Documentación API GestorPass

Configuración del Proyecto

Proyecto Django: Proyecto_Gestorpass

El proyecto Django **`Proyecto_Gestorpass`** es el entorno principal que contiene nuestra aplicación **`gestorpass`.**

Estructura del Proyecto

- **`Proyecto_Gestorpass/`**
  - **`settings.py`:** Configuraciones del proyecto Django, incluyendo la aplicación **`gestorpass`,** middleware, bases de datos, etc.
  - **`urls.py`:** Define las URLs del proyecto para incluir las URLs de la aplicación **`gestorpass`.**

Aplicación Django: gestorpass

La aplicación **`gestorpass`** es donde reside la lógica principal de la API.

Estructura de la Aplicación

- **`gestorpass/`**
  - **`models.py`:** Define los modelos de Django, que son las estructuras de datos para `Usuario`.
  - **`serializers.py`:** Contiene los serializadores para convertir los modelos en JSON y viceversa.
  - `class UsuarioSerializer(serializers.ModelSerializer)`**: Aquí estás definiendo una nueva clase** `UsuarioSerializer`**, que hereda de** `serializers.ModelSerializer`**. Esta herencia significa que tu serializador está diseñado específicamente para trabajar con modelos**

de Django, simplificando la conversión de instancias del modelo `Usuario` **a JSON y viceversa.**

- `serializers.ModelSerializer` **se encarga de gran parte de la lógica necesaria para convertir los campos del modelo en tipos de datos adecuados para la serialización.**
- **`views.py`**: Define las vistas de Django para interactuar con los modelos.
- **`urls.py`:** Define las URLs específicas para la aplicación `gestorpass`.

Modelos

Usuario

El modelo **`Usuario`** representa a los usuarios en la base de datos.

```
from django.db import models
'''
Esta línea importa el módulo models del paquete django.db. Django utiliza este
módulo para manejar todo lo relacionado con las bases de datos, incluyendo
definir y trabajar con modelos de datos.
'''
class Usuario(models.Model):
'''
Esta línea define una nueva clase llamada Usuario, la cual hereda de
models.Model. En Django, un modelo representa una tabla de base de datos. Cada
atributo del modelo representará un campo en la base de datos. En este caso,
Usuario será una nueva tabla en tu base de datos.
'''
    nickname = models.CharField(max_length=255)
    contraseña = models.CharField(max_length=255)
'''
Aquí se define el campo llamado nickname en el modelo Usuario.
models.CharField es un tipo de campo que representa una cadena de caracteres
(texto). Es ideal para nombres, títulos, o cualquier otro texto corto.
max_length=255 como su nombre indica es el máximo de carácteres que puede tener
una cadena de caracteres.

'''
    def __str__(self):
        return self.nickname

'''
```

def __str__(self): En los modelos de Django, se utiliza para definir cómo se debe representar una instancia de un modelo (un registro en la base de datos) como una cadena de texto.

return self.nickname: Éste método devuelve la instancia del campo nickname. Esto es útil para fines de visualización, como en el sitio de administración de Django, o en éste caso Postman

'''

Serializadores:

El archivo serializers.py en Django, particularmente cuando se trabaja con Django REST Framework (DRF), se utiliza para convertir objetos de modelo complejos (como los que interactúan con la base de datos) en tipos de datos que se pueden renderizar fácilmente en formatos como JSON. Esto es crucial para la creación de APIs, ya que permite una fácil serialización y deserialización de datos de y hacia la base de datos.

En resumen El UsuarioSerializer convierte instancias del modelo Usuario a JSON y viceversa.

UsuarioSerializer

serializers.py

```
from rest_framework import serializers
from .models import Usuario
'''
```

from rest_framework import serializers: Esto importa el módulo serializers de Django REST Framework, que es necesario para crear tus propios serializadores.
from .models import Usuario: Esto importa la clase Usuario de tu archivo models.py. Esto se utiliza para indicarle al serializador qué modelo debe utilizar.

“”

class UsuarioSerializer(serializers.ModelSerializer):

“”

Aquí estás definiendo una nueva clase UsuarioSerializer, que hereda de serializers.ModelSerializer. Esta herencia significa que el serializador está diseñado específicamente para trabajar con modelos de Django, es decir, de instancias del modelo Usuario a JSON y viceversa.
serializers.ModelSerializer se encarga de gran parte de la lógica necesaria para convertir los campos del modelo en tipos de datos adecuados para la serialización.

“”

```
    class Meta:
'''
class Meta: Esta clase interna llamada Meta se utiliza en Django para
proporcionar metadatos a la clase UsuarioSerializer. Los metadatos son
simplemente datos sobre datos; en este caso, estamos diciendo a Django más
sobre cómo debe comportarse el UsuarioSerializer.
'''
        model = Usuario
        fields = '__all__'
'''
```

model = Usuario: Esto le indica al serializador que el modelo asociado a este serializador es el modelo Usuario. Esto es esencial porque le dice a Django REST Framework de qué modelo debe serializar y deserializar datos.

fields = '__all__': Esto especifica los campos del modelo que deben incluirse en la serialización. __all__ significa que se incluirán todos los campos del modelo Usuario. Si solo quisieramos incluir algunos campos, podríamos reemplazar __all__ por una lista de los nombres de los campos que queramos incluir.

```
'''
```

Vistas

El archivo views.py en un proyecto Django, especialmente cuando se utiliza Django REST Framework (DRF), es donde defines las vistas que manejarán las solicitudes a tu API. En el código que has proporcionado, se define una vista basada en clases que se encarga de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para el modelo Usuario

UsuarioViewSet

`UsuarioViewSet` proporciona operaciones CRUD para el modelo **`Usuario`.**

`views.py`

```
from rest_framework import viewsets
from .models import Usuario
from .serializers import UsuarioSerializer

class UsuarioViewSet(viewsets.ModelViewSet):
 queryset = Usuario.objects.all()
 serializer_class = UsuarioSerializer
```

NOTAS Y APUNTES PERSONALES DE VIEWS.PY:

Importaciones:

from rest_framework import viewsets: Importa viewsets de Django REST Framework, que es una forma de organizar la lógica de las vistas.
from .models import Usuario: Importa el modelo Usuario de tu archivo models.py.
from .serializers import UsuarioSerializer: Importa la clase UsuarioSerializer de tu archivo serializers.py.
Clase UsuarioViewSet:

class UsuarioViewSet(viewsets.ModelViewSet): Define una clase UsuarioViewSet que hereda de viewsets.ModelViewSet. ModelViewSet automáticamente proporciona implementaciones para acciones CRUD en tu modelo.
queryset = Usuario.objects.all(): Define el queryset que el ViewSet utilizará. Aquí, estás diciendo que la vista debe operar en todos los objetos Usuario disponibles.
serializer_class = UsuarioSerializer: Indica que UsuarioSerializer se usará para serializar y deserializar instancias del modelo Usuario.
Métodos CRUD:

create, update, destroy: Estos métodos están sobrescritos de la clase base ModelViewSet. Aquí, puedes agregar lógica personalizada para cada acción CRUD.
*create(self, request, *args, **kwargs): Método para crear un nuevo usuario. Llama a super().create, que ejecuta la lógica de creación predeterminada, pero puedes agregar lógica adicional antes o después de esta llamada.*
*update(self, request, *args, **kwargs): Método para actualizar un usuario existente. De manera similar al método create, puedes personalizar la lógica de actualización.*

*destroy(self, request, *args, **kwargs): Método para eliminar un usuario. Nuevamente, puedes personalizar la lógica de eliminación.*
En resumen, UsuarioViewSet es una clase que encapsula la lógica para realizar operaciones CRUD en el modelo Usuario. Utiliza el serializador UsuarioSerializer para la entrada y salida de datos, y define un conjunto de consultas que operan en todos los objetos Usuario. Con este ViewSet, Django REST Framework puede manejar automáticamente muchas de las tareas comunes

asociadas con la creación de una API, como analizar JSON, validar datos y generar respuestas apropiadas.

URLs

URL Patterns

Las URLs conectan las solicitudes entrantes con las vistas adecuadas.

`urls.py` en `Proyecto_Gestorpass`

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

urlpatterns = [
 path('admin/', admin.site.urls),
 path('api/', include('gestorpass.urls')),
]
```

NOTAS Y APUNTES:

importaciones:

from django.urls import path: Importa la función path que se utiliza para definir las rutas URL.
from .views import UsuarioViewSet: Importa UsuarioViewSet de tu archivo views.py.

Definición de urlpatterns:
urlpatterns es una lista de patrones URL. Cada patrón asocia una URL con una vista.
Patrones URL:

path('usuarios/', UsuarioViewSet.as_view({'get': 'list', 'post': 'create'}), name='usuario-list'): Define una ruta URL para operaciones de lista y creación de usuarios.
'usuarios/': Es el patrón URL. Por ejemplo, si tu dominio es example.com, acceder a example.com/usuarios/ activará esta ruta.

UsuarioViewSet.as_view({'get': 'list', 'post': 'create'}): Mapea los métodos HTTP a las acciones del ViewSet. Aquí, GET mapea a la acción list (obtener una lista de usuarios) y POST mapea a create (crear un nuevo usuario).
name='usuario-list': Es el nombre de esta ruta URL, que puede usarse para referenciarla internamente en Django.

path('usuarios/<int:pk>/', UsuarioViewSet.as_view({'get': 'retrieve', 'put': 'update', 'delete': 'destroy'}), name='usuario-detail'): Define una ruta URL para operaciones de recuperación, actualización y eliminación de un usuario específico.

'usuarios/<int:pk>/': El segmento <int:pk> es un parámetro dinámico que Django interpretará como un entero y pasará como argumento pk (primary key) a la vista. Permite especificar un usuario en particular.

UsuarioViewSet.as_view({'get': 'retrieve', 'put': 'update', 'delete': 'destroy'}): Asigna métodos HTTP a acciones específicas para un usuario individual: GET para retrieve (obtener un usuario específico), PUT para update (actualizar un usuario), y DELETE para destroy (eliminar un usuario).

name='usuario-detail': Es el nombre de esta ruta URL.

urls.py en gestorpass

```
from django.urls import path, include
from rest_framework import routers
from .views import UsuarioViewSet

router = routers.DefaultRouter()
router.register(r'usuarios', UsuarioViewSet)

urlpatterns = [
 path('', include(router.urls)),
]
```

Interacción con la API usando Postman

Para interactuar con la API, se utiliza Postman, una herramienta que facilita el envío de solicitudes HTTP a la API y la visualización de las respuestas.

Pasos para usar Postman:

Abrir Postman y crear una nueva solicitud.
Configurar la solicitud:

- Método: GET/POST/PUT/DELETE, dependiendo de la acción deseada.
- URL: **`http://localhost:8000/api/usuarios/`** para listar o crear usuarios, **`http://localhost:8000/api/usuarios/<id>/`** para operaciones específicas del usuario.

Para POST/PUT:

- Seleccionamos **`Body -> raw -> JSON.`**
- Introducimos los datos del usuario, por ejemplo: **`{"nickname": "nuevo_usuario", "contraseña": "nueva_contraseña"}.`**

Enviar la solicitud y revisar la respuesta.

Notas Finales: