

Documentação JWT - EmployeeVirtual

Sistema de Autenticação Seguro

Versão: 1.0

Data: Julho 2025






Índice

1. [Visão Geral](#)
 2. [Estrutura da Pasta auth/](#)
 3. [Como Usar nos Endpoints](#)
 4. [Configuração](#)
 5. [Testando com Postman](#)
 6. [Exemplos Práticos](#)
 7. [Troubleshooting](#)
 8. [Fluxos de Autenticação](#)
-

1. Visão Geral

O que é?



Sistema de autenticação JWT (JSON Web Token) com **máxima segurança**:



-  **HttpOnly Cookies** - protege contra XSS
-  **SameSite Cookies** - protege contra CSRF
-  **Token Blacklist** - logout instantâneo
-  **Refresh Tokens** - renovação automática
-  **Expiração curta** - 15 minutos por segurança

Como funciona?

Login → JWT Token → HttpOnly Cookie → Requests protegidos 

Vantagens

-  **Seguro** - protege contra ataques web
-  **Rápido** - não consulta banco a cada request

-  **Escalável** - funciona com múltiplos servidores
 -  **Simples** - 1 linha por endpoint
-

2. Estrutura da Pasta auth/

```
auth/  
├── config.py      # Configurações JWT (.env)  
├── jwt_service.py # Lógica de tokens (criar/validar)  
└── dependencies.py # Dependencies FastAPI (get_current_user)
```

auth/config.py

O que faz: Lê configurações do `.env`

```
python  
  
JWT_SECRET_KEY = "chave-secreta-do-env"  
ACCESS_TOKEN_EXPIRE_MINUTES = 15  
COOKIE_SECURE = True # HTTPS only
```

auth/jwt_service.py

O que faz: Trabalha com tokens JWT

```
python  
  
class JWTService:  
    def create_access_token(user_id, email) → str  
    def verify_token(token) → dict | None  
    def blacklist_token(token) → None
```

auth/dependencies.py

O que faz: Dependencies para proteger endpoints

```
python  
  
async def get_current_user(request, db) → UserResponse  
async def require_premium_user(user) → UserResponse  
async def require_admin_user(user) → UserResponse
```

3. Como Usar nos Endpoints

Importar

```
python

from auth.dependencies import get_current_user
```

Endpoint protegido básico

```
python

@router.post("/agents")
async def create_agent(
    data: AgentCreate,
    user: UserResponse = Depends(get_current_user), # ← Protegido!
    db: Session = Depends(get_db)
):
    # user.id, user.email, user.name já disponíveis! ✨
    return {"message": f"Agente criado por {user.name}"}
```

Diferentes níveis de proteção

```
python
```

```

from auth.dependencies import (
    get_current_user,      # Básico
    get_current_user_optional, # Opcional
    require_premium_user,   # Só premium
    require_admin_user      # Só admin
)

# Endpoint público
@router.get("/public")
async def public_endpoint():
    return {"message": "Todo mundo pode acessar"}

# Endpoint protegido
@router.get("/protected")
async def protected_endpoint(user = Depends(get_current_user)):
    return {"message": f"Olá {user.name}!"}

# Endpoint opcional (personaliza se logado)
@router.get("/personalized")
async def personalized(user = Depends(get_current_user_optional)):
    if user:
        return {"message": f"Bem-vindo {user.name}!", "vip": True}
    else:
        return {"message": "Olá visitante!", "vip": False}

# Endpoint premium
@router.post("/premium-feature")
async def premium_only(user = Depends(require_premium_user)):
    return {"message": f"Recurso premium para {user.name}! 💎"}

# Endpoint admin
@router.delete("/admin/users/{user_id}")
async def admin_only(user_id: int, admin = Depends(require_admin_user)):
    return {"message": f"Admin {admin.name} deletou usuário {user_id}"}

```

4. Configuração

Arquivo .env (obrigatório)

```
bash
```

```
# JWT Configuration
```

```
JWT_SECRET_KEY=sua_chave_super_secreta_64_caracteres_aqui_mude_em_producao
```

```
JWT_ALGORITHM=HS256
```

```
JWT_ACCESS_TOKEN_EXPIRE_MINUTES=15
```

```
JWT_REFRESH_TOKEN_EXPIRE_DAYS=7
```

```
# Database
```

```
DATABASE_URL=postgresql://user:pass@localhost:5432/db
```

```
# Environment
```

```
ENVIRONMENT=development
```

```
DEBUG=True
```

Gerar chave secreta segura

```
bash
```

```
python -c "import secrets; print('JWT_SECRET_KEY=' + secrets.token_urlsafe(64))"
```

Dependências necessárias

```
bash
```

```
pip install PyJWT python-dotenv passlib[bcrypt]
```

Adicionar no UserService

```
python
```

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

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

```
    user = self.user_repository.get_user_by_email(email)
```

```
    if not user or not self._verify_password(password, user.password_hash):
```

```
        return None
```

```
    self.user_repository.update_user(user.id, last_login=datetime.utcnow())
```

```
    return UserResponse(
```

```
        id=user.id, name=user.name, email=user.email,
```

```
        plan=user.plan, created_at=user.created_at,
```

```
        updated_at=user.updated_at, last_login=user.last_login
```

```
)
```

5. Testando com Postman

Método 1: HttpOnly Cookies (Recomendado) 🍪

Passo 1: Login

```
http
POST http://localhost:8000/api/auth/login
Content-Type: application/json

{
  "email": "user@test.com",
  "password": "senha123"
}
```

Passo 2: Postman salva cookies automaticamente! ✨

Resposta:

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

Passo 3: Endpoints protegidos funcionam automaticamente

```
http
GET http://localhost:8000/api/agents/
# ← Cookie enviado automaticamente!
```

Método 2: Bearer Token (Tradicional) 🗝️

Passo 1: Login sem cookies

```
http
```

POST http://localhost:8000/api/auth/login?use_cookies=false

Content-Type: application/json

```
{  
  "email": "user@test.com",  
  "password": "senha123"  
}
```

Passo 2: Copiar token da resposta

```
json  
  
{  
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
  "token_type": "bearer"  
}
```

Passo 3: Configurar Authorization no Postman

- Aba "Authorization"
- Type: "Bearer Token"
- Token: [colar access_token]

Passo 4: Testar endpoint protegido

http

GET http://localhost:8000/api/agents/

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

6. Exemplos Práticos

Endpoint de agentes (agent_api.py)

python

```

from fastapi import APIRouter, Depends, HTTPException
from auth.dependencies import get_current_user, require_premium_user

router = APIRouter()

@router.get("/")
async def list_agents(user = Depends(get_current_user)):
    """Lista agentes do usuário"""
    return {"agents": [], "user_id": user.id}

@router.post("/")
async def create_agent(
    data: AgentCreate,
    user = Depends(get_current_user)
):
    """Cria novo agente"""
    return {"message": f"Agente criado por {user.name}", "data": data}

@router.post("/premium")
async def create_premium_agent(
    data: AgentCreate,
    user = Depends(require_premium_user) # Só premium!
):
    """Cria agente premium (só usuários premium)"""
    return {"message": f"Agente premium criado por {user.name}! 💎"}

```

Endpoints de autenticação (auth_api.py)

python


```

from fastapi import APIRouter, Response, Request
from auth.dependencies import get_current_user
from auth.jwt_service import create_token_pair, JWTService

router = APIRouter()

@router.post("/login")
async def login(response: Response, login_data: UserLogin):
    """Login com JWT e cookies"""
    user_service = UserService(db)
    user = user_service.authenticate_user_basic(login_data.email, login_data.password)

    if not user:
        raise HTTPException(status_code=401, detail="Credenciais inválidas")

    # Gerar tokens
    tokens = create_token_pair(user.id, user.email)

    # Configurar cookie seguro
    response.set_cookie(
        key="access_token",
        value=tokens["access_token"],
        httponly=True, # XSS protection
        secure=True, # HTTPS only
        samesite="strict" # CSRF protection
    )

    return {"message": "Login realizado", "user": tokens["user"]}

@router.get("/me")
async def get_me(user = Depends(get_current_user)):
    """Dados do usuário atual"""
    return user

@router.post("/logout")
async def logout(request: Request, response: Response, user = Depends(get_current_user)):
    """Logout seguro"""
    token = request.cookies.get("access_token")
    if token:
        JWTService.blacklist_token(token) # Revogar token

```

```
response.delete_cookie("access_token")  
return {"message": "Logout realizado"}
```

7. Troubleshooting

Problemas comuns

✗ "JWT_SECRET_KEY deve ter pelo menos 32 caracteres"

Solução: Gerar chave mais forte

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

✗ "Token inválido ou expirado"

Possíveis causas:

- Token expirou (15 minutos)
- Chave secreta mudou
- Token foi revogado (logout)

Solução: Fazer novo login ou refresh

✗ "Token de acesso não fornecido"

Possíveis causas:

- Cookie não enviado
- Authorization header ausente

Solução Postman:

- Verificar se cookies estão habilitados
- Ou usar Bearer Token manual

✗ **ModuleNotFoundError: auth**

Solução: Verificar estrutura de pastas

```
projeto/
├── auth/
│   ├── config.py
│   ├── jwt_service.py
│   └── dependencies.py
└── api/
    └── agent_api.py
```

Debug úteis

Ver informações do token

```
python

from auth.jwt_service import JWTService

token = "seu_token_aqui"
info = JWTService.get_token_info(token)
print(info)
```

Endpoint de debug

```
http

GET /api/auth/token-info
# Mostra informações do token atual
```

Logs de atividade

```
python

# UserService já loga automaticamente:
# - Login realizado
# - Token criado
# - Logout realizado
```

8. Fluxos de Autenticação

Fluxo completo com cookies

```
mermaid
```

sequenceDiagram

participant C as Cliente

participant A as API

participant DB as Database

C->>A: POST /login (email, senha)

A->>DB: Verificar usuário

DB-->>A: Usuário válido

A->>A: Gerar JWT token

A-->>C: Set-Cookie: access_token (HttpOnly)

C->>A: GET /agents/ (cookie automático)

A->>A: Verificar JWT do cookie

A->>DB: Buscar usuário

DB-->>A: Dados do usuário

A-->>C: Lista de agentes

C->>A: POST /logout

A->>A: Blacklist token

A-->>C: Clear cookie

Fluxo de refresh token

mermaid

sequenceDiagram

participant C as Cliente

participant A as API

Note over C: Access token expira (15 min)

C->>A: Request com token expirado

A-->>C: 401 Unauthorized

C->>A: POST /refresh (refresh_token cookie)

A->>A: Verificar refresh token

A->>A: Gerar novo access token

A-->>C: Set-Cookie: novo access_token

C->>A: Repetir request original

A-->>C: Sucesso com novo token

Diferentes níveis de acesso

mermaid

graph TD

```
A[Request] --> B{Token válido?}
B -->|Não| C[401 Unauthorized]
B -->|Sim| D{Usuário ativo?}
D -->|Não| E[401 User not found]
D -->|Sim| F{Endpoint requer premium?}
F -->|Não| G[✅ Acesso liberado]
F -->|Sim| H{Usuário é premium?}
H -->|Não| I[402 Premium required]
H -->|Sim| J{Endpoint requer admin?}
J -->|Não| K[✅ Acesso premium]
J -->|Sim| L{Usuário é admin?}
L -->|Não| M[403 Admin required]
L -->|Sim| N[✅ Acesso admin]
```

Resumo de Segurança

Proteção	Implementado	Como funciona
XSS	✅ HttpOnly cookies	JavaScript não acessa tokens
CSRF	✅ SameSite cookies	Navegador só envia de mesmo site
Token Replay	✅ Expiração + Blacklist	Tokens expiram em 15min
Token Forgery	✅ JWT assinado	Validação rigorosa de assinatura
Brute Force	🔧 Infraestrutura	Rate limiting no nginx
MITM	🔧 Infraestrutura	HTTPS obrigatório

Quick Start

1. **Configurar .env** com JWT_SECRET_KEY
2. **Instalar dependências:** `pip install PyJWT python-dotenv`
3. **Adicionar método no UserService** (código fornecido)
4. **Importar nos endpoints:** `from auth.dependencies import get_current_user`
5. **Testar no Postman** com login + requests protegidos

Pronto! Sistema seguro em 5 minutos! ✨

