



El empleo
es de todos

Mintrabajo

Clase pruebas API REST python



@SENAcomunica

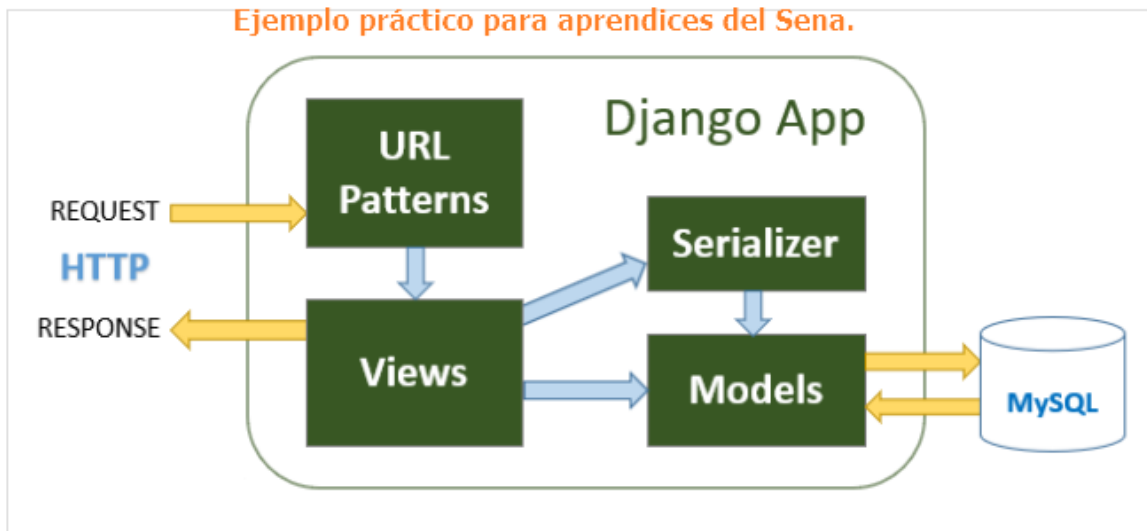
www.sena.edu.co

En este paso a paso, crearemos un CRUD Python 3 / Django con un ejemplo de MySQL que usa Django Rest Framework para construir api REST.



Arquitectura

Veamos el diagrama a continuación, muestra la arquitectura de la aplicación Django CRUD API REST con base de datos MySQL:



Explicación:

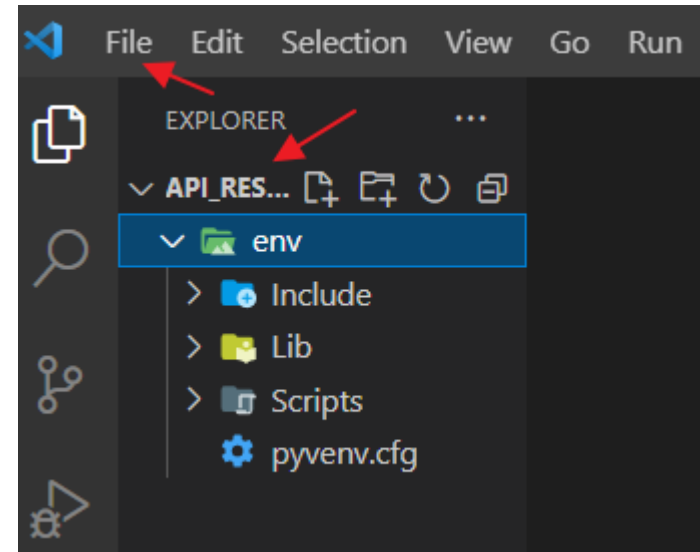
1. Las solicitudes HTTP coincidirán con los patrones de URL y se pasarán a las vistas(controladores).
2. Views procesa las solicitudes HTTP y devuelve respuestas HTTP (con la ayuda de Serializer).
3. Serializador serializa / deserializa objetos del modelo de datos.
4. Los modelos contienen campos y comportamientos esenciales para operaciones CRUD con base de datos MySQL

1. Creo una carpeta y uso el siguiente comando (según corresponda a su computadora) para crear un entorno virtual con el nombre env de su intérprete actual:

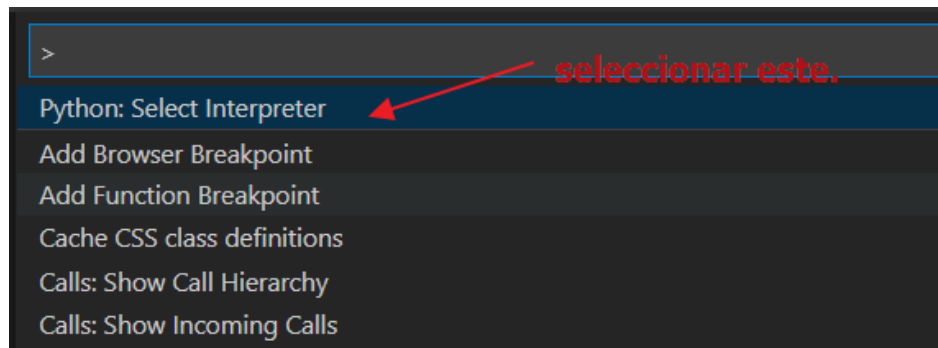


```
D:\1. sena_2021\Api_REST_Django>python -m venv env
```

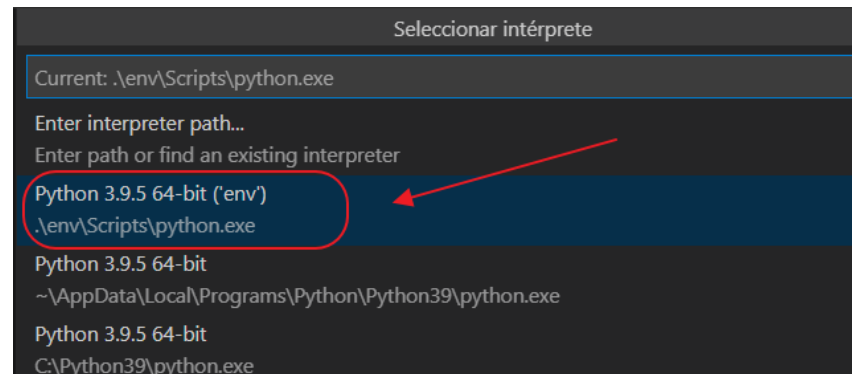
2. Abra la carpeta del proyecto en VS Code y usando el comando Archivo > Abrir carpeta .



3. En VS Code, abra la paleta de comandos (Ver > Paleta de comandos o (Ctrl + Shift + P)). Luego seleccione el comando Python: Seleccionar intérprete :



4. El comando presenta una lista de intérpretes disponibles que VS Code puede ubicar automáticamente (su lista variará; si no ve el intérprete deseado)



5. Actualice pip en el entorno virtual ejecutando el siguiente comando en VS Code Terminal:



```
PS D:\1. sena_2021\Api_REST_Django> python -m pip install --upgrade pip
```

6. Instale Django en el entorno virtual ejecutando el siguiente comando en VS Code Terminal:

```
PS D:\1. sena_2021\Api_REST_Django> python -m pip install django
```

7. El marco Django REST nos ayuda a crear servicios web RESTful de forma flexible. Para instalar este paquete, ejecute el comando:

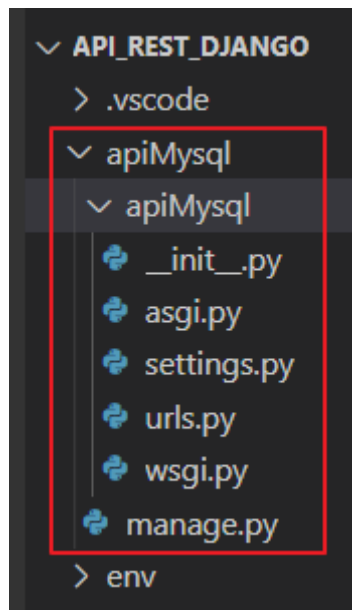
```
PS D:\1. sena_2021\Api_REST_Django> pip install djangorestframework
```

8. Creemos un nuevo proyecto de Django con el comando:

```
PS D:\1. sena_2021\Api_REST_Django> django-admin startproject apiMySQL
```



Cuando finaliza el proceso, puede ver un árbol de carpetas como este:



9. Ahora abrimos settings.py y agregamos el marco Django REST a la APPS INSTALLED aquí.

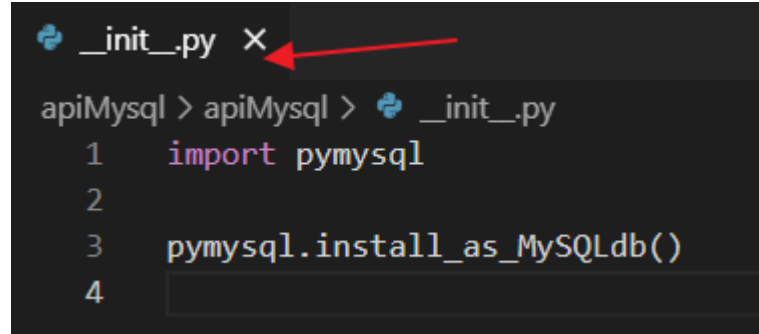
```
33  INSTALLED_APPS = [  
34      'django.contrib.admin',  
35      'django.contrib.auth',  
36      'django.contrib.contenttypes',  
37      'django.contrib.sessions',  
38      'django.contrib.messages',  
39      'django.contrib.staticfiles',  
40      'rest_framework',  
41  ]
```


10. Necesitamos un cliente MySQL para trabajar con la base de datos MySQL.

Ejecutar el comando para instalarlo: pip install pymysql.

```
PS D:\1. sena_2021\Api_REST_Django> pip install pymysql
```

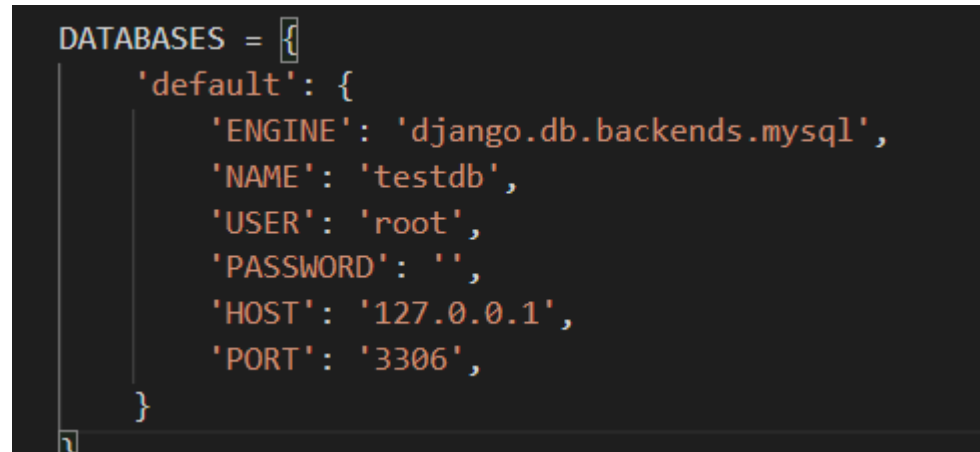

11. Luego abra `__init__.py` y escriba el siguiente código para importar pymysql a nuestro proyecto Django:

A screenshot of a code editor window titled "__init__.py". The editor shows the following code:

```
apiMysql > apiMysql >  __init__.py
1  import pymysql
2
3  pymysql.install_as_MySQLdb()
4
```

A red arrow points to the tab title "__init__.py".

12. También necesitamos configurar el motor de base de datos MySQL. Así que abre `settings.py` y cambia la declaración de `DATABASES`:

A screenshot of a code editor showing the `DATABASES` configuration in `settings.py`. The code is as follows:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'testdb',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

13. Ejecute los siguientes comandos para crear un nuevo modulo de la aplicación Django :



```
PS D:\1. sena_2021\Api_REST_Django> cd apiMysql
PS D:\1. sena_2021\Api_REST_Django\apiMysql> python manage.py startapp tutoriales
```

14, Ahora abra tutoriales / apps.py , puede ver la clase TutorialesConfig (subclase de django.apps.AppConfig).

Esto representa la aplicación Django que acabamos de crear con su configuración:

```
apps.py  X
apiMysql > tutoriales > apps.py
1  from django.apps import AppConfig
2
3
4  class TutorialesConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'tutoriales'
```

15. No olvide agregar esta aplicación a la APPs INSTALLED_ en settings.py :



```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'tutoriales.apps.TutorialesConfig',  
]
```

16. Necesitamos permitir solicitudes a nuestra aplicación Django desde otros orígenes. En este ejemplo, configuraremos CORS para aceptar solicitudes de localhost:8081

Primero, instale la biblioteca django-cors-headers :

```
PS D:\1. sena_2021\Api_REST_Django> cd apiMySQL  
PS D:\1. sena_2021\Api_REST_Django\apiMySQL> pip install django-cors-headers
```

17. En settings.py , agregue la configuración para CORS:

18. También debe agregar una clase de middleware para escuchar las respuestas:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'tutoriales.apps.TutorialesConfig',  
    'corsheaders',  
]
```

```
MIDDLEWARE = [  
    'corsheaders.middleware.CorsMiddleware',  
    'django.middleware.common.CommonMiddleware',  
  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

Nota: CorsMiddleware debe colocarse lo más alto posible, especialmente antes de cualquier middleware que pueda generar respuestas como CommonMiddleware.

19. A continuación, en el archivo de settings.py configure CORS_ORIGIN_ALLOW_ALL y agregue el host a CORS_ORIGIN_WHITELIST :

```
CORS_ORIGIN_ALLOW_ALL = False
CORS_ORIGIN_WHITELIST = (
    'http://localhost:8081',
)
```

CORS_ORIGIN_ALLOW_ALL : Si es True, se aceptarán todos los orígenes (no use la lista blanca a continuación). Por defecto es False.

CORS_ORIGIN_WHITELIST : Lista de orígenes que están autorizados para realizar solicitudes HTTP entre sitios. Por defecto es [].

20. Abra tutoriales / models.py , agregue la Clase Tutorial como subclase de django.db.models.Model.

Hay 3 campos: título , descripción , publicado .



```
models.py X
apiMySQL > tutoriales > models.py
1  from django.db import models
2
3  class Tutorial(models.Model):
4      titulo = models.CharField(max_length=70, blank=False, default='')
5      descripcion = models.CharField(max_length=200, blank=False, default='')
6      publicado = models.BooleanField(default=False)
7
```

Cada campo se especifica como un atributo de clase y cada atributo se asigna a una columna de la base de datos.

El campo de identificación se agrega automáticamente.

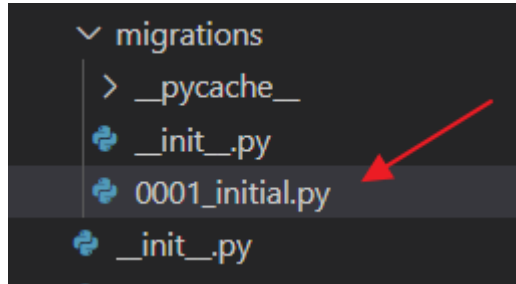
21. Migrar el modelo de datos a la base de datos ejecutando la secuencia de comandos de Python: `python manage.py makemigrations tutoriales`.



```
PS D:\1. sena_2021\Api_REST_Django> cd apiMySQL  
PS D:\1. sena_2021\Api_REST_Django\apiMySQL> python manage.py makemigrations tutoriales
```

Actualice el espacio de trabajo, puede ver el nuevo archivo `tutoriales / migrations / 0001_initial.py`.

Incluye código para crear el modelo Tutorial de datos:





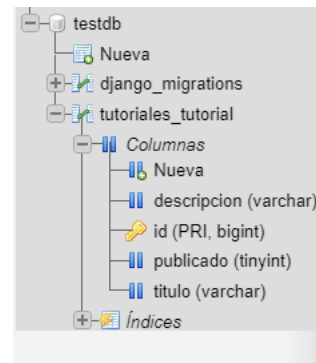
El código generado define la clase Migration (subclase de django.db.migrations.Migration). Tiene gran variedad de operaciones que contiene la operación de creación de tabla de modelos : migrations.CreateModel().

La llamada a esto creará un nuevo modelo en el historial del proyecto y una tabla correspondiente en la base de datos para que coincida.

22. Para aplicar la migración generada anteriormente, ejecute la siguiente secuencia de comandos, teniendo en cuenta de encender su servicio de mysql:

```
PS D:\1. sena_2021\Api_REST_Django> cd apiMySQL
PS D:\1. sena_2021\Api_REST_Django\apiMySQL> python manage.py migrate tutoriales
```

Service	Module	PID(s)	Port(s)	Actions	
	Apache	18500 14544	80, 443	Stop	Admin
	MySQL	11228	3306	Stop	Admin



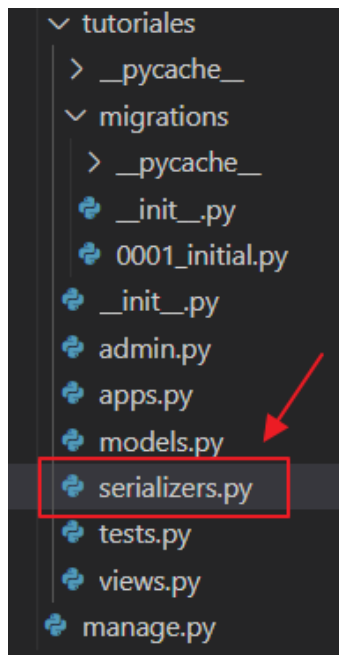
23. Crear clase de serializador para el modelo de datos

Creemos una clase que gestionará la serialización y deserialización desde JSON.



Hereda de la superclase `rest_framework.serializers.ModelSerializer` que automáticamente llena un conjunto de campos y por defecto validators.

Necesitamos especificar la clase de modelo aquí:



```
serializers.py X
apiMySQL > tutoriales > serializers.py
1  from rest_framework import serializers
2  from tutoriales.models import Tutorial
3
4
5  class TutorialSerializer(serializers.ModelSerializer):
6
7      class Meta:
8          model = Tutorial
9          fields = ('id',
10                  'titulo',
11                  'descripcion',
12                  'publicado')
```

Rutas :Cuando un cliente envía una solicitud para un punto final mediante una solicitud HTTP (GET, POST, PUT, DELETE), debemos determinar cómo responderá el servidor definiendo las rutas.

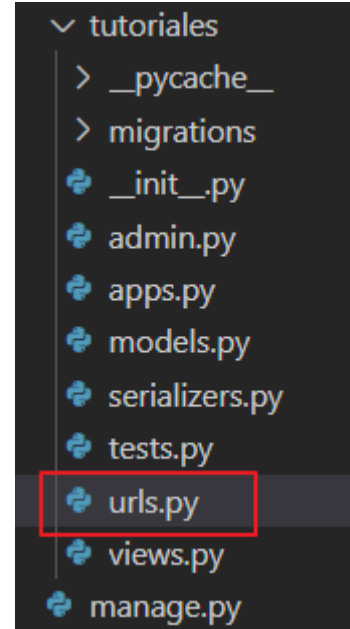
Estas son nuestras rutas:

/api/tutoriales: OBTENER, PUBLICAR, ELIMINAR

/api/tutoriales/:id: OBTENER, PONER, ELIMINAR

/api/tutoriales/publicado: OBTENER

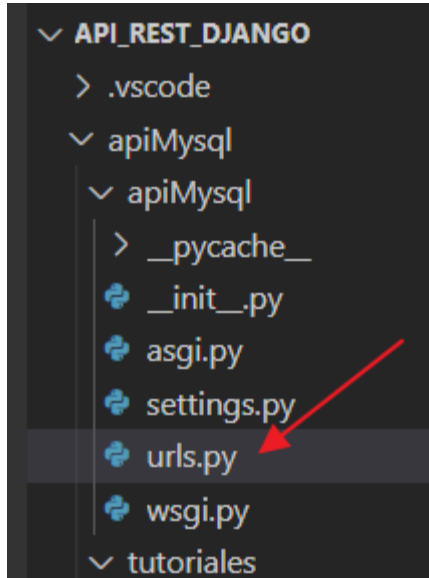
24. Dentro de la aplicación de tutoriales cree un archivo de urls.py que contenga rutas.



```
urls.py ×  
apiMysql > tutoriales > urls.py  
1  from django.conf.urls import url  
2  from tutoriales import views  
3  
4  urlpatterns = [  
5      url(r'^api/tutoriales$', views.tutorial_lista),  
6      url(r'^api/tutoriales/(?P<pk>[0-9]+)$', views.tutorial_detalle),  
7      url(r'^api/tutoriales/publicado$', views.tutorial_list_publicado)  
8  ]  
9
```

No olvide incluir estos patrones de URL en el archivo urls.py de URL raíz.

25. Abra apiMysql / urls.py y modifique el contenido con el siguiente código.



```
from django.contrib import admin
from django.urls import path
from django.conf.urls import url, include

urlpatterns = [
    path('admin/', admin.site.urls),
    url(r'^', include('tutoriales.urls')),
]
```

Escribir views(controladores) de API

Vamos a crear estas funciones de API para operaciones CRUD:

- tutorial_lista(): OBTENER la lista de tutoriales, PUBLICAR un nuevo tutorial, ELIMINAR todos los tutoriales.
- tutorial_detalle(): OBTENER / PUT / DELETE tutorial por 'id'
- tutorial_list_publicado(): OBTENER todos los tutoriales publicados

26. Abra tutoriales / views.py y escriba el siguiente código:

```
views.py x
apiMySQL > tutoriales > views.py
1  from django.shortcuts import render
2
3  from django.http.response import JsonResponse
4  from rest_framework.parsers import JSONParser
5  from rest_framework import status
6
7  from tutoriales.models import Tutorial
8  from tutoriales.serializers import TutorialSerializer
9  from rest_framework.decorators import api_view
10
11
12  @api_view(['GET', 'POST', 'DELETE'])
13  def tutorial_lista(request):
14      # obtiene lista de tutoriales, POST a un nuevo tutorial, DELETE todos los tutoriales
15      if request.method == 'GET':
16          tutoriales = Tutorial.objects.all()
17
18          titulo = request.GET.get('titulo', None)
19          if titulo is not None:
20              tutoriales = tutoriales.filter(titulo__icontains=titulo)
21
22          tutoriales_serializer = TutorialSerializer(tutoriales, many=True)
23          return JsonResponse(tutoriales_serializer.data, safe=False)
24
```

parte 1 del archivo views.py
controlador

parte 2 archivo views.py

```
25 elif request.method == 'POST':
26     tutorial_data = JSONParser().parse(request)
27     tutorial_serializer = TutorialSerializer(data=tutorial_data)
28     if tutorial_serializer.is_valid():
29         tutorial_serializer.save()
30         return JsonResponse(tutorial_serializer.data, status=status.HTTP_201_CREATED)
31     return JsonResponse(tutorial_serializer.errors, status=status.HTTP_400_BAD_REQUEST)
32
33 elif request.method == 'DELETE':
34     count = Tutorial.objects.all().delete()
35     return JsonResponse({'message': '{} el Tutorial eliminado Satisfactoriamente!'.format(
36         count[0])}, status=status.HTTP_204_NO_CONTENT)
37
38 @api_view(['GET', 'PUT', 'DELETE'])
39 def tutorial_detalle(request, pk):
40     # encuentra tutorial por pk (id)
41     try:
42         tutorial = Tutorial.objects.get(pk=pk)
43     except Tutorial.DoesNotExist:
44         return JsonResponse({'message': 'Tutorial no existe!'}, status=status.
45             HTTP_404_NOT_FOUND)
```

parte 3 de views.py

```
44     if request.method == 'GET':
45         tutorial_serializer = TutorialSerializer(tutorial)
46         return JsonResponse(tutorial_serializer.data)
47     elif request.method == 'PUT':
48         tutorial_data = JSONParser().parse(request)
49         tutorial_serializer = TutorialSerializer(tutorial, data=tutorial_data)
50         if tutorial_serializer.is_valid():
51             tutorial_serializer.save()
52             return JsonResponse(tutorial_serializer.data)
53
54     elif request.method == 'DELETE':
55         tutorial.delete()
56         return JsonResponse({'message': 'Tutorial ha sido borrado!'}, status=status.
57             HTTP_204_NO_CONTENT)
58 # GET / PUT / DELETE tutorial
59
60
61 @api_view(['GET'])
62 def tutorial_list_publicado(request):
63     # obtiene listado de tutoriales publicados
64     tutoriales = Tutorial.objects.filter(publicado=True)
65
```



```
66     if request.method == 'GET':           parte 4
67         tutoriales_serializer = TutorialSerializer(tutoriales, many=True)
68         return JsonResponse(tutoriales_serializer.data, safe=False)
69
```



27. Ejecutar nuestro proyecto Django con el comando: `python manage.py runserver 8080`.

La consola muestra:

```
PS D:\1. sena_2021\Api_REST_Django\apiMySQL> python manage.py runserver 8080
Watching for file changes with StatReloader
Performing system checks...
```

```
System check identified no issues (0 silenced).
June 05, 2021 - 13:23:02
Django version 3.2.4, using settings 'apiMySQL.settings'
Starting development server at http://127.0.0.1:8080/
Quit the server with CTRL-BREAK.
```

← → ↻ 🏠 ⓘ 127.0.0.1:8080/api/tutoriales 1

Aplicaciones Dharma Consulting... Dharma Consulting... 3. Técnicas para Ide... Certif

```
[
  {
    id: 1,
    titulo: "Tutorial de Django",
    descripcion: "Contiene una descripción del paso a paso de Django",
    publicado: true
  },
  {
    id: 2,
    titulo: "Tutorial de Java",
    descripcion: "Contiene ejercicios prácticos de java",
    publicado: false
  }
]
```

← → ↻ 🏠 ⓘ 127.0.0.1:8080/api/tutoriales/1 2

Aplicaciones Dharma Consulting... Dharma Consulting... 3. Técnicas para Ide...

```
{
  id: 1,
  titulo: "Tutorial de Django",
  descripcion: "Contiene una descripción del paso a paso de Django",
  publicado: true
}
```

← → ↻ 🏠 ⓘ 127.0.0.1:8080/api/tutoriales/publicado

Aplicaciones Dharma Consulting... Dharma Consulting... 3. Técnicas para Ide...

3

```
[
  {
    id: 1,
    titulo: "Tutorial de Django",
    descripcion: "Contiene una descripción del paso a paso de Django",
    publicado: true
  }
]
```

Vamos a descargar el cliente para poder probar:

<https://www.postman.com/product/rest-client/>

The Postman app

The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

 **Download the App**

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

Version 8.5.1 | [Release Notes](#) | [Product Roadmap](#)



Search Postman

POST http://127.0.0.1:8080/api/tutoriales

1

No Environment

Save

Send

POST http://127.0.0.1:8080/api/tutoriales

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL Text

```
1 {
2   "id": 3,
3   "titulo": "Tutorial de Ensayos",
4   "descripcion": "Contiene las pruebas de Postman para aprendices",
5   "publicado": true
6 }
```

Ingreso el Json

body Cookies Headers (9) Test Results

Status: 201 Created Time: 32 ms Size: 430 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "titulo": "Tutorial de Ensayos",
4   "descripcion": "Contiene las pruebas de Postman para aprendices",
5   "publicado": true
6 }
```

Seguido revise en la base de datos.

+ Opciones

				id	titulo	descripcion	publicado
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	Tutorial de Django	Contiene una descripción del paso a paso de Django	1
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	Tutorial de Java	Contiene ejercicios prácticos de java	0
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	Tutorial de Ensayos	Contiene las pruebas de Postman para aprendices	1

Recuperar todos los tutoriales usando GET /



Overview GET http://127.0.0.1:80... + ... No Environment

http://127.0.0.1:8080/api/tutoriales Save

GET http://127.0.0.1:8080/api/tutoriales Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL Text

1

Body Cookies Headers (9) Test Results Status: 200 OK Time: 20 ms Size: 685 B Save Response

Pretty Raw Preview Visualize JSON

```
2 {
3   "id": 1,
4   "titulo": "Tutorial de Django",
5   "descripcion": "Contiene una descripción del paso a paso de Django",
6   "publicado": true
7 },
8 {
9   "id": 2,
10  "titulo": "Tutorial de Java",
11  "descripcion": "Contiene ejercicios prácticos de java",
12  "publicado": false
13 },
14 {
15   "id": 3,
16   "titulo": "Tutorial de Ensayos",
```

información de la base de datos.

Actualizar un tutorial usando PUT /tutoriales/:id



Overview PUT http://127.0.0.1:80... No Environment

http://127.0.0.1:8080/api/tutoriales/3

PUT http://127.0.0.1:8080/api/tutoriales/3

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL Text

```
1 {
2   "titulo": "Tutorial de Angular",
3   "descripcion": "Contiene actualización angular",
4   "publicado": true
5 }
```

Nueva data

Body Cookies Headers (9) Test Results Status: 200 OK Time: 31 ms Size: 413 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "titulo": "Tutorial de Angular",
4   "descripcion": "Contiene actualización angular",
5   "publicado": true
6 }
```


Recuperar un solo tutorial por id usando GET

A screenshot of a REST client interface. At the top, a tab shows "GET http://127.0.0.1:80...". Below this, the URL "http://127.0.0.1:8080/api/tutoriales/2" is entered in the address bar. A red box highlights the URL, and a red arrow points from a purple circle with the number "4" to the "Send" button. The "Send" button is blue with white text. Below the address bar, there are tabs for "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is selected, and it shows "none" as the content type. Below the tabs, a message states "This request does not have a body". At the bottom, the "Body" tab is selected, showing the response in "Pretty" format. The response is a JSON object: {"id": 2, "titulo": "Tutorial de Java", "descripcion": "Contiene ejercicios prácticos de java", "publicado": false}. A red box highlights the JSON response. The status bar at the bottom right shows "Status: 200 OK", "Time: 39 ms", "Size: 417 B", and a "Save Response" button.

Overview GET http://127.0.0.1:80... + ... No Environment

http://127.0.0.1:8080/api/tutoriales/2 Save

GET http://127.0.0.1:8080/api/tutoriales/2 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (9) Test Results Status: 200 OK Time: 39 ms Size: 417 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "titulo": "Tutorial de Java",
4   "descripcion": "Contiene ejercicios prácticos de java",
5   "publicado": false
6 }
```

Encuentre todos los tutoriales cuyo título contenga 'ja': GET
/tutoriales?titulo=ja

A screenshot of a REST client interface. At the top, there's a tab labeled "GET http://127.0.0.1:8080...". Below it, the URL "http://127.0.0.1:8080/api/tutoriales?titulo=ja" is entered. A red box highlights the URL field, and a red arrow points from it to a blue "Send" button. Below the URL bar, there are tabs for "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is selected, and it shows "This request does not have a body". Below the request tabs, there are radio buttons for "none", "form-data", "x-www-form-urlencoded", "raw", "binary", and "GraphQL". The "none" option is selected. At the bottom, the response is displayed in the "Body" tab, showing a JSON array of two tutorial objects. The response status is "200 OK", time is "27 ms", and size is "556 B".

Overview GET http://127.0.0.1:8080... No Environment

http://127.0.0.1:8080/api/tutoriales?titulo=ja Save

GET http://127.0.0.1:8080/api/tutoriales?titulo=ja Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (9) Test Results Status: 200 OK Time: 27 ms Size: 556 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "titulo": "Tutorial de Django",
5     "descripcion": "Contiene una descripción del paso a paso de Django",
6     "publicado": true
7   },
8   {
9     "id": 2,
10    "titulo": "Tutorial de Java",
11    "descripcion": "Contiene ejercicios prácticos de java",
12    "publicado": false
13  }
14 }
```

Encuentra todos los tutoriales publicados usando GET /tutoriales/publicado



Overview GET http://127.0.0.1:80... + ... No Environment

http://127.0.0.1:8080/api/tutoriales/publicado Save

GET http://127.0.0.1:8080/api/tutoriales/publicado Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (9) Test Results Status: 200 OK Time: 32 ms Size: 538 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "titulo": "Tutorial de Django",
5     "descripcion": "Contiene una descripción del paso a paso de Django",
6     "publicado": true
7   },
8   {
9     "id": 3,
10    "titulo": "Tutorial de Angular",
11    "descripcion": "Contiene actualización angular",
12    "publicado": true
13  }
14 }
```

Eliminar un tutorial usando DELETE /tutoriales/:id



Overview **DEL** http://127.0.0.1:80... + ... 7 No Environment ▼

http://127.0.0.1:8080/api/tutoriales/2 Save ▼ ✎ 💬

DELETE ▼ http://127.0.0.1:8080/api/tutoriales/2 Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (9) Test Results 🌐 Status: 204 No Content Time: 37 ms Size: 344 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ ☰

```
1  [
2    "message": "Tutorial ha sido borrado!"
3  ]
```

Eliminar todos los tutoriales usando DELETE



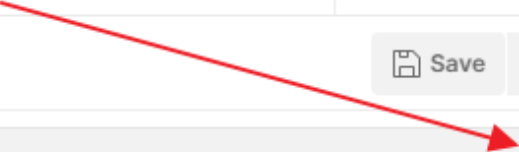
Overview **DEL** http://127.0.0.1:80... + ... No Environment

http://127.0.0.1:8080/api/tutoriales Save

DELETE http://127.0.0.1:8080/api/tutoriales Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL





GRACIAS

Línea de atención al ciudadano: 018000 910270
Línea de atención al empresario: 018000 910682



@SENAcomunica

www.sena.edu.co