Django REST Framework

Cruz Arredondo Jose David. Monday November 21, 2023

***Summary*—Django REST framework is a customizable toolkit that makes it easy to build APIs. It's based on Django's class-based views, so it can be an excellent choice if you're familiar with Django. Developers can authenticate people on their web app with OAuth. Provide both ORM and non-ORM serialization.**

I. INTRODUCTION

Django REST framework is a powerful and flexible toolkit for building Web APIs. It is based on Django's class-based views, making it an excellent choice if you're familiar with Django. REST framework provides both ORM and non-ORM serialization, and developers can authenticate people on their web app with OAuth.

II. USES OF REST FRAMEWORK

RESTful APIs are widely used in modern web development. They are used to create web services that follow the REST architectural style. RESTful APIs are lightweight, scalable, and easy to maintain. They are used by many popular websites and applications such as Twitter, Facebook, and Google Maps. Also, is used to provide access to data and functionality to other applications. They are used to create mobile applications, web applications, and other software that needs to interact with a web service. RESTful APIs are also used to create mashups, which are web applications that combine data from multiple sources.

In addition to the above, RESTful APIs are used to create microservices, which are small, independent services that work together to create a larger application. Microservices are used to create scalable and resilient applications that can handle large amounts of traffic 2.

III. CHARACTERISTICS

Client-server architecture: The client and the server are separate entities that communicate through a standardized interface, such as HTTP. The client is responsible for initiating requests and handling responses, while the server is responsible for processing requests and providing resources.

Statelessness: The server does not store any information about the client's state or session. Each request from the client contains all the necessary information for the server to fulfill it. This improves scalability, performance, and reliability of the web service.

Cacheability: The server can indicate whether the resources are cacheable or not, and the client can store the responses in a local cache to reduce the number of requests and improve efficiency. The cache can be implemented on the client, the server, or any intermediate layer.

Layered system: The client and the server do not need to know the details of each other's implementation or the network infrastructure. There can be multiple layers or components between them, such as proxies, gateways, firewalls, etc. This enhances security, modularity, and flexibility of the web service. [1]

IV. REST FRAMEWORK CONCEPTS

Serializers: Are a way of converting complex data types, such as models and querysets, into native Python datatypes that can be easily rendered into JSON, XML, or other formats. They also allow you to validate and deserialize incoming data, creating or updating model instances based on the validated data. Serializers are like Django's forms and model forms, but they work with REST framework's views and responses.

There are different types of serializers in REST framework, such as Serializer, ModelSerializer, HyperlinkedModelSerializer, ListSerializer, and BaseSerializer. Each serializer class has a set of fields that determine how the data is validated, serialized, and deserialized. You can also define custom fields and validators for your serializers or use third-party packages that provide additional fields and features.

```python
from django.contrib.auth.models import User, Group
from rest_framework import serializers


class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ['url', 'username', 'email', 'groups']


class GroupSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Group
        fields = ['url', 'name']
```

Views: Core component of Django REST framework, as they provide a way to write logic for handling different types of requests, such as GET, POST, PUT, PATCH, or DELETE. Views can also perform validation, serialization, deserialization, content negotiation, authentication, permission checking, and throttling on the incoming and outgoing data.

* UTT Cruz Jose

There are different ways to write views in Django REST framework, depending on the level of customization and abstraction you need. Some of the common types of views are:

APIViews: These are the most basic type of views, which inherit from Django's View class. They allow you to define handler methods for different HTTP methods, such as. get() or .post(), and use REST framework's Request and Response objects instead of Django's HttpRequest and HttpResponse. APIViews give you full control over the logic and behavior of your views, but you must write more code than other types of views.

Generic views: These are subclasses of APIView that provide some common functionality for standard CRUD operations, such as list, create, retrieve, update, or destroy. They use serializer classes to handle validation and serialization, and queryset attributes to determine the data source. Generic views reduce the amount of code you must write, but you can still override or customize any of the methods or attributes as needed.

```
from django.contrib.auth.models import User, Group
from rest_framework import viewsets
from rest_framework import permissions
from tutorial.quickstart.serializers import UserSerializer, GroupSerializer


class UserViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows users to be viewed or edited.
    """
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer
    permission_classes = [permissions.IsAuthenticated]
```

URLs: Are an important part of designing a RESTful API, as they define how the clients can access and manipulate the resources on the server. URLs should be consistent, descriptive, and follow some best practices to make the API easy to use and understand:

Routers: Routers are a way of automatically generating URL patterns for your viewsets, based on common conventions. Routers can save you time and effort by creating standard CRUD routes for your resources, such as users/, users/{pk}/, accounts/, and accounts/{pk}/. You can also customize the routers to suit your needs, or use different types of routers, such as SimpleRouter, DefaultRouter, or CustomRouter. To use routers, you need to register your viewsets with a router instance, and then include the router.urls in your URL conf1.

Reverse: Reverse is a utility function that allows you to return absolute URIs from your API views, using the view name and optional arguments or keyword arguments. Reverse can help you create hyperlinks in your responses, which can improve the usability and discoverability of your API. For example, you can use reverse to get the URL of a user detail view, like this: url = reverse('user-detail', kwargs={'pk': 1}). You can also use reverse with request objects, to get the full URL including the domain name and query parameters.

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

router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'groups', views.GroupViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

Pagination: Is a technique of dividing a large set of data into smaller subsets, which can be accessed by using page numbers, offsets, cursors, or other indicators. Pagination is useful for improving the performance and usability of your API, as it reduces the amount of data that needs to be transferred and processed at a time. [2]

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10
}
```

V. CONCLUSION

Django REST Framework is a tool that allows us to have more tools to create Web applications. It is quite useful since it allows you to have greater control over important aspects such as urls, data pagination, views and the serializer. The latter is one of the most important since it allows us to render data in JSON, XML, among others, to communicate with other formats.

VI. REFERENCES

[1] Christie, T. (no date) Quickstart, Quickstart - Django REST framework. Available at: https://www.django-rest-framework.org/tutorial/quickstart/#quickstart (Accessed: 21 November 2023).

[2] GirlLovesToCode), Špela G. (aka (2023) Django rest framework basics, testdriven.io. Available at: https://testdriven.io/blog/drf-basics/ (Accessed: 21 November 2023).