

Guia Completo: Autenticação FastAPI + Frontend

Introdução

Este guia explica como funciona a **injeção de dependência** e **autenticação** no FastAPI, desde o login até a validação de cada requisição.

1. O que é `Depends`?

`Depends` é uma função especial do FastAPI que diz: "Execute esta função ANTES e me passe o resultado".

Exemplo Simples:






```
python

from fastapi import Depends

# Função que será executada ANTES
def get_database():
    print("Conectando no banco...")
    return "conexao_banco"

# Sua rota
@app.get("/users")
def get_users(db = Depends(get_database)): # ← Aqui está o Depends!
    print(f"Usando: {db}")
    return {"users": []}
```

O que acontece:

1.  Alguém chama `GET /users`
 2.  FastAPI vê o `Depends(get_database)`
 3.  FastAPI executa `get_database()` primeiro
 4.  FastAPI pega o resultado e passa para `db`
 5.  Sua função executa com `db = "conexao_banco"`
-

2. HTTPBearer - Extraíndo Tokens

```
python
```

```
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
```

```
security = HTTPBearer() # ← Dependência pré-construída do FastAPI
```

 O que `HTTPBearer()` faz:

 **Extraí automaticamente o token do header:**

```
http
```

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
```

 **Retorna um objeto estruturado:**

```
python
```

```
HTTPAuthorizationCredentials(  
    scheme="Bearer",  
    credentials="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."  
)
```

 **Se não tiver o header:**

```
json
```

```
{  
    "detail": "Not authenticated"  
}
```

3. Fluxo Completo de Autenticação

ETAPA 1: Login (Frontend → Backend)

Frontend envia credenciais:

```
javascript
```

```
const response = await fetch('/auth/login', {
  method: 'POST',
  body: JSON.stringify({
    email: "joao@email.com",
    password: "123456"
  })
});

const data = await response.json();
// data = { token: "eyJ0eXAiOi...", user: {...} }
```

Backend valida e retorna token:

```
python

@router.post("/auth/login")
async def login(credentials: LoginRequest, db: Session = Depends(get_db)):
    # 🔍 Valida email/senha no banco
    user = authenticate_user(credentials.email, credentials.password)

    if not user:
        raise HTTPException(401, "Email ou senha inválidos")

    # 🔑 Cria o token JWT
    token = create_jwt_token(user.id)

    return {"token": token, "user": user}
```

ETAPA 2: Frontend Salva o Token

Opção 1: localStorage (mais comum)

```
javascript

// Salva na máquina local (persiste entre sessões)
localStorage.setItem('authToken', data.token);

// Para recuperar depois:
const token = localStorage.getItem('authToken');
```

Opção 2: sessionStorage

```
javascript
```

```
// Só dura enquanto a aba estiver aberta
```

```
sessionStorage.setItem('authToken', data.token);
```

🍪 Opção 3: Cookie

```
javascript
```

```
// Salva como cookie
```

```
document.cookie = `authToken=${data.token}; path=/`;
```

🔔 ETAPA 3: Enviando Token nas Requisições

```
javascript
```

```
// 🔄 Toda vez que faz uma requisição, envia o token:
```

```
const token = localStorage.getItem('authToken');
```

```
const response = await fetch('/api/agents', {  
  method: 'GET',  
  headers: {  
    'Authorization': `Bearer ${token}` // ← CRUCIAL!  
  }  
});
```

🛡️ ETAPA 4: Backend Valida o Token

```
python
```

```

async def get_current_user_dependency(
    credentials: HTTPAuthorizationCredentials = Depends(security), # ← Pega token
    db: Session = Depends(get_db) # ← Conexão banco
) -> UserResponse:

    # 🗝️ Token extraído automaticamente
    token = credentials.credentials

    user_service = UserService(db)

    # 🔍 Valida o token e busca o usuário no banco
    user = user_service.get_user_by_token(token)

    if not user:
        # ❌ Se token inválido/expirado, rejeita
        raise HTTPException(
            status_code=401,
            detail="Token inválido ou expirado"
        )

    # ✅ Retorna o usuário logado
    return user

```

👤 4. Cadeia de Dependências

📊 Hierarquia:

```

🎯 Sua rota principal
  ↓ depende de
🔒 get_current_user_dependency
  ↓ depende de
  ├── 🗝️ security (extrai token)
  └── 📄 get_db (conecta banco)

```

⚡ Ordem de execução:

```
python
```

 *FastAPI executa security*

credentials = HTTPAuthorizationCredentials(...)

 *FastAPI executa get_db*

db = SessionLocal()









 *FastAPI executa get_current_user_dependency*

user = get_current_user_dependency(credentials, db)

 *FastAPI executa SUA rota*

create_agent(agent_data, user, db)

5. Fluxo Visual Completo







-  1. [FRONTEND] Login com email/senha
↓
-  2. [BACKEND] Valida credenciais → Cria token JWT
↓
-  3. [FRONTEND] Salva token no localStorage
↓
-  4. [FRONTEND] Requisições: Authorization: Bearer TOKEN
↓
-  5. [BACKEND] security extrai token do header
↓
-  6. [BACKEND] get_current_user_dependency valida token
↓
-  7. [BACKEND] Token válido = usuário logado
-  [BACKEND] Token inválido = erro 401

6. Exemplo Prático Completo

Frontend (React/JavaScript):

javascript

//  Login

```
const login = async (email, password) => {  
  const response = await fetch('/auth/login', {  
    method: 'POST',  
    headers: { 'Content-Type': 'application/json' },  
    body: JSON.stringify({ email, password })  
  });  
  
  const data = await response.json();  
  localStorage.setItem('authToken', data.token); //  Salva na máquina  
};  
  
//  Qualquer outra requisição  
const getAgents = async () => {  
  const token = localStorage.getItem('authToken'); //  Pega da máquina  
  
  const response = await fetch('/api/agents', {  
    headers: {  
      'Authorization': `Bearer ${token}` //  Envia para o backend  
    }  
  });  
  
  return response.json();  
};  
  
//  Logout  
const logout = () => {  
  localStorage.removeItem('authToken'); //  Remove token  
};
```

Backend (FastAPI):

python

🔑 *Security setup*

security = HTTPBearer()

🛡️ *Dependência de autenticação*

```
async def get_current_user_dependency(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db: Session = Depends(get_db)
) -> UserResponse:
    user_service = UserService(db)
    user = user_service.get_user_by_token(credentials.credentials)
```

```
    if not user:
        raise HTTPException(401, "Token inválido ou expirado")
```

```
    return user
```

🎯 *Rota protegida*

@router.get("/agents")

```
async def get_agents(
    current_user: UserResponse = Depends(get_current_user_dependency)
):
```

🍌 *current_user já contém os dados do usuário logado!*

```
    return f"Olá {current_user.name}! Aqui estão seus agentes."
```

🎯 *Rota de criação*

@router.post("/agents")

```
async def create_agent(
    agent_data: AgentCreate,
    current_user: UserResponse = Depends(get_current_user_dependency),
    db: Session = Depends(get_db)
):
```

✅ *Usuário já validado automaticamente!*

```
    new_agent = Agent(
        name=agent_data.name,
        user_id=current_user.id # 🎯 Liga ao usuário logado
    )
    db.add(new_agent)
    db.commit()
    return new_agent
```

7. Resumo dos Benefícios

✅ Vantagens da Injeção de Dependência:

- 🔄 **Reutilização:** Mesmas dependências em várias rotas
- 🛠️ **Testabilidade:** Fácil de "mockar" dependências nos testes
- 🏠 **Separação:** Lógica de auth/banco separada da lógica de negócio
- ⚡ **Automático:** FastAPI gerencia tudo (conexões, validações, etc)

🚫 Como seria SEM injeção de dependência:

python

```
async def create_agent_ruim(agent_data: AgentCreate):  
    # 🤖 Teria que fazer tudo manualmente em CADA rota  
    token = request.headers.get("Authorization")  
    if not token:  
        raise HTTPException(401, "Token missing")  
  
    user = validate_token(token) # 🔄 Repetir em toda rota  
    if not user:  
        raise HTTPException(401, "Invalid token")  
  
    db = create_db_connection() # 🔄 Repetir em toda rota  
    try:  
        # Sua lógica aqui..  
    finally:  
        db.close() # 🧠 Lembrar de fechar sempre
```






🎉 Com FastAPI + Depends:

python

```
async def create_agent(  
    agent_data: AgentCreate,  
    current_user: UserResponse = Depends(get_current_user_dependency),  
    db: Session = Depends(get_db)  
):  
    # 🎯 Foco só na lógica de negócio!  
    # FastAPI fez toda a parte chata automaticamente
```

🔑 8. Pontos-Chave para Lembrar


1. 🛡️ **Frontend DEVE enviar token:** `Authorization: Bearer TOKEN`

2.  **Token salvo localmente:** `localStorage.setItem('authToken', token)`
 3.  **Backend valida automaticamente:** Via `Depends(get_current_user_dependency)`
 4.  **Dependências em cadeia:** Uma dependência pode ter outras dependências
 5.  **Tudo automático:** FastAPI gerencia a ordem de execução
 6.  **Token inválido = 401:** Automático, sem código extra
-

Conclusão

O FastAPI + sistema de dependências torna a autenticação **simples, segura e reutilizável**. Você escreve a lógica uma vez e usa em todas as rotas que precisam de autenticação!

 **Resultado:** Código mais limpo, menos bugs, e desenvolvimento mais rápido!

 *Dica: Salve este documento como referência para seus projetos FastAPI!*