Top 75 Django REST Framework Interview Questions for 2025

July 2025

Introduction

This document provides 75 Django REST Framework (DRF) interview questions with answers and code examples, organized into Basics, Intermediate, and Advanced sections. Designed for 2025 technical interviews, it helps developers prepare for roles requiring DRF expertise. Each question includes a practical code snippet to illustrate the concept.

1 DRF Basics

1. **What is Django REST Framework?**
   Answer: DRF is a toolkit for building Web APIs in Django, providing tools for serialization, authentication, and views.

```
# settings.py
INSTALLED_APPS = ['rest_framework']
```

2. **How do you install DRF?**
   Answer: Install via `pip` and add `rest_framework` to `INSTALLED_APPS`.

```
# Terminal
pip install djangorestframework

# settings.py
INSTALLED_APPS = ['rest_framework']
```

3. **What is a serializer in DRF?**
   Answer: A serializer converts model instances to JSON and validates input data.

```
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = ['id', 'title']
```

4. **How do you create a basic API view?**
   Answer: Use `APIView` to define request handling logic.

```python
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class PostView(APIView):
    def get(self, request):
        return Response({"message": "Hello, DRF!"})
```

5. **What are generic views in DRF?**
   Answer: Pre-built views for common CRUD operations, like `ListAPIView`.

```python
# blog/views.py
from rest_framework.generics import ListAPIView
from .models import Post
from .serializers import PostSerializer
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

6. **How do you define URL patterns for DRF?**
   Answer: Map views to URLs using `path()` in `urls.py`.

```python
# blog/urls.py
from django.urls import path
from .views import PostList
urlpatterns = [path('posts/', PostList.as_view(),
   name='post-list')]
```

7. **What is the purpose of `Response` in DRF?**
   Answer: `Response` wraps API data with status codes and headers.

```python
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class PostView(APIView):
    def get(self, request):
        return Response({'status': 'success'}, status=200)
```

8. **How do you handle GET requests in DRF?**
   Answer: Define a `get` method in an `APIView` or use generic views.

```python
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class PostView(APIView):
```

```
    def get(self, request):
        posts = Post.objects.all()
        serializer = PostSerializer(posts, many=True)
        return Response(serializer.data)
```

9. **How do you handle POST requests in DRF?**
   Answer: Validate and save data using a serializer in a `post` method.

```
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class PostCreate(APIView):
    def post(self, request):
        serializer = PostSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=201)
        return Response(serializer.errors, status=400)
```

10. **What is a `ModelSerializer`?**
    Answer: A serializer that automatically maps model fields to JSON.

```
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = ['id', 'title', 'content']
```

11. **How do you validate data in a serializer?**
    Answer: Use `validate` methods or field-level validation.

```
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    def validate_title(self, value):
        if len(value) < 5:
            raise serializers.ValidationError("Title too
                short")
        return value
    class Meta:
        model = Post
        fields = ['id', 'title']
```

12. **What is the `to_representation` method?**
    Answer: Customizes the serialized output of a serializer.

```
# blog/serializers.py
```

```python
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    def to_representation(self, instance):
        data = super().to_representation(instance)
        data['custom_field'] = 'value'
        return data
    class Meta:
        model = Post
        fields = ['id', 'title']
```

13. **How do you use `CreateAPIView`?**
Answer: A generic view for creating objects via POST requests.

```python
# blog/views.py
from rest_framework.generics import CreateAPIView
from .models import Post
from .serializers import PostSerializer
class PostCreate(CreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

14. **What is the `ListCreateAPIView`?**
Answer: Combines listing and creating objects in one view.

```python
# blog/views.py
from rest_framework.generics import ListCreateAPIView
from .models import Post
from .serializers import PostSerializer
class PostListCreate(ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

15. **How do you handle authentication in DRF?**
Answer: Configure `DEFAULT_AUTHENTICATION_CLASSES` in settings.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
    ]
}
```

16. **What is session authentication in DRF?**
Answer: Uses Djangos session framework for user authentication.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
```

```
        'rest_framework.authentication.SessionAuthentication',
    ]
}
```

17. **How do you implement token authentication in DRF?**
Answer: Use `rest_framework.authtoken` to generate tokens.

```
# settings.py
INSTALLED_APPS = ['rest_framework.authtoken']
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ]
}
# urls.py
from django.urls import path
from rest_framework.authtoken.views import obtain_auth_token
urlpatterns = [path('api-token-auth/', obtain_auth_token)]
```

18. **What is a permission class in DRF?**
Answer: Controls access to views based on user permissions.

```
# blog/views.py
from rest_framework.permissions import IsAuthenticated
from rest_framework.generics import ListAPIView
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    permission_classes = [IsAuthenticated]
```

19. **How do you use `IsAuthenticated` permission?**
Answer: Restricts access to authenticated users only.

```
# blog/views.py
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView
class SecureView(APIView):
    permission_classes = [IsAuthenticated]
    def get(self, request):
        return Response({"message": "Authenticated"})
```

20. **What is the `APIView` class?**
Answer: Base class for creating custom API views.

```
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class CustomView(APIView):
```

```
    def get(self, request):
        return Response({"message": "Custom API"})
```

21. **How do you handle query parameters in DRF?**
Answer: Access `request.query_params` in views.

```
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class SearchView(APIView):
    def get(self, request):
        query = request.query_params.get('q', '')
        return Response({"query": query})
```

22. **What is the status module in DRF?**
Answer: Provides HTTP status codes for responses.

```
# blog/views.py
from rest_framework import status
from rest_framework.response import Response
class PostView(APIView):
    def post(self, request):
        return Response({"message": "Created"},
            status=status.HTTP_201_CREATED)
```

23. **How do you configure DRF settings?**
Answer: Use `REST_FRAMEWORK` dictionary in `settings.py`.

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

24. **What is pagination in DRF?**
Answer: Splits large result sets into pages for better performance.

```
# blog/views.py
from rest_framework.pagination import PageNumberPagination
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    pagination_class = PageNumberPagination
```

25. **How do you test DRF APIs?**
Answer: Use `APITestCase` for testing API endpoints.

```python
# blog/tests.py
from rest_framework.test import APITestCase
class PostTests(APITestCase):
    def test_list_posts(self):
        response = self.client.get('/posts/')
        self.assertEqual(response.status_code, 200)
```

2 DRF Intermediate

1. **What is a `HyperlinkedModelSerializer`?**
   Answer: A serializer that includes URLs for related objects.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Post
        fields = ['url', 'title']
```

2. **How do you implement filtering in DRF?**
   Answer: Use `django-filter` with `filterset_fields`.

```python
# blog/views.py
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.generics import ListAPIView
from .models import Post
from .serializers import PostSerializer
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['title']
```

3. **How do you implement search in DRF?**
   Answer: Use `SearchFilter` with `search_fields`.

```python
# blog/views.py
from rest_framework.filters import SearchFilter
from rest_framework.generics import ListAPIView
from .models import Post
from .serializers import PostSerializer
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    filter_backends = [SearchFilter]
    search_fields = ['title']
```

4. **How do you implement sorting in DRF?**
   Answer: Use `OrderingFilter` with `ordering_fields`.

```python
# blog/views.py
from rest_framework.filters import OrderingFilter
from rest_framework.generics import ListAPIView
from .models import Post
from .serializers import PostSerializer
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    filter_backends = [OrderingFilter]
    ordering_fields = ['title']
```

5. **What is a viewset in DRF?**
   Answer: Combines logic for multiple related views into a single class.

```python
# blog/views.py
from rest_framework import viewsets
from .models import Post
from .serializers import PostSerializer
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

6. **How do you use routers with viewsets?**
   Answer: Use `DefaultRouter` to generate URLs automatically.

```python
# blog/urls.py
from rest_framework.routers import DefaultRouter
from .views import PostViewSet
router = DefaultRouter()
router.register(r'posts', PostViewSet)
urlpatterns = router.urls
```

7. **What is the `ReadOnlyModelViewSet`?**
   Answer: A viewset for read-only operations (GET, LIST).

```python
# blog/views.py
from rest_framework import viewsets
from .models import Post
from .serializers import PostSerializer
class PostViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

8. **How do you create a custom permission in DRF?**
   Answer: Subclass `BasePermission` and override `has_permission`.

```python
# blog/permissions.py
from rest_framework.permissions import BasePermission
class IsOwner(BasePermission):
    def has_object_permission(self, request, view, obj):
        return obj.owner == request.user
```

9. **How do you apply permissions to viewsets?**
   Answer: Set `permission_classes` in the viewset.

```python
# blog/views.py
from rest_framework import viewsets
from .permissions import IsOwner
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    permission_classes = [IsOwner]
```

10. **What is throttling in DRF?**
    Answer: Limits the rate of API requests to prevent abuse.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES':
       ['rest_framework.throttling.AnonRateThrottle'],
    'DEFAULT_THROTTLE_RATES': {'anon': '100/day'}
}
```

11. **How do you implement custom throttling?**
    Answer: Subclass `SimpleRateThrottle` and define `get_cache_key`.

```python
# blog/throttles.py
from rest_framework.throttling import SimpleRateThrottle
class CustomThrottle(SimpleRateThrottle):
    rate = '5/hour'
    def get_cache_key(self, request, view):
        return request.user.username
```

12. **How do you handle nested serializers?**
    Answer: Use a serializer for related fields with `many=True`.

```python
# blog/serializers.py
from rest_framework import serializers
class CommentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Comment
        fields = ['id', 'text']
```

```
class PostSerializer(serializers.ModelSerializer):
    comments = CommentSerializer(many=True, read_only=True)
    class Meta:
        model = Post
        fields = ['id', 'title', 'comments']
```

13. **What is the `to_internal_value` method?**
Answer: Customizes deserialization of input data in serializers.

```
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    def to_internal_value(self, data):
        data['title'] = data['title'].upper()
        return super().to_internal_value(data)
    class Meta:
        model = Post
        fields = ['id', 'title']
```

14. **How do you handle file uploads in DRF?**
Answer: Use `FileField` in serializers and process `request.FILES`.

```
# blog/serializers.py
from rest_framework import serializers
class DocumentSerializer(serializers.ModelSerializer):
    file = serializers.FileField()
    class Meta:
        model = Document
        fields = ['id', 'file']
```

15. **What is the `Renderer` class in DRF?**
Answer: Converts response data to formats like JSON or XML.

```
# blog/views.py
from rest_framework.renderers import JSONRenderer
class PostView(APIView):
    renderer_classes = [JSONRenderer]
    def get(self, request):
        return Response({"message": "JSON output"})
```

16. **How do you customize response formats?**
Answer: Use custom renderers or override `renderers`.

```
# blog/renderers.py
from rest_framework.renderers import JSONRenderer
class CustomRenderer(JSONRenderer):
    def render(self, data, accepted_media_type=None,
       renderer_context=None):
```

```
        data['custom'] = 'value'
        return super().render(data, accepted_media_type,
            renderer_context)
```

17. **What is the `Parser` class in DRF?**
Answer: Parses incoming request data into Python objects.

```
# blog/views.py
from rest_framework.parsers import JSONParser
class PostView(APIView):
    parser_classes = [JSONParser]
    def post(self, request):
        return Response({"data": request.data})
```

18. **How do you implement pagination with viewsets?**
Answer: Set `pagination_class` in the viewset.

```
# blog/views.py
from rest_framework import viewsets
from rest_framework.pagination import PageNumberPagination
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    pagination_class = PageNumberPagination
```

19. **How do you handle authentication errors?**
Answer: Catch exceptions in views or use custom handlers.

```
# blog/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class SecureView(APIView):
    def get(self, request):
        try:
            user = request.user
            return Response({"user": user.username})
        except AuthenticationFailed:
            return Response({"error": "Unauthenticated"},
                status=401)
```

20. **What is the `GenericAPIView`?**
Answer: Base class for generic views with reusable methods.

```
# blog/views.py
from rest_framework.generics import GenericAPIView
from .serializers import PostSerializer
class PostView(GenericAPIView):
    queryset = Post.objects.all()
```

```
    serializer_class = PostSerializer
    def get(self, request):
        serializer =
           self.get_serializer(self.get_queryset(),
           many=True)
        return Response(serializer.data)
```

21. **How do you use mixins in DRF?**
Answer: Combine mixins with `GenericAPIView` for reusable behavior.

```
# blog/views.py
from rest_framework.mixins import ListModelMixin,
   CreateModelMixin
from rest_framework.generics import GenericAPIView
class PostView(ListModelMixin, CreateModelMixin,
   GenericAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)
```

22. **How do you implement JWT authentication?**
Answer: Use `djangorestframework-simplejwt` for JWT tokens.

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ]
}
# urls.py
from django.urls import path
from rest_framework_simplejwt.views import
   TokenObtainPairView
urlpatterns = [path('api/token/',
   TokenObtainPairView.as_view())]
```

23. **What is the `DefaultRouter`?**
Answer: Automatically generates URL patterns for viewsets.

```
# blog/urls.py
from rest_framework.routers import DefaultRouter
from .views import PostViewSet
router = DefaultRouter()
router.register(r'posts', PostViewSet)
urlpatterns = router.urls
```

24. **How do you handle relationships in serializers?**
Answer: Use `PrimaryKeyRelatedField` or nested serializers.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    author =
       serializers.PrimaryKeyRelatedField(queryset=User.objects.all())
    class Meta:
        model = Post
        fields = ['id', 'title', 'author']
```

25. **How do you customize error responses?**
Answer: Override `exception_handler` for custom errors.

```python
# blog/exceptions.py
from rest_framework.views import exception_handler
def custom_exception_handler(exc, context):
    response = exception_handler(exc, context)
    if response is not None:
        response.data['error'] = 'Custom error message'
    return response
# settings.py
REST_FRAMEWORK = {
    'EXCEPTION_HANDLER':
       'blog.exceptions.custom_exception_handler'
}
```

26. **How do you implement versioning in DRF?**
Answer: Use URL or query parameter versioning in settings.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_VERSIONING_CLASS':
       'rest_framework.versioning.URLPathVersioning',
    'DEFAULT_VERSION': 'v1'
}
# urls.py
urlpatterns = [path('v1/posts/', PostList.as_view())]
```

3 DRF Advanced

1. **How do you implement custom authentication?**
Answer: Subclass `BaseAuthentication` and override `authenticate`.

```python
# blog/auth.py
from rest_framework.authentication import BaseAuthentication
```

```python
class CustomAuth(BaseAuthentication):
    def authenticate(self, request):
        token = request.headers.get('X-Custom-Token')
        if token == 'valid-token':
            return (None, None)
        raise AuthenticationFailed('Invalid token')
```

2. **What is a `SerializerMethodField`?**
   Answer: Adds computed fields to a serializer.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    custom_field = serializers.SerializerMethodField()
    def get_custom_field(self, obj):
        return f"Post: {obj.title}"
    class Meta:
        model = Post
        fields = ['id', 'title', 'custom_field']
```

3. **How do you handle nested writes in serializers?**
   Answer: Override `create` or `update` for nested data.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    comments = CommentSerializer(many=True)
    def create(self, validated_data):
        comments_data = validated_data.pop('comments')
        post = Post.objects.create(**validated_data)
        for comment_data in comments_data:
            Comment.objects.create(post=post, **comment_data)
        return post
    class Meta:
        model = Post
        fields = ['id', 'title', 'comments']
```

4. **How do you implement custom pagination?**
   Answer: Subclass `PageNumberPagination` and override attributes.

```python
# blog/pagination.py
from rest_framework.pagination import PageNumberPagination
class CustomPagination(PageNumberPagination):
    page_size = 5
    page_size_query_param = 'size'
    max_page_size = 100
```

5. **How do you use `GenericViewSet`?**
   Answer: Combine mixins with `GenericViewSet` for flexibility.

```python
# blog/views.py
from rest_framework import viewsets, mixins
class PostViewSet(mixins.ListModelMixin,
   viewsets.GenericViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

6. **What is the `action` decorator?**
   Answer: Defines custom actions in viewsets.

```python
# blog/views.py
from rest_framework import viewsets
from rest_framework.decorators import action
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    @action(detail=True, methods=['get'])
    def custom_action(self, request, pk=None):
        return Response({"message": "Custom action"})
```

7. **How do you implement rate limiting per user?**
   Answer: Use `UserRateThrottle` with custom rates.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES':
       ['rest_framework.throttling.UserRateThrottle'],
    'DEFAULT_THROTTLE_RATES': {'user': '1000/day'}
}
```

8. **How do you handle bulk operations in DRF?**
   Answer: Override view methods to process multiple objects.

```python
# blog/views.py
from rest_framework.views import APIView
class BulkCreateView(APIView):
    def post(self, request):
        serializer = PostSerializer(data=request.data,
           many=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=201)
        return Response(serializer.errors, status=400)
```

9. **What is the `ListSerializer`?**
   Answer: Handles serialization of multiple objects.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
        fields = ['id', 'title']
        list_serializer_class = serializers.ListSerializer
```

10. **How do you implement schema versioning?**
    Answer: Use `AcceptHeaderVersioning` for versioned responses.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_VERSIONING_CLASS':
       'rest_framework.versioning.AcceptHeaderVersioning',
    'DEFAULT_VERSION': 'v1'
}
```

11. **How do you use DRF with WebSockets?**
    Answer: Combine DRF with Channels for real-time APIs.

```python
# blog/consumers.py
from channels.generic.websocket import
   AsyncJsonWebsocketConsumer
class PostConsumer(AsyncJsonWebsocketConsumer):
    async def connect(self):
        await self.accept()
    async def receive_json(self, content):
        await self.send_json({"message": content})
```

12. **How do you optimize DRF performance?**
    Answer: Use `select_related`, caching, and pagination.

```python
# blog/views.py
from rest_framework.views import APIView
from django.views.decorators.cache import cache_page
@cache_page(60 * 10)
class PostList(APIView):
    def get(self, request):
        posts = Post.objects.select_related('author')
        serializer = PostSerializer(posts, many=True)
        return Response(serializer.data)
```

13. **What is the `NestedRouter`?**
    Answer: Extends `DefaultRouter` for nested resources.

```python
# blog/urls.py
from rest_framework_nested.routers import NestedSimpleRouter
router = DefaultRouter()
posts_router = router.register(r'posts', PostViewSet)
NestedSimpleRouter(posts_router, r'posts',
   lookup='post').register(r'comments', CommentViewSet)
```

14. **How do you implement API documentation?**
Answer: Use `drf-spectacular` for OpenAPI schemas.

```python
# settings.py
INSTALLED_APPS = ['drf_spectacular']
REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS':
       'drf_spectacular.openapi.AutoSchema'
}
```

15. **How do you handle partial updates?**
Answer: Use `partial=True` in serializer for PATCH requests.

```python
# blog/views.py
from rest_framework.views import APIView
class PostUpdate(APIView):
    def patch(self, request, pk):
        post = Post.objects.get(pk=pk)
        serializer = PostSerializer(post, data=request.data,
           partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)
```

16. **What is the `ModelViewSet`?**
Answer: Combines all CRUD operations in a single viewset.

```python
# blog/views.py
from rest_framework import viewsets
from .models import Post
from .serializers import PostSerializer
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

17. **How do you implement custom actions in viewsets?**
Answer: Use `@action` to add custom endpoints.

```python
# blog/views.py
from rest_framework import viewsets
from rest_framework.decorators import action
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    @action(detail=False, methods=['get'])
    def latest(self, request):
        latest_post = Post.objects.latest('created_at')
        serializer = self.get_serializer(latest_post)
        return Response(serializer.data)
```

18. **How do you handle file downloads in DRF?**
Answer: Return a `FileResponse` with the file content.

```python
# blog/views.py
from rest_framework.views import APIView
from django.http import FileResponse
class FileDownload(APIView):
    def get(self, request):
        file = open('file.pdf', 'rb')
        return FileResponse(file, as_attachment=True,
           filename='file.pdf')
```

19. **How do you implement API caching?**
Answer: Use `cache_page` or `CacheResponseMixin`.

```python
# blog/views.py
from rest_framework.views import APIView
from django.views.decorators.cache import cache_page
@cache_page(60 * 10)
class PostList(APIView):
    def get(self, request):
        posts = Post.objects.all()
        serializer = PostSerializer(posts, many=True)
        return Response(serializer.data)
```

20. **What is the `APIView` renderer context?**
Answer: Provides metadata for rendering responses.

```python
# blog/views.py
from rest_framework.views import APIView
class PostView(APIView):
    def get(self, request):
        renderer_context = self.renderer_context
        return Response({"view":
           renderer_context['view'].__class__.__name__})
```

21. **How do you handle large datasets in DRF?**
Answer: Use pagination and efficient querysets.

```python
# blog/views.py
from rest_framework.pagination import LimitOffsetPagination
class PostList(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    pagination_class = LimitOffsetPagination
```

22. **How do you implement custom validation logic?**
Answer: Override `validate` in serializers for complex rules.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    def validate(self, data):
        if data['title'] == data['content']:
            raise serializers.ValidationError("Title and
                content cannot be the same")
        return data
    class Meta:
        model = Post
        fields = ['id', 'title', 'content']
```

23. **What is the `SerializerRelationField`?**
Answer: Customizes serialization of related objects.

```python
# blog/serializers.py
from rest_framework import serializers
class PostSerializer(serializers.ModelSerializer):
    author = serializers.StringRelatedField()
    class Meta:
        model = Post
        fields = ['id', 'title', 'author']
```

24. **How do you implement OAuth2 authentication?**
Answer: Use `django-oauth-toolkit` for OAuth2 providers.

```python
# settings.py
INSTALLED_APPS = ['oauth2_provider']
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES':
       ['oauth2_provider.contrib.rest_framework.OAuth2Authentication']
}
```

25. **How do you handle concurrent API requests?**
Answer: Use database transactions or optimistic locking.

```python
# blog/views.py
from django.db import transaction
class PostUpdate(APIView):
    @transaction.atomic
    def put(self, request, pk):
        post = Post.objects.select_for_update().get(pk=pk)
        serializer = PostSerializer(post, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)
```

26. **How do you deploy DRF APIs with Docker?**
Answer: Containerize the application with a `Dockerfile`.

```dockerfile
# Dockerfile
FROM python:3.11
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "--bind", "0.0.0.0:8000",
   "myproject.wsgi:application"]
```

Conclusion

These 75 DRF questions with code examples prepare you for 2025 technical interviews. Practice these snippets and consult DRFs documentation for deeper insights.