AUTENTICACIÓN CON TOKEN

Según vimos en el apartado anterior, la configuración de permisos y accesos es muy configurable. Podemos decidir a qué vistas y cuáles nos se permite el acceso e incluso a que tipo de solicitudes y a cuáles no. Configuramos tanto los permisos como la autenticación.

```
DEFAULT_PERMISSION_CLASSES
```

```
DEFAULT_AUTHENTICATION_CLASSES
```

La autenticación se configura dentro del fichero DRF→ settings.py

En el ejemplo anterior teníamos la configuración que se muestra en la imagen. Pero ahora vamos a cambiar la autenticación básica por la de token (cambia BasicAuthentication por **TokenAuthentication**) y mantendremos la obligatoriedad de autenticación.

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        #'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly',
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
    ]
}
```

¿EN QUÉ CONSISTE LA AUTENTICACIÓN CON TOKEN?

- Debemos añadir en las aplicaciones instaladas (archivo DRF → settings.py en el apartado INSTALLED_APPS) la aplicación que permite generar los tokens.

  'rest_framework.authtoken'

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'api',
]
```

La introducción de este elemento va a generar una nueva tabla para guardar los tokens que se generan. Por ello debemos realizar una migración. Recuerda que una migración es la forma en que se propagan los cambios que se realizan en los modelos (agregando un campo, eliminando un modelo, etc.) en el esquema de la base de datos.

Abre el terminal y con el entorno virtual activado realiza la migración.
¿Recuerdas cómo se hacía? Lo dejo en tus manos.

Comprueba que en la base de datos se haya creado la tabla **authtoken_token** (puedes instalarte en Visual Code la extensión Sqlite3 Viewer y abrir el fichero DRF → db.sqlite3)

Veamos todo ésto un poco más en profundidad.

1.- Abrimos la shell en un terminal (python manage.py shell)
2.- Importamos el objeto Token
3.- Importamos los usuarios del sistema

```
(.venv) PS D:\CURSO 22 - 23\Acceso a datos\API REST\API_AED> python manage.py shell
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from rest_framework.authtoken.models import Token
>>> from django.contrib.auth.models import User
>>>
```

Cuando queremos asignar un token a un usuario del sistema podemos hacerlo creando una instancia del objeto User y diciendo al sistema que genere el token para dicho usuario.

```
>>> my_user = User.objects.get(id=1)
>>> token = Token.objects.create(user=my_user)
>>> print (token)
3b7b2c2f6352605cadc6d4c8850df82578889262
>>>
```

El token mostrado validaría las peticiones de autenticación que va a pedir el sistema.
Comprueba en este momento el contenido de la tabla authtoken_token en la base de datos.
Si se intenta generar un token para un usuario que ya tiene se va a generar un error.

Supongamos que sea el propio usuario el que quiere generar su token o consultarlo:

- Importamos la librería views de rest_framework.authtoken en el fichero urls.py
- Añadimos otra ruta

urls.py > ...

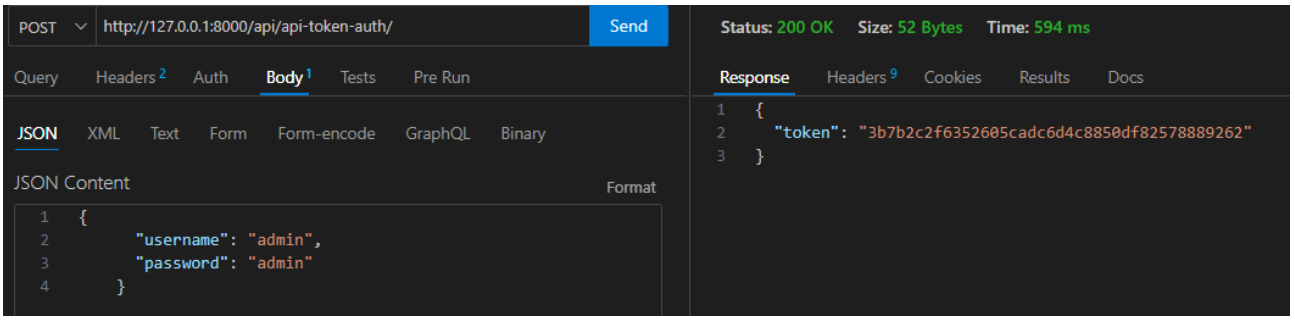
```python
from django.urls import include,  path
from rest_framework import routers
from rest_framework.authtoken import views
from .views import TaskView, TaskViewXML, TaskViewSet

router = routers.DefaultRouter()
router.register (r'tareas', TaskViewSet)


urlpatterns = [path ('tasks/', TaskView.as_view(), name = 'task_list'),
               path ('tasks/<int:pk>', TaskView.as_view(), name = 'una_task'),
               path ('tasks/completadas/', TaskView.as_view(), name = 'completadas'),
               path ('tasks_xml/', TaskViewXML.as_view(), name = 'tareas_xml'),
               path ('', include (router.urls)),
               path ('api-token-auth/', views.obtain_auth_token)
]
```

Cuando un usuario está autenticado y acceda a la ruta se le devolverá su token.

Vamos a comprobarlo desde Thunder. Arranca al servidor y accede a la ruta con una petición POST enviando en el body usuario y contraseña como se muestra en la imagen.



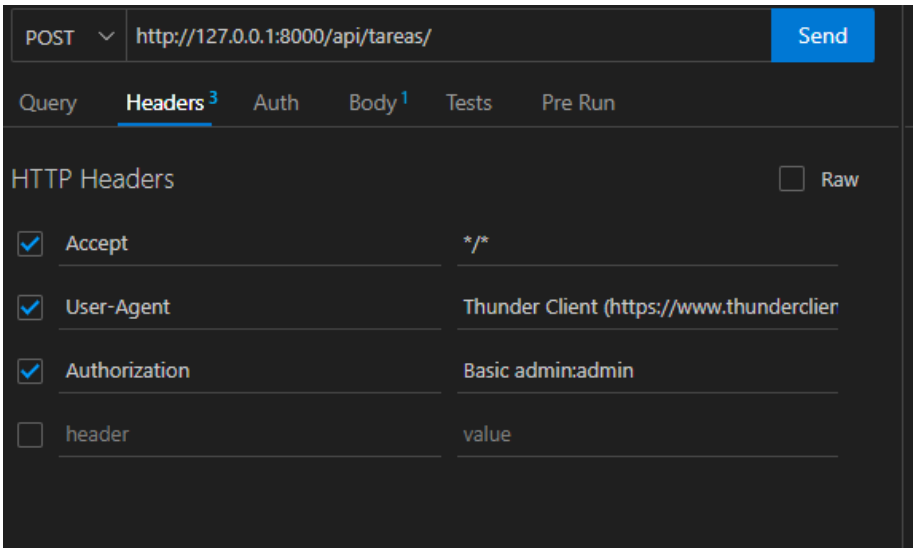
Comprueba la respuesta del servidor si haces la misma petición sin enviar usuario ni contraseña.

¿Cómo enviamos la información de autenticación?

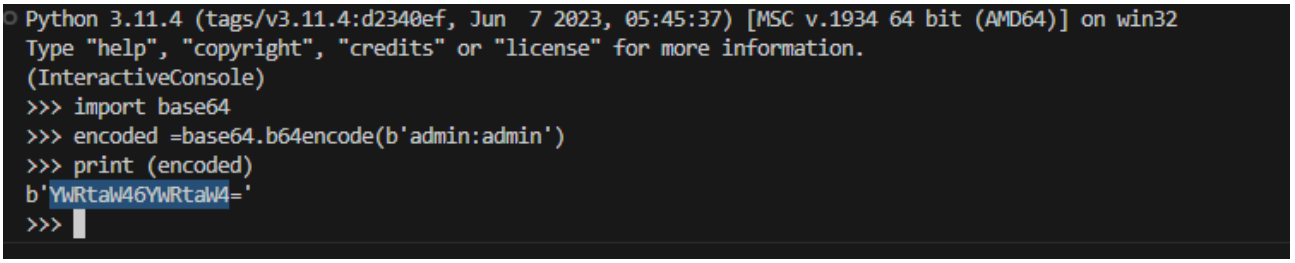
AUTENTICACIÓN BÁSICA: Debemos enviar la información de usuario (usuario y contraseña) a través de las cabeceras de los archivos que se envían. Para hacer la prueba usamos el cliente Thunder con el usuario admin y la contraseña que hayas añadido cuando se configuró Django.

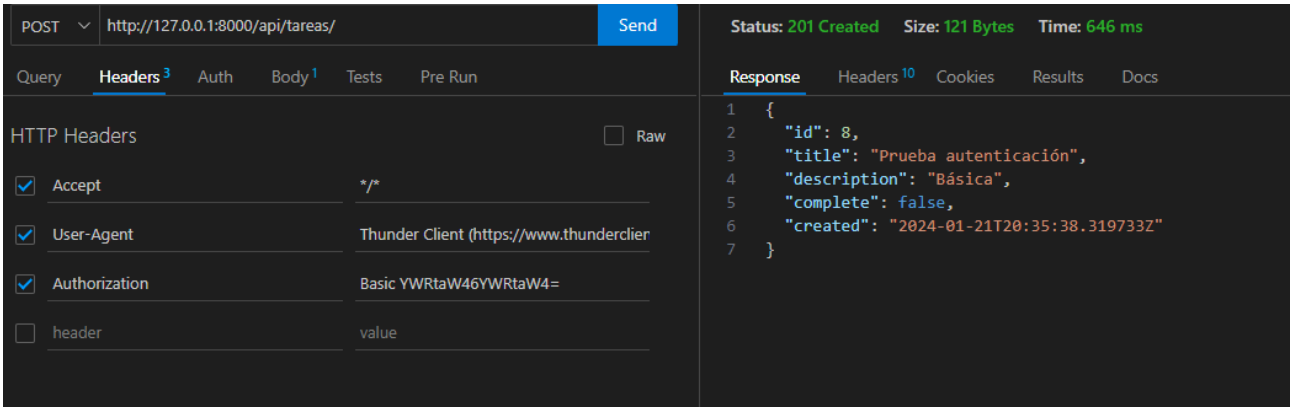
Modo 1: A través de la pestaña Headers

Al hacer la petición, como decíamos, debemos hacer uso de las cabeceras como se muestra en la imagen. Fíjate que debes añadir un nuevo elemento en dicha cabecera que será **"Authorization"** y con la información **Basic usuario:contraseña** (en mi caso admin:admin). La cuestión es que esa información (admin:admin) no se puede enviar así, como texto, sino codificado en base 64. Para hacer esto fíjate en la segunda imagen.

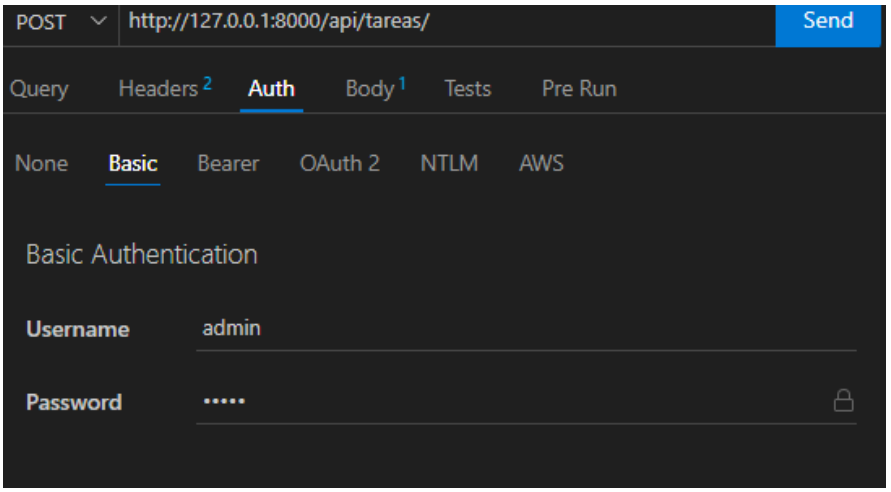


- Apaga el servidor y entra en la shell: *(tu_entorno) … : python manage.py shell*
- Importa el módulo base64
- codifica como se muestra en la imagen tu usuario y contraseña.
- Imprime la variable que contiene ya la codificación.
- Copia toda parte que se muestra sombreada en la imagen incluyendo el signo = (en la imagen no está sombreado)
- Copia el código como se muestra en la siguiente imagen.





Modo 2: Usando la pestaña Auth en combinación con Basic



Supongamos que incluso para las peticiones GET queremos que el cliente esté autenticado:

El fichero settings.py quedaría modificado de la siguiente manera:

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        #'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly',
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
    ]
}
```

Si ahora intentas hacer una petición GET sin enviar al servidor los parámetros de autenticación obtendrás un mensaje 401 (Unauthorized) del servidor.

AUTENTICACIÓN PERSONALIZADA (SÓLO EN ALGUNAS VISTAS)

Modificamos el fichero views.py añadiendo, por un lado, los import correspondientes a la autenticación que vamos a añadir en las vistas:

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.authentication import SessionAuthentication, BasicAuthentication
from rest_framework.response import Response
```

Y por otro lado añadimos en la vista que queramos configurar lo siguiente (en mi caso la vista TaskViewSet que es la que está funcionando por defecto).

```python
class TaskViewSet (viewsets.ModelViewSet):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer

    authentication_classes = [SessionAuthentication, BasicAuthentication]
    permission_classes = [IsAuthenticated]

    def get (self, request, format = None):
        content = {
            'user': str (request.user),
            'auth': str (request.auth),
        }
        return Response(content)
```

En este momento, aunque en el fichero setting estableciéramos como al principio que las consultas GET si se pueden ejecutar sin autenticación; estamos explicitando en la vista en el propio método GET que obligatoriamente debe autenticarse para acceder y por tanto el servidor volverá a enviar un error 401.