       │
│ 5. Redireciona para home:                                │
│    return RedirectResponse("/")                          │
│                  ↓                                       │
│ 6. Próximas requisições não terão usuário logado         │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

Código simples:

```
@router.get("/logout")
async def logout(request: Request):
    """Encerra a sessão do usuário"""
    request.session.clear()
    return RedirectResponse("/", status.HTTP_303_SEE_OTHER)
```