

# Tornando o sistema de autenticação robusto

| <https://fastapidozero.dunossauro.com/08/>

## Objetivos da Aula

- Testar os casos de autenticação de forma correta
- Testar os casos de autorização de forma correta
- Implementar o refresh do token
- Introduzir testes que param o tempo com `freezegun`
- Introduzir geração de modelos automática com `factory-boy`

# Testes de autorização

| Parte 1

## Garantir que o user faça somente o que pode

```
@router.put('/{user_id}', response_model=UserPublic)
def update_user(
    user_id: int,
    user: UserSchema,
    session: Session,
    current_user: CurrentUser,
):
    if current_user.id != user_id:
        raise HTTPException(
            status_code=HTTPStatus.FORBIDDEN,
            detail='Not enough permissions'
        )
```

Não deve ser possível alterar via PUT os dados que não são seus

```
def test_update_user_with_wrong_user(client, user, token):
    response = client.put(
        f'/users/{user.id + 1}',
        headers={'Authorization': f'Bearer {token}'},
        json={
            'username': 'bob',
            'email': 'bob@example.com',
            'password': 'mynewpassword',
        },
    )
    assert response.status_code == HTTPStatus.FORBIDDEN
    assert response.json() == {'detail': 'Not enough permissions'}
```

Na ultima aula fizemos algo parecido com isso!

## O problema dessa abordagem

O uso do `+1` nos leva a algumas discussões interessantes:

- Validamos a situação com um "hack", não existe o `+1`
- Caso ele exista, o que vai acontecer em produção, vai funcionar?
- Como representamos um cenário mais próximo da realidade?
- **precisamos adicionar um novo user ao cenário de teste**

## Criando modelos `Users` sob demanda

Para fazer a criação de users de forma mais intuitiva e sem a preocupação de valores repetidos, podemos usar uma "**fábrica**" de usuários.

Isso pode ser feito com uma biblioteca chamada `factory-boy`:

```
poetry add --group dev factory-boy
```

| Fábrica é um padrão de projeto de construção de objetos.

# O Factory-boy

```
#conftest.py
import factory

# ...

class UserFactory(factory.Factory):
    class Meta:
        model = User

    username = factory.Sequence(lambda n: f'test{n}')
    email = factory.LazyAttribute(lambda obj: f'{obj.username}@test.com')
    password = factory.LazyAttribute(lambda obj: f'{obj.username}@example.com')
```



# O uso do factory-boy + fixture

## Aplicando a fabrica:

```
@pytest.fixture
def user(session):
    password = 'testtest'
    user = UserFactory(
        password=get_password_hash(password)
    )

    session.add(user)
    session.commit()
    session.refresh(user)

    user.clean_password = 'testtest'

    return user
```

## O cenário "real":

```
@pytest.fixture
def other_user(session):
    password = 'testtest'
    user = UserFactory(
        password=get_password_hash(password)
    )

    session.add(user)
    session.commit()
    session.refresh(user)

    user.clean_password = 'testtest'

    return user
```

## Alterando o teste de put para esse cenário

```
def test_update_user_with_wrong_user(client, other_user, token):
    response = client.put(
        f'/users/{other_user.id}',
        headers={'Authorization': f'Bearer {token}'},
        json={
            'username': 'bob',
            'email': 'bob@example.com',
            'password': 'mynewpassword',
        },
    )
    assert response.status_code == HTTPStatus.FORBIDDEN
    assert response.json() == {'detail': 'Not enough permissions'}
```

## Criando um teste de DELETE para o cenário

```
def test_delete_user_wrong_user(client, other_user, token):  
    response = client.delete(  
        f'/users/{other_user.id}',  
        headers={'Authorization': f'Bearer {token}'},  
    )  
    assert response.status_code == HTTPStatus.FORBIDDEN  
    assert response.json() == {'detail': 'Not enough permissions'}
```

# Expiração do token

| Parte 2

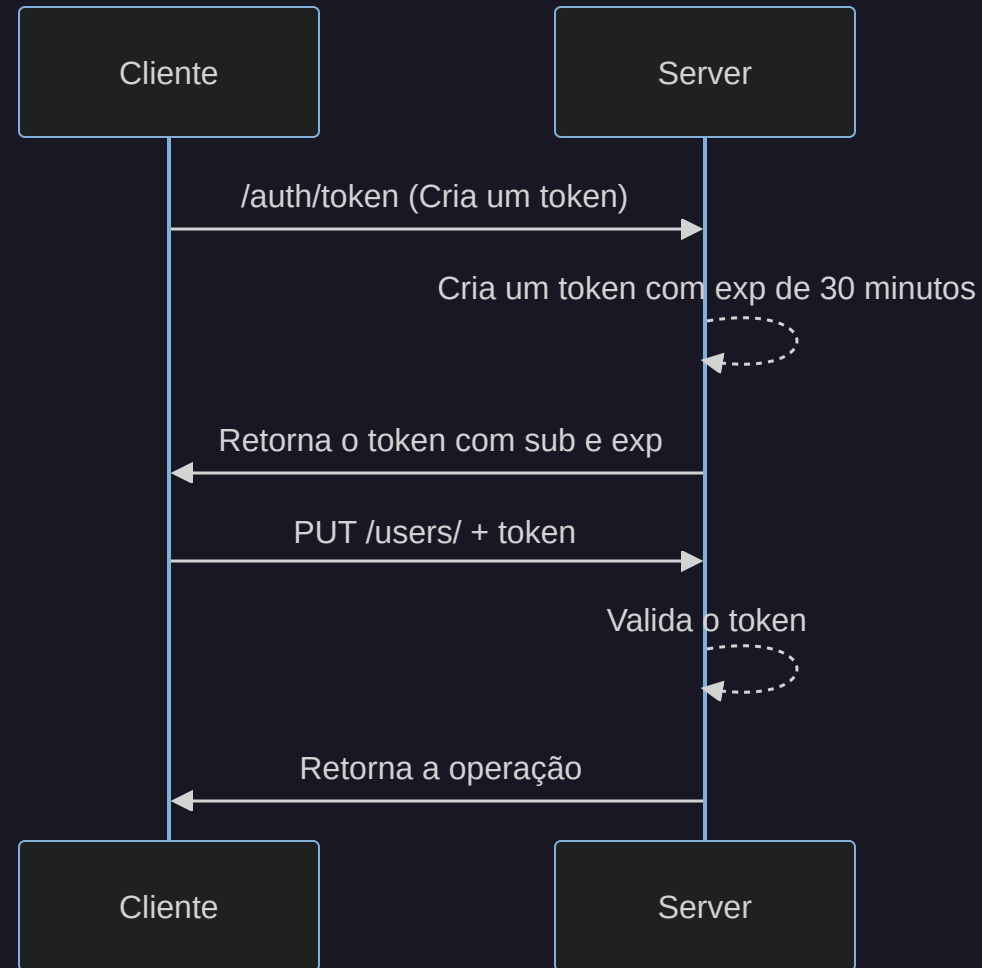
## Sobre o tempo do token

Quando geramos o token, ele tem a claim 'exp' que é referente a validade do token.

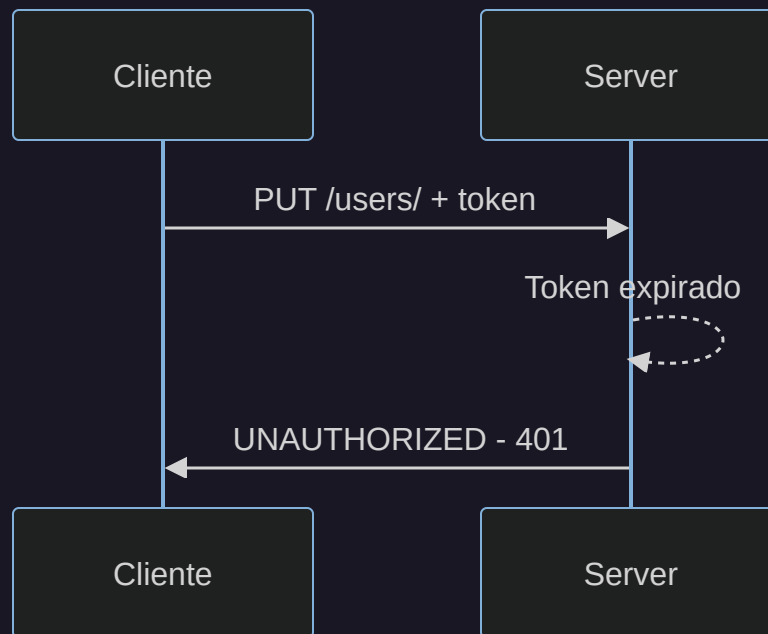
```
{  
  "exp": 1690258153,  
  "sub": "dudu@dudu.du"  
}
```

```
def create_access_token(data: dict):  
    expire = datetime.now(  
        tz=ZoneInfo('UTC')  
    ) + timedelta( # 30m  
        minutes=ACCESS_TOKEN_EXPIRE_MINUTES  
    )
```

# O sistema do token



# O tempo de duração



Esse fluxo ainda não está implementado

## Precisamos "viajar no tempo"

Para isso temos a "arma do tempo" o `freezegun`, uma biblioteca que ajuda a pausar o tempo durante os testes:

```
poetry add freezegun
```



## Usando o freezegun

```
from freezegun import freeze_time

def test_token_expired_after_time(client, user):
    # Para o tempo nessa data e hora
    with freeze_time('2023-07-14 12:00:00'):
        response = client.post(
            '/auth/token',
            data={'username': user.email, 'password': user.clean_password},
        )
        assert response.status_code == HTTPStatus.OK
        token = response.json()['access_token']
```

# Viajando no tempo

```
def test_token_expired_after_time(client, user):
    # ...
    with freeze_time('2023-07-14 12:31:00'):
        response = client.put(
            f'/users/{user.id}',
            headers={'Authorization': f'Bearer {token}'},
            json={
                'username': 'wrongwrong',
                'email': 'wrong@wrong.com',
                'password': 'wrong',
            },
        )
        assert response.status_code == HTTPStatus.UNAUTHORIZED
        assert response.json() == {'detail': 'Could not validate credentials'}
```

## A validação da expiração

```
def get_current_user(  
    try:  
        payload = decode(  
            token, settings.SECRET_KEY, algorithms=[settings.ALGORITHM]  
        )  
        username: str = payload.get('sub')  
        if not username:  
            raise credentials_exception  
        token_data = TokenData(username=username)  
    except DecodeError:  
        raise credentials_exception  
    except ExpiredSignatureError:  
        raise credentials_exception
```

# Problemas de autenticação

| Parte 4

## Algumas coisas não foram cobertas

Os nossos testes não cobrem os casos onde temos:

- Senha incorreta
- Email inexistente

## Testando a senha incorreta

```
def test_token_wrong_password(client, user):  
    response = client.post(  
        '/auth/token',  
        data={'username': user.email, 'password': 'wrong_password'}  
    )  
    assert response.status_code == HTTPStatus.BAD_REQUEST  
    assert response.json() == {'detail': 'Incorrect email or password'}
```

## Testando o user inexistente

```
def test_token_inexistent_user(client):  
    response = client.post(  
        '/auth/token',  
        data={'username': 'no_user@no_domain.com', 'password': 'testtest'},  
    )  
    assert response.status_code == HTTPStatus.BAD_REQUEST  
    assert response.json() == {'detail': 'Incorrect email or password'}
```

# Refresh do token

| Parte 5



## O que acontece quando o exp vence?

É necessário que o cliente renove o token de tempos em tempos para que ele não perca a validade e possa continuar usando a aplicação sem logar novamente.

```
# fast_zero/routes/auth.py

@router.post('/refresh_token', response_model=Token)
def refresh_access_token(
    user: User = Depends(get_current_user),
):
    new_access_token = create_access_token(data={'sub': user.email})

    return {'access_token': new_access_token, 'token_type': 'bearer'}
```

# O teste para o refresh

Ver se o refresh faz mesmo refresh :)

```
def test_refresh_token(client, user, token):  
    response = client.post(  
        '/auth/refresh_token',  
        headers={'Authorization': f'Bearer {token}'},  
    )  
  
    data = response.json()  
  
    assert response.status_code == HTTPStatus.OK  
    assert 'access_token' in data  
    assert 'token_type' in data  
    assert data['token_type'] == 'bearer'
```

# Testando o fluxo de refresh

Não se pode dar refresh depois que o token fica inválido

```
def test_token_expired_dont_refresh(client, user):  
    with freeze_time('2023-07-14 12:00:00'):  
        response = client.post(  
            '/auth/token',  
            data={'username': user.email, 'password': user.clean_password},  
        )  
        assert response.status_code == HTTPStatus.OK  
        token = response.json()['access_token']  
  
        with freeze_time('2023-07-14 12:31:00'):  
            response = client.post(  
                '/auth/refresh_token',  
                headers={'Authorization': f'Bearer {token}'},  
            )  
            assert response.status_code == HTTPStatus.UNAUTHORIZED  
            assert response.json() == {'detail': 'Could not validate credentials'}
```

## Exercício

O endpoint de PUTusa dois users criados na base de dados, porém, até o momento ele cria um novo user no teste via request na API por falta de uma fixture como other\_user. Atualize o teste para usar essa nova fixture.

# Quiz

Não esqueça de responder o quiz dessa aula!

# Commit

```
git add .  
git commit -m "Implementando o reresh do token e testes de autorização"
```