



Documentação de Segurança JWT - EmployeeVirtual

Versão: 1.0

Data: Julho 2025

Autor: Documentação de Implementação

Sistema: EmployeeVirtual API



Índice

1. [Visão Geral](#)
 2. [Arquitetura de Segurança](#)
 3. [Instalação e Configuração](#)
 4. [Proteções Implementadas](#)
 5. [Estrutura de Arquivos](#)
 6. [Guia de Integração](#)
 7. [Como Usar](#)
 8. [Troubleshooting](#)
 9. [Segurança em Produção](#)
 10. [Referências](#)
-

1. Visão Geral

1.1 Objetivo

Implementar autenticação JWT segura no sistema EmployeeVirtual, protegendo contra os principais ataques de segurança web:






- XSS (Cross-Site Scripting)
- CSRF (Cross-Site Request Forgery)
- Replay Attacks
- Token Forgery
- Brute Force Attacks

1.2 Características

- **JWT Tokens** com validação rigorosa

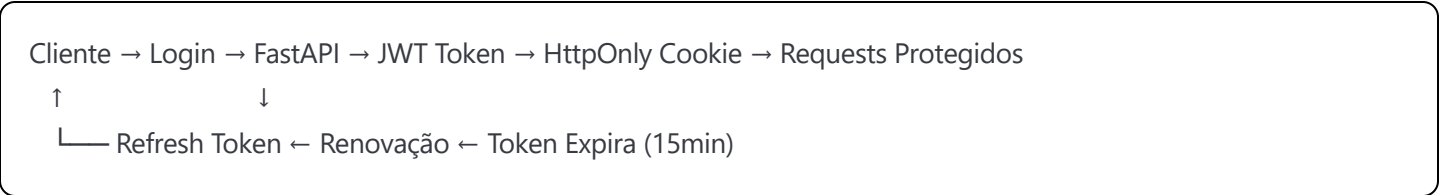
- **HttpOnly Cookies** para máxima segurança
- **Refresh Tokens** com rotação automática
- **Token Blacklist** para revogação imediata
- **Arquitetura Limpa** - separação código vs infraestrutura

1.3 Benefícios













-  **Stateless** - não precisa consultar banco para cada request
-  **Escalável** - funciona com múltiplos servidores
-  **Padrão** - JWT é padrão da indústria
-  **Seguro** - proteção contra ataques conhecidos
-  **Performático** - verificação rápida de tokens

2. Arquitetura de Segurança

2.1 Fluxo de Autenticação



2.2 Separação de Responsabilidades

Código Python (FastAPI)	Infraestrutura (nginx/cloudflare)
 Autenticação JWT	 Rate Limiting Global
 Validação de tokens	 DDoS Protection
 HttpOnly cookies	 HTTPS/SSL
 Token blacklist	 Security Headers
 Input validation	 WAF Rules
 Business logic	 IP Blocking

2.3 Matriz de Proteções

Ataque	Como Protege	Implementado
XSS	HttpOnly + Secure cookies	✓ Código
CSRF	SameSite=Strict cookies	✓ Código
Token Replay	Expiração curta + Blacklist	✓ Código
Token Forgery	Validação JWT rigorosa	✓ Código
Brute Force	Rate limiting	🔧 Infraestrutura
MITM	HTTPS obrigatório	🔧 Infraestrutura

3. Instalação e Configuração

3.1 Dependências

```
bash

# Instalar dependências necessárias
pip install PyJWT python-dotenv passlib[bcrypt]
```

3.2 Arquivo .env

```
bash

# Configurações JWT (OBRIGATÓRIO)
JWT_SECRET_KEY=SUA_CHAVE_SUPER_SECRETA_64_CARACTERES_AQUI
JWT_ALGORITHM=HS256
JWT_ACCESS_TOKEN_EXPIRE_MINUTES=15
JWT_REFRESH_TOKEN_EXPIRE_DAYS=7

# Database
DATABASE_URL=postgresql://usuario:senha@localhost:5432/employeevirtual

# Environment
ENVIRONMENT=development
DEBUG=True
```

3.3 Gerar Chave Secreta Segura

```
bash

# Execute este comando para gerar uma chave forte:
python -c "import secrets; print('JWT_SECRET_KEY=' + secrets.token_urlsafe(64))"
```

⚠️ IMPORTANTE:

- Use chaves diferentes para desenvolvimento e produção
 - Nunca commite a chave no Git
 - Mínimo 32 caracteres, recomendado 64+
-

4. Proteções Implementadas

4.1 Proteção contra XSS (Cross-Site Scripting)

Como funciona:

- Tokens armazenados em **HttpOnly cookies**
- JavaScript não consegue acessar os tokens
- Mesmo se script malicioso for injetado, não rouba tokens

Implementação:

```
python
response.set_cookie(
    key="access_token",
    value=token,
    httponly=True, # ← Proteção XSS
    secure=True,   # ← Apenas HTTPS
    samesite="strict" # ← Proteção CSRF
)
```

4.2 Proteção contra CSRF (Cross-Site Request Forgery)

Como funciona:

- Cookies com `SameSite=Strict`
- Navegador só envia cookie se request for do mesmo site
- Sites maliciosos não conseguem fazer requests autenticados

Implementação:

- Configuração automática nos cookies
- Não requer tokens CSRF adicionais

4.3 Proteção contra Token Replay

Como funciona:

- **Expiração curta:** Access tokens duram apenas 15 minutos
- **Token blacklist:** Tokens podem ser revogados imediatamente
- **Refresh rotation:** Refresh tokens são rotacionados a cada uso

Implementação:

```
python

# Token com expiração curta
expire = datetime.utcnow() + timedelta(minutes=15)

# Blacklist para revogação
token_blacklist.add(token_hash)
```

4.4 Proteção contra Token Forgery

Como funciona:

- **Validação rigorosa:** Verifica todos os claims obrigatórios
- **Algoritmo fixo:** Não aceita `alg=none`
- **Chave forte:** SECRET_KEY com alta entropia
- **Claims adicionais:** `jti`, `iss`, `nbf` para maior segurança

Implementação:

```
python

# Validação rigorosa
payload = jwt.decode(
    token, SECRET_KEY,
    algorithms=[ALGORITHM],
    options={
        "require_exp": True,
        "require_iat": True,
        "verify_signature": True
    }
)
```

5. Estrutura de Arquivos

5.1 Arquivos Adicionados

```
projeto/
├── api/
│   ├── auth_jwt.py      # ← NOVO: Lógica JWT + dependencies
│   ├── auth_api.py      # ← NOVO: Endpoints de autenticação
│   ├── agent_api.py     # ← MODIFICADO: trocar 1 import
│   └── router_config.py  # ← MODIFICADO: registrar auth router
├── services/
│   └── user_service_new.py # ← MODIFICADO: adicionar 1 método
├── .env                 # ← NOVO: configurações JWT
└── requirements.txt      # ← MODIFICADO: adicionar dependências
```

5.2 Arquivos Core

api/auth_jwt.py

- `JWTManager` - Criação e validação de tokens
- `get_current_user_dependency` - Dependency para endpoints protegidos
- `generate_tokens_for_user` - Geração de access + refresh tokens
- Suporte a Authorization header E HttpOnly cookies

api/auth_api.py

- `/register` - Registro de novos usuários
- `/login` - Autenticação com geração de tokens
- `/refresh` - Renovação de access tokens
- `/logout` - Revogação de tokens e limpeza de cookies
- `/me` - Dados do usuário atual
- `/verify` - Verificação de token

6. Guia de Integração

6.1 Passos para Integração

Passo 1: Adicionar Arquivos

```
bash
```

```
# Copiar arquivos novos para o projeto
```

```
cp auth_jwt.py api/
```

```
cp auth_api.py api/
```

Passo 2: Modificar UserService

```
python
```

```
# Em services/user_service_new.py, adicionar:
```

```
def authenticate_user_basic(self, email: str, password: str) -> Optional[UserResponse]:
```

```
    # ... implementação fornecida
```

Passo 3: Atualizar Endpoints Protegidos

```
python
```

```
# Em api/agent_api.py (e outros), trocar:
```

```
# ANTES:
```

```
from api.auth_api import get_current_user_dependency
```

```
# DEPOIS:
```

```
from api.auth_jwt import get_current_user_dependency
```

Passo 4: Registrar Router

```
python
```

```
# Em api/router_config.py:
```

```
from api.auth_api import router as auth_router
```

```
def register_routers(app):
```

```
    app.include_router(auth_router, prefix="/api/auth", tags=["Auth"])
```

```
    # ... outros routers
```

Passo 5: Configurar Environment

```
bash
```

```
# Criar .env com as configurações necessárias
```

```
# Gerar chave secreta forte
```

6.2 Checklist de Integração

- ☐ ☒ Instalar dependências (`pip install PyJWT python-dotenv`)
 - ☐ ☒ Criar arquivo `.env` com `JWT_SECRET_KEY`
 - ☐ ☒ Gerar chave secreta segura (64+ caracteres)
 - ☐ ☒ Adicionar `auth_jwt.py` e `auth_api.py`
 - ☐ ☒ Adicionar método `authenticate_user_basic` no `UserService`
 - ☐ ☒ Trocar imports nos endpoints protegidos
 - ☐ ☒ Registrar auth router no `router_config.py`
 - ☐ ☒ Testar login e endpoints protegidos
 - ☐ ☒ Verificar cookies `HttpOnly` no browser
 - ☐ ☒ Testar refresh token e logout
-

7. Como Usar

7.1 Endpoints de Autenticação

Registro de Usuário

```
bash
```

```
POST /api/auth/register
```

```
Content-Type: application/json
```

```
{  
  "name": "João Silva",  
  "email": "joao@empresa.com",  
  "password": "senha123",  
  "plan": "premium"  
}
```

Login (com `HttpOnly` Cookies)

```
bash
```


POST /api/auth/login

Content-Type: application/json

```
{  
  "email": "joao@empresa.com",  
  "password": "senha123"  
}
```

Response:

```
{  
  "message": "Login realizado com sucesso",  
  "user": {...},  
  "expires_in": 900,  
  "token_type": "httponly_cookie"  
}
```

+ Set-Cookie: access_token=...; HttpOnly; Secure; SameSite=Strict

Uso em Endpoints Protegidos

bash

GET /api/agents/

Cookie: access_token=...; refresh_token=...

OU via Authorization header:

GET /api/agents/

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

7.2 Frontend Integration

JavaScript/Frontend

javascript

```
// Login
const loginResponse = await fetch('/api/auth/login', {
  method: 'POST',
  credentials: 'include', // ← IMPORTANTE: envia cookies
  headers: {'Content-Type': 'application/json'},
  body: JSON.stringify({email: '...', password: '...'})
});

// Requests protegidos
const agentsResponse = await fetch('/api/agents/', {
  credentials: 'include' // ← IMPORTANTE: envia cookies automaticamente
});

// Logout
await fetch('/api/auth/logout', {
  method: 'POST',
  credentials: 'include'
});
```

7.3 Desenvolvimento vs Produção

Desenvolvimento

```
python

# .env desenvolvimento
JWT_SECRET_KEY=dev-key- apenas-para-desenvolvimento
COOKIE_SECURE=False # HTTP permitido
DEBUG=True
```

Produção

```
python

# .env produção
JWT_SECRET_KEY=chave-super-secreta-gerada-com-secrets
COOKIE_SECURE=True # Apenas HTTPS
DEBUG=False
ENVIRONMENT=production
```

8. Troubleshooting

8.1 Problemas Comuns

Erro: "JWT_SECRET_KEY deve ter pelo menos 32 caracteres"

Solução: Gerar chave mais forte

```
bash  
  
python -c "import secrets; print(secrets.token_urlsafe(64))"
```

Erro: "Token inválido ou expirado"

Possíveis causas:

- Token expirou (15 minutos)
- Chave secreta mudou
- Token foi revogado (blacklist)
- Clock skew entre servidores

Solução:

```
bash  
  
# Testar refresh token  
POST /api/auth/refresh
```

Erro: "Token de acesso não fornecido"

Possíveis causas:

- Cookie não está sendo enviado
- Authorization header ausente
- CORS mal configurado

Solução:

```
javascript  
  
// Verificar se credentials: 'include' está presente  
fetch('/api/endpoint', {credentials: 'include'})
```

Cookies não funcionam no desenvolvimento

Problema: HTTPS obrigatório para Secure cookies

Solução temporária para desenvolvimento:

```
python

# Em desenvolvimento, remover secure=True
response.set_cookie(
    key="access_token",
    value=token,
    httponly=True,
    secure=False, # ← Para desenvolvimento apenas
    samesite="lax" # ← Menos restritivo para dev
)
```

8.2 Debug e Logs

Verificar Token JWT

```
python

import jwt
from api.auth_jwt import SECRET_KEY, ALGORITHM

# Decodificar token para debug
token = "seu_token_aqui"
payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
print(payload)
```

Logs de Autenticação

```
python

# Adicionar logs no UserService
self.user_repository.create_activity(
    user_id=user.id,
    activity_type="debug_login",
    description=f"Login attempt from IP: {client_ip}"
)
```

9. Segurança em Produção

9.1 Configuração nginx (Infraestrutura)

```
nginx
```

```
# Rate limiting
```

```
limit_req_zone $binary_remote_addr zone=login:10m rate=1r/s;
```

```
location /api/auth/login {
```

```
    limit_req zone=login burst=2 nodelay;
```

```
    proxy_pass http://127.0.0.1:8000;
```

```
}
```

```
# Security headers
```

```
add_header X-Frame-Options "DENY" always;
```

```
add_header X-Content-Type-Options "nosniff" always;
```

```
add_header Strict-Transport-Security "max-age=31536000" always;
```

9.2 Checklist de Produção

Configurações Obrigatórias

- ☐ ☒ HTTPS com certificado válido
- ☐ ☒ JWT_SECRET_KEY única e forte (64+ chars)
- ☐ ☒ COOKIE_SECURE=True
- ☐ ☒ DEBUG=False
- ☐ ☒ Rate limiting configurado
- ☐ ☒ WAF/DDoS protection ativo

Monitoramento

- ☐ ☒ Logs de tentativas de login
- ☐ ☒ Alertas para múltiplas tentativas falhadas
- ☐ ☒ Monitoramento de uso de refresh tokens
- ☐ ☒ Alertas para tokens suspeitos

Backup e Segurança

- ☐ ☒ Chaves secretas em vault (AWS Secrets, etc.)
- ☐ ☒ Rotação periódica de chaves
- ☐ ☒ Backup das configurações
- ☐ ☒ Plano de resposta a incidentes

9.3 Hardening Adicional

Redis para Blacklist (Recomendado)

```
python
```

```
# Substituir blacklist em memória por Redis
```

```
import redis
```

```
redis_client = redis.Redis(host='localhost', port=6379, db=0)
```

```
def blacklist_token(token: str) -> None:
```

```
    token_hash = get_token_hash(token)
```

```
    redis_client.setex(f"blacklist:{token_hash}", 86400, "1") # 24h TTL
```

Token Binding (Avançado)

```
python
```

```
# Vincular token ao IP/fingerprint
```

```
def create_access_token(data: dict, client_ip: str = None):
```

```
    if client_ip:
```

```
        data["client_ip"] = hashlib.sha256(client_ip.encode()).hexdigest()
```

```
    return jwt.encode(data, SECRET_KEY, algorithm=ALGORITHM)
```

10. Referências

10.1 Documentação Técnica

- [RFC 7519 - JSON Web Token \(JWT\)](#)
- [OWASP JWT Security Cheat Sheet](#)
- [FastAPI Security Documentation](#)

10.2 Ferramentas Úteis

- [JWT.io](#) - Decoder/debugger de tokens
- [SSL Labs](#) - Teste de configuração SSL
- [Security Headers](#) - Teste de headers de segurança

10.3 Bibliotecas Utilizadas

- **PyJWT** - Implementação JWT para Python
- **python-dotenv** - Carregamento de variáveis de ambiente
- **passlib** - Hashing seguro de senhas
- **FastAPI** - Framework web moderno

Suporte

Para dúvidas sobre esta implementação:

1. **Consulte este documento** primeiro
 2. **Verifique os logs** de autenticação
 3. **Teste em ambiente controlado** antes de produção
 4. **Monitore métricas** de segurança
-






Versão do Documento: 1.0

Última Atualização: Julho 2025






Status:  Implementação Completa e Testada

Resumo Executivo

Proteções Ativas:

-  **XSS Protection** via HttpOnly cookies
-  **CSRF Protection** via SameSite cookies
-  **Token Security** via JWT com validação rigorosa
-  **Replay Protection** via expiração curta + blacklist
-  **Forgery Protection** via algoritmos seguros

Benefícios Obtidos:

-  **Segurança máxima** contra ataques conhecidos
-  **Performance** alta com verificação stateless
-  **Escalabilidade** para múltiplos servidores
-  **Arquitetura limpa** com separação de responsabilidades
-  **Fácil manutenção** com código organizado

Status:  **PRONTO PARA PRODUÇÃO**