

# ESCOLA E FACULDADE DE TECNOLOGIA SENAI ROBERTO MANGE

## DESENVOLVIMENTO DE SISTEMAS

JULHO DE 2022 — SENAI-SP



ESCOLA SENAI "ROBERTO MANGE"  
UMI. REACTIVAZÃO DA INDÚSTRIA

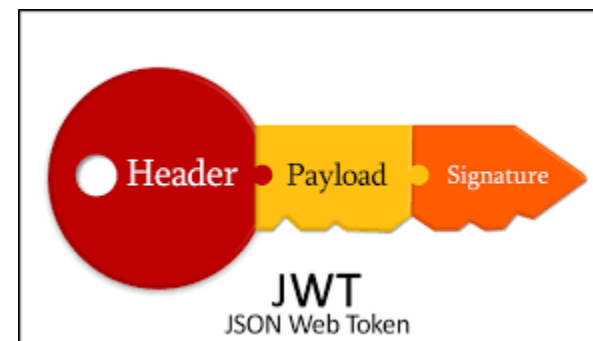
**SENAI**  
SÃO PAULO

## IMPLEMENTANDO AUTENTICAÇÃO

Até agora estávamos usando as APIs de forma a serem acessadas por qualquer um que soubesse sua rota, ou seja, estávamos construindo APIs públicas.

Porém a realidade das aplicações é que na maioria dos casos precisamos nos preocupar com o controle de acessos e permissões para cada API.

Nestes slides estaremos vendo dois modos: **Autenticação com WebToken** e **Autenticação com JWT**.



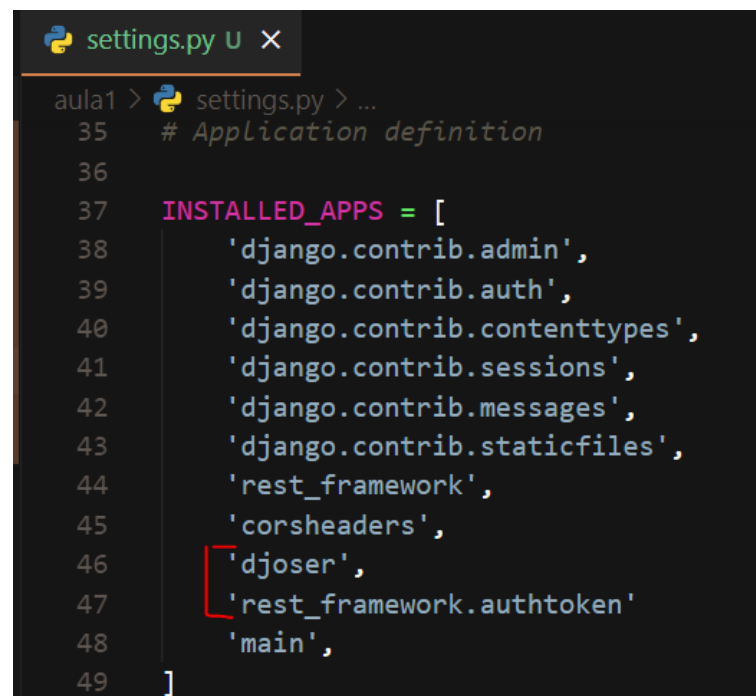
## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN

O **Web Token** é gerado cada vez que o usuário faz o login, não possuindo uma expiração por tempo de acesso, ou seja, caso o usuário não faça o logout ele permanecerá válido.

Ao ser feito o logout o token é destruído e não pode mais ser utilizado.

Obs.: Os passos a seguir são já considerando que tenha criado todo o projeto do Django como API com a biblioteca `rest_framework`, conforme passos que foram informados nos slides anteriores.

1. **pip install djoser** → instala a biblioteca que será usada para auxiliar no processo de autenticação/autorização.
2. registre o djoser e o auth token em seu projeto, dentro de `settings.py` :



```
settings.py U x
aula1 > settings.py > ...
35 # Application definition
36
37 INSTALLED_APPS = [
38     'django.contrib.admin',
39     'django.contrib.auth',
40     'django.contrib.contenttypes',
41     'django.contrib.sessions',
42     'django.contrib.messages',
43     'django.contrib.staticfiles',
44     'rest_framework',
45     'corsheaders',
46     'djoser',
47     'rest_framework.authtoken'
48     'main',
49 ]
```



## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN

3. Também em settings.py, adicionar a informação que deseja trabalhar com Web Token para a biblioteca RestFramework:

```
REST_FRAMEWORK = {  
    'DEFAULT_PERMISSION_CLASSES': [  
        'rest_framework.permissions.AllowAny'  
    ],  
    'DEFAULT_AUTHENTICATION_CLASSES': [  
        'rest_framework.authentication.TokenAuthentication'  
    ],  
}
```

4. Nas views.py, importar os tipos de permissões que deseja:

```
from rest_framework.permissions import IsAuthenticated, IsAuthenticatedOrReadOnly
```

Veja todas as existentes aqui: <https://www.django-rest-framework.org/api-guide/permissions/>

Ex.: IsAuthenticated → para se ter acesso à qualquer método é necessário estar autenticado

IsAuthenticatedOrReadOnly → para se fazer o GET não precisa estar autenticado, porém para os outros sim.

## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN

5. Nas urls.py do PROJETO, inclua as rotas padrão de autenticação geradas pelo djoser:



```
EXPLORER
PROGRAMACAO
  > __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
  main
  > __pycache__

aula1 > urls.py > ...
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('', include('main.urls')),
7     path('', include('djoser.urls')), #adiciona as rotas de gerenciamento de usuarios
8     path('auth/', include('djoser.urls.authtoken')), #adiciona a rota de obter token
9 ]
```

6. Faça os comandos de migrações:

```
py .\manage.py makemigrations
```

```
py .\manage.py migrate
```

## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN TESTANDO...

- Perceba que agora quando fizer qualquer chamada para a classe que adicionou `IsAuthenticated` (neste caso a `People`), ela irá estar bloqueada, informando que as credenciais não foram informadas:

The screenshot shows a REST client interface. The top bar displays the method `GET`, the URL `http://localhost:8000/people`, a `Send` button, and the response status `403 Forbidden` in an orange box. To the right, it shows the response time `55.5 ms`, the size `58 B`, and the timestamp `Just Now`. Below the top bar, there are tabs for `Body`, `Auth`, `Query` (with a count of 1), `Headers`, and `Docs`. The `Body` tab is selected, showing a table with columns `Name` and `Value`. The `Preview` tab is also visible, showing the response body in JSON format:

```
1 {
2   "detail": "Authentication credentials were not
provided."
3 }
```

- Faça um `POST` para a rota `/users/`, enviando via body o username, password e email que deseja cadastrar (tome atenção que o email tem que ter o formato padrão de e-mail e a senha não pode ser muito simples e deve conter letras e números), caso o backend aceite este usuário, você receberá de volta o email, username e o id deste novo user:

The screenshot shows a REST client interface. The top bar displays the method `POST`, the URL `http://localhost:8000/users/`, a `Send` button, and the response status `201 Created` in a green box. To the right, it shows the response time `471 ms`, the size `63 B`, and the timestamp `Just Now`. Below the top bar, there are tabs for `JSON`, `Auth`, `Query`, `Headers` (with a count of 2), and `Docs`. The `JSON` tab is selected, showing the request body in JSON format:

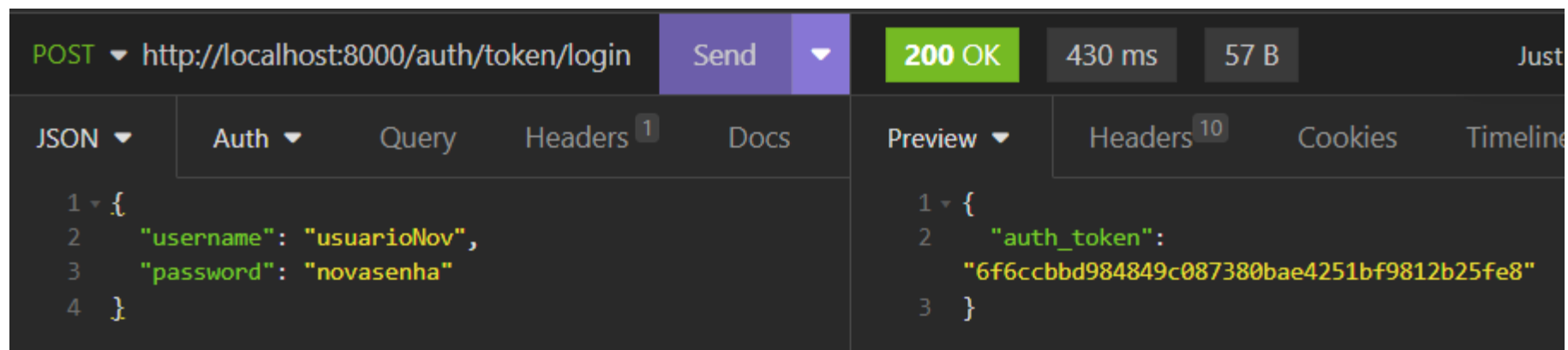
```
1 {
2   "username": "usuarioNov",
3   "password": "novasenha",
4   "email": "usuarioNov@teste.com"
5 }
```

The `Preview` tab is also visible, showing the response body in JSON format:

```
1 {
2   "email": "usuarioNov@teste.com",
3   "username": "usuarioNov",
4   "id": 3
5 }
```

## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN TESTANDO...

- Agora faça o login com o username e password do usuario criado, onde você deve obter como resposta o token (faça um POST para a rota **/auth/token/login**):



```
POST http://localhost:8000/auth/token/login 200 OK 430 ms 57 B

JSON Auth Query Headers 1 Docs Preview Headers 10 Cookies Timeline

1 {
2   "username": "usuarioNov",
3   "password": "novasenha"
4 }

1 {
2   "auth_token":
   "6f6ccbbd984849c087380bae4251bf9812b25fe8"
3 }
```

## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN TESTANDO...

- Copie o token que foi criado no seu login, e use-o passando em cada chamada no backend no cabeçalho da requisição (Header), atente-se que o formato que deve usar é:

Nome do Header: **Authorization**

Valor: **Token VALOR\_DO\_SEU\_TOKEN\_AQUI**

Entenda que, quando o backend receber este Token, ele procederá com a descryptografia deste token e dessa forma saberá que se trata do usuário que fez o login, concedendo-lhe o acesso à API.

Perceba que trabalhar com tokens é melhor pois neste processo você não precisou trafegar a sua senha.

GET http://localhost:8000/people Send 200 OK 25.7 ms 420 B Just No

Body Auth Query **Headers** 1 Docs

Add Delete All Toggle Description

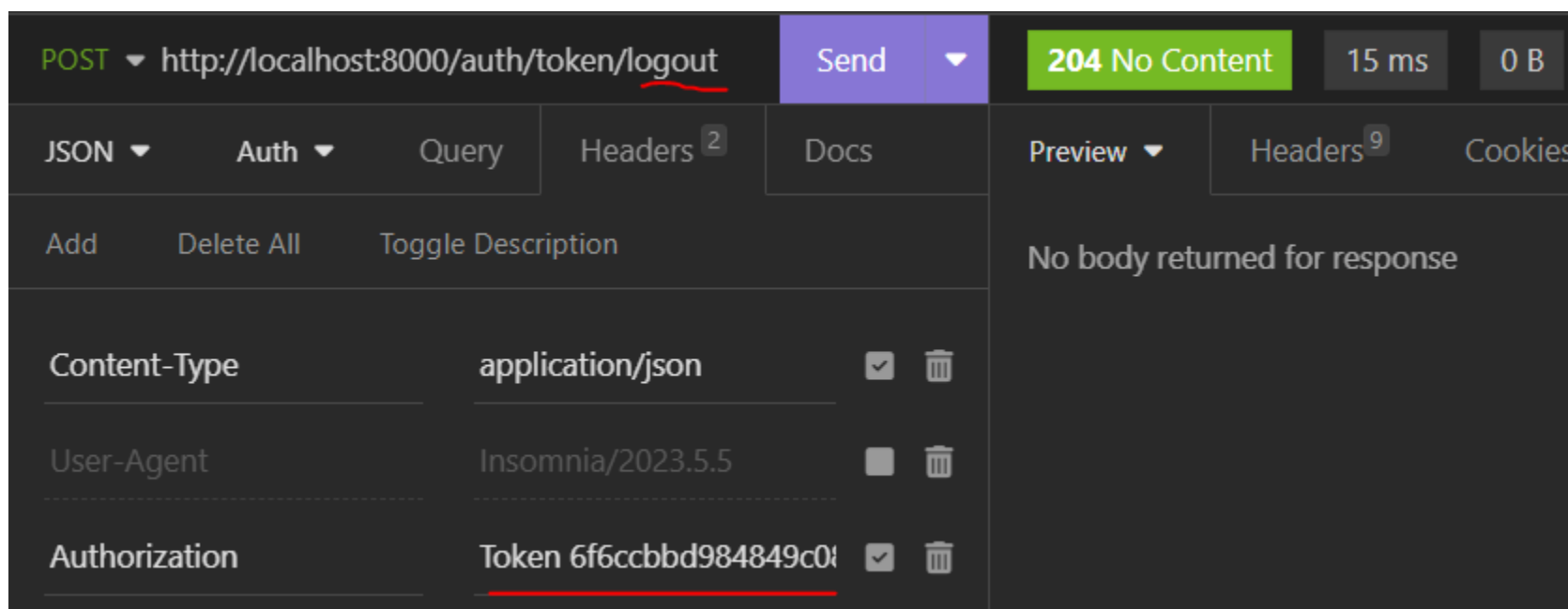
Authorization Token 6f6ccbbd984849c

```
1 [
2   {
3     "id": 1,
4     "name": "DJDJJGKKFKF",
5     "birthYear": "ABC1988",
6     "eyeColor": "Vermelho",
7     "gender": "Masculino",
8     "hairColor": "Azul",
9     "height": "1.50",
10    "mass": "68.00",
11    "created": "2023-08-03T01:17:39.680805Z",
12    "planet": 1,
13    "starship": 1
14  },
```



## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN TESTANDO...

- Quando não quiser mais utilizar a aplicação, então este usuário irá fazer o logout, e, neste momento, o token gerado será destruído e não será mais válido, vamos testar fazendo o logout:



- Note que quando enviar para a rota `/auth/token/logout`, você também deve enviar no Headers o token que deseja fazer o logout, dessa maneira o backend poderá saber qual o token que ele deve destruir.

## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN TESTANDO...

- Perceba que após o logout, o mesmo token que anteriormente era válido agora se torna inútil:

The screenshot shows a web client interface with the following details:

- Method:** GET
- URL:** http://localhost:8000/people
- Status:** 401 Unauthorized
- Time:** 41.7 ms
- Size:** 27 B
- Body:** {  
 "detail": "Invalid token."  
}
- Headers:** 11
- Cookies:**
- Authorization:** Token 6f6ccbbd984849c08 (checked)

## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN TESTANDO...

- Outro ponto que vale evidenciar é que na rota /planet, como utilizamos o `IsAuthenticatedOrReadOnly`, então para fazer o método GET não é necessário estar autenticado (ou seja, enviar o token), porém para os outros métodos sim:

```
class PlanetAPIView(ModelViewSet):  
    queryset = Planet.objects.all()  
    serializer_class = PlanetSerializer  
    filter_backends = [DjangoFilterBackend]  
    filterset_fields = ['name', 'climate', 'diameter']  
    permission_classes = (IsAuthenticatedOrReadOnly,)
```

GET http://localhost:8000/planet

Send 200 OK 24.5 ms 394 B

Body Auth Query Headers Docs

Add Delete All Toggle Description

Name	Value
------	-------

```
1 [
2   {
3     "id": 1,
4     "name": "Júpiter",
5     "climate": "Árido",
6     "created": "2023-08-03T01:04:41Z",
7     "diameter": "30.00",
8     "gravity": 5,
9     "population": 1000
10  },
```

GET funciona  
mesmo sem  
token



## IMPLEMENTANDO AUTENTICAÇÃO COM WEB TOKEN

### CARACTERÍSTICAS DO WEB TOKEN:

1. O web token permanece o mesmo enquanto o usuário não faz o logout (não expira por tempo de uso);
2. O backend armazena esse token enquanto está válido, permitindo que no momento desejado se faça o logout (exclusão do token que estava salvo)
3. Acaba apresentando o risco que, caso o usuário esqueça de fazer o logout e o token for hackeado, este token poderá ser usado por validade indeterminada.
4. É mais fácil de gerenciar na integração entre frontend e backend por ser o mesmo token durante o login inteiro do usuário.