Top 100 Django and Django REST Framework Interview Questions for 2025

July 2025

Introduction

This document provides 100 Django and Django REST Framework (DRF) interview questions with answers and code examples, designed for 2025 technical interviews. Organized into four sections, it covers beginner to advanced topics for developers preparing for web development and API design roles. Each question includes a practical code snippet to illustrate the concept.

1 Django Basics

1. **What is Django, and what are its key features?**
   Answer: Django is a Python web framework for rapid development, emphasizing security and scalability. Key features include ORM, admin interface, URL routing, and authentication.

```python
# settings.py
INSTALLED_APPS = ['django.contrib.admin',
   'django.contrib.auth']
```

2. **How do you start a Django project?**
   Answer: Use `django-admin startproject` to create a project directory.

```
django-admin startproject myproject
```

3. **What is the role of `manage.py`?**
   Answer: `manage.py` executes administrative tasks like running servers or migrations.

```
python manage.py runserver
```

4. **How do you create a Django app?**
   Answer: Run `python manage.py startapp <app_name>` to create an app directory, then add the app to `INSTALLED_APPS` in `settings.py`.

```
# Command to create a Django app
python manage.py startapp myapp
```

```

# settings.py
INSTALLED_APPS = [
    'myapp',
]
```

5. **What is a Django model?**
   Answer: A model is a Python class defining database structure and behavior.

```
# models.py
from django.db import models
class Post(models.Model):
    title = models.CharField(max_length=200)
```

6. **How do you apply migrations in Django?**
   Answer: Use `makemigrations` and `migrate` to update the database schema.

```
python manage.py makemigrations
python manage.py migrate
```

7. **What is the Django Admin Interface?**
   Answer: A web-based tool for managing model data, configured in `admin.py`.

```
# admin.py
from django.contrib import admin
from .models import Post
admin.site.register(Post)
```

8. **How do you define URLs in Django?**
   Answer: Use `path()` in `urls.py` to map URLs to views.

```
# urls.py
from django.urls import path
from . import views
urlpatterns = [path('posts/', views.post_list,
   name='post_list')]
```

9. **What is a Django view?**
   Answer: A view processes HTTP requests and returns responses.

```
# views.py
from django.http import HttpResponse
def post_list(request):
    return HttpResponse("Post list")
```

10. **How do you render templates in Django?**
    Answer: Use `render()` to combine templates with context data.

```
# views.py
from django.shortcuts import render
def home(request):
    return render(request, 'home.html', {'title': 'Home'})
```

11. **How do you configure templates in Django?**
Answer: Set `TEMPLATES['DIRS']` in `settings.py` to locate template files.

```
# settings.py
TEMPLATES = [{'BACKEND':
   'django.template.backends.django.DjangoTemplates', 'DIRS':
   ['templates']}]
```

12. **How do you handle static files in Django?**
Answer: Configure `STATIC_URL` and `STATICFILES_DIRS` in `settings.py`, use `% static %` in templates, and run `collectstatic` for production.

```
# settings.py
STATIC_URL = '/static/'
STATICFILES_DIRS = ['static']
STATIC_ROOT = 'staticfiles'

# Run in terminal
python manage.py collectstatic
```

13. **What is Django's CSRF protection?**
Answer: CSRF tokens prevent cross-site request forgery in forms.

```
<!-- template.html -->
<form method="post">{% csrf_token %}
    <input type="submit">
</form>
```

14. **How do you create a superuser in Django?**
Answer: Use `createsuperuser` to create an admin user.

```
python manage.py createsuperuser
```

15. **What is a Django QuerySet?**
Answer: A QuerySet is a lazily evaluated collection of database queries.

```
# views.py
from .models import Post
posts = Post.objects.all()
```

16. **How do you filter QuerySets in Django?**
Answer: Use `filter()` to query objects based on conditions.

```
posts = Post.objects.filter(published=True)
```

17. **What is the `exclude()` method?**
Answer: `exclude()` filters out objects matching criteria.

```
posts = Post.objects.exclude(published=False)
```

18. **How do you order QuerySets?**
Answer: Use `order_by()` to sort QuerySets, with `-` for descending order.

```
# views.py
from .models import Post
posts = Post.objects.order_by('-created_at')
```

19. **What is the `get()` method?**
Answer: `get()` retrieves a single object or raises an exception.

```
post = Post.objects.get(id=1)
```

20. **How do you create a Django form?**
Answer: Define a form class and render it in a template.

```
# forms.py
from django import forms
class PostForm(forms.Form):
    title = forms.CharField(max_length=200)
```

21. **What are Django template tags?**
Answer: Tags provide logic in templates, like loops or conditionals.

```
<!-- template.html -->
{% for post in posts %}
    <p>{{ post.title }}</p>
{% endfor %}
```

22. **How do you handle POST requests in Django?**
Answer: Check `request.method` and process `request.POST`.

```
# views.py
def create_post(request):
    if request.method == 'POST':
        form = PostForm(request.POST)
        if form.is_valid():
            form.save()
```

23. **What is Django's authentication system?**
Answer: It manages users, permissions, and sessions via `django.contrib.auth`.

```
# views.py
from django.contrib.auth import login
def login_view(request):
    login(request, user)
```

24. **How do you redirect in Django?**
Answer: Use `redirect()` to navigate to another URL.

```
# views.py
from django.shortcuts import redirect
def my_view(request):
    return redirect('home')
```

25. **What is the `DEBUG` setting?**
Answer: `DEBUG=True` shows detailed errors; set to `False` in production.

```
# settings.py
DEBUG = False
```

2 Django Intermediate

1. **What is a Django model manager?**
Answer: A manager provides QuerySet methods for models, customizable for specific queries.

```
# models.py
class PostManager(models.Manager):
    def published(self):
        return self.filter(published=True)
class Post(models.Model):
    objects = PostManager()
```

2. **How do you create a custom model manager?**
Answer: Subclass `models.Manager` and assign to `objects`.

```
# models.py
class Post(models.Model):
    objects = PostManager()
```

3. **What is a `ForeignKey` field?**
Answer: `ForeignKey` defines a one-to-many relationship.

```
# models.py
class Comment(models.Model):
    post = models.ForeignKey('Post', on_delete=models.CASCADE)
```

4. **What is a `ManyToManyField`?**
Answer: `ManyToManyField` creates a many-to-many relationship via a junction table.

```
# models.py
class Post(models.Model):
    tags = models.ManyToManyField('Tag')
```

5. **What is `select_related()`?**
Answer: `select_related()` performs a SQL JOIN to fetch ForeignKey-related objects in a single query, reducing database hits.

```
# views.py
from .models import Comment
comments = Comment.objects.select_related('post').all()
```

6. **What is `prefetch_related()`?**
Answer: `prefetch_related()` fetches ManyToMany or reverse ForeignKey objects in separate queries to optimize performance.

```python
# views.py
from .models import Post
posts = Post.objects.prefetch_related('tags').all()
```

7. **How do you create a class-based view?**
Answer: Subclass `View` or a generic view like `ListView`.

```python
# views.py
from django.views.generic import ListView
class PostListView(ListView):
    model = Post
    template_name = 'post_list.html'
```

8. **What are Django signals?**
Answer: Signals trigger functions on events like model saves.

```python
# signals.py
from django.db.models.signals import post_save
from django.dispatch import receiver
@receiver(post_save, sender=Post)
def post_saved(sender, instance, **kwargs):
    print("Post saved")
```

9. **How do you create a custom signal?**
Answer: Define a `Signal` and connect it to a receiver.

```python
# signals.py
from django.dispatch import Signal
custom_signal = Signal()
@receiver(custom_signal)
def handler(sender, **kwargs):
    print("Custom signal")
```

10. **What is Django middleware?**
Answer: Middleware processes requests and responses globally.

```python
# middleware.py
class LogMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
    def __call__(self, request):
        print("Request received")
        return self.get_response(request)
```

11. **How do you create custom middleware?**
Answer: Define a class and add it to `MIDDLEWARE`.

```
# settings.py
MIDDLEWARE = ['myapp.middleware.LogMiddleware']
```

12. **What is Django's caching framework?**
    Answer: It stores data to reduce database load, using backends like Redis.

```
# views.py
from django.views.decorators.cache import cache_page
@cache_page(60 * 10)
def post_list(request):
    return render(request, 'post_list.html')
```

13. **How do you handle file uploads?**
    Answer: Use `FileField` and process `request.FILES`.

```
# models.py
class Document(models.Model):
    file = models.FileField(upload_to='docs/')
# views.py
def upload(request):
    if request.method == 'POST':
        form = DocumentForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
```

14. **What is the Django Debug Toolbar?**
    Answer: A tool for profiling queries and performance.

```
# settings.py
INSTALLED_APPS = ['debug_toolbar']
MIDDLEWARE =
   ['debug_toolbar.middleware.DebugToolbarMiddleware']
```

15. **How do you implement pagination?**
    Answer: Use `Paginator` to split QuerySets into pages.

```
# views.py
from django.core.paginator import Paginator
def post_list(request):
    posts = Post.objects.all()
    paginator = Paginator(posts, 10)
    page = paginator.get_page(request.GET.get('page'))
    return render(request, 'post_list.html', {'page': page})
```

16. **What are template filters?**
    Answer: Filters transform template variables, like `upper`.

```
<!-- template.html -->
{{ title|upper }}
```

17. **How do you create a custom template filter?**
    Answer: Define a function and register it with `@register.filter`.

```python
# templatetags/filters.py
from django import template
register = template.Library()
@register.filter
def add_str(value, arg):
    return f"{value}{arg}"
```

18. **How do you extend the Django `User` model?**
    Answer: Use a one-to-one field or subclass `AbstractUser`.

```python
# models.py
from django.contrib.auth.models import AbstractUser
class CustomUser(AbstractUser):
    bio = models.TextField()
```

19. **What is `LoginRequiredMixin`?**
    Answer: Restricts views to authenticated users.

```python
# views.py
from django.contrib.auth.mixins import LoginRequiredMixin
class SecureView(LoginRequiredMixin, View):
    def get(self, request):
        return HttpResponse("Secure")
```

20. **How do you handle transactions?**
    Answer: Use `atomic()` for atomic database operations.

```python
# views.py
from django.db import transaction
@transaction.atomic
def update_post(request, pk):
    post = Post.objects.get(pk=pk)
    post.title = 'Updated'
    post.save()
```

21. **What is the contenttypes framework?**
    Answer: Enables generic relationships using `ContentType`.

```python
# models.py
from django.contrib.contenttypes.fields import
   GenericForeignKey
class Comment(models.Model):
    content_type =
       models.ForeignKey('contenttypes.ContentType',
       on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey()
```

22. **How do you implement internationalization?**
    Answer: Use `gettext` and `makemessages` for translations.

```python
# views.py
from django.utils.translation import gettext as _
def greet(request):
    return HttpResponse(_("Hello"))
```

23. **What are Django generic views?**
Answer: Pre-built views for common tasks, like `ListView`.

```python
# views.py
from django.views.generic import ListView
class PostListView(ListView):
    model = Post
    template_name = 'post_list.html'
```

24. **How do you secure a Django app?**
Answer: Use HTTPS, CSRF tokens, and set `DEBUG=False`.

```python
# settings.py
SECURE_SSL_REDIRECT = True
DEBUG = False
```

25. **How do you create a custom admin interface?**
Answer: Subclass `ModelAdmin` to customize fields and actions.

```python
# admin.py
from django.contrib import admin
class PostAdmin(admin.ModelAdmin):
    list_display = ['title', 'created_at']
admin.site.register(Post, PostAdmin)
```

26. **What is a Django formset?**
Answer: A formset manages multiple forms, useful for repeated data entry.

```python
# views.py
from django.forms import formset_factory
PostFormSet = formset_factory(PostForm)
def manage_posts(request):
    formset = PostFormSet(request.POST or None)
    if formset.is_valid():
        formset.save()
```

3 Django REST Framework Basics

1. **What is Django REST Framework?**
Answer: DRF builds RESTful APIs with serializers and viewsets.

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

2. **How do you install DRF?**
   Answer: Install Django REST Framework using `pip install djangorestframework` and add `'rest_framework'` to `INSTALLED_APPS` in `settings.py`.

```
# Install via terminal
pip install djangorestframework

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

3. **What is a DRF serializer?**
   Answer: Converts Python objects to JSON and validates data.

```
# serializers.py
from rest_framework import serializers
class PostSerializer(serializers.Serializer):
    title = serializers.CharField(max_length=200)
```

4. **What is a `ModelSerializer`?**
   Answer: Automatically maps model fields to serialized fields.

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

5. **How do you create a DRF API endpoint?**
   Answer: Define a serializer, view, and URL.

```
# views.py
from rest_framework.views import APIView
from rest_framework.response import Response
class PostList(APIView):
    def get(self, request):
        posts = Post.objects.all()
        serializer = PostSerializer(posts, many=True)
        return Response(serializer.data)
```

6. **What are DRF viewsets?**
   Answer: Viewsets combine API operations for a model.

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

7. **How do you use DRF routers?**
   Answer: Routers generate URLs for viewsets.

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

8. **What is DRF authentication?**
   Answer: Supports token, session, or JWT authentication.

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

9. **What are DRF permission classes?**
   Answer: Control access, like `IsAuthenticated`.

```python
# views.py
from rest_framework.permissions import IsAuthenticated
class PostList(APIView):
    permission_classes = [IsAuthenticated]
    def get(self, request):
        return Response({'data': 'Posts'})
```

10. **How do you enable CORS in DRF?**
    Answer: Use `django-cors-headers` for cross-origin requests.

```python
# settings.py
INSTALLED_APPS = ['corsheaders']
MIDDLEWARE = ['corsheaders.middleware.CorsMiddleware']
CORS_ALLOWED_ORIGINS = ['http://localhost:3000']
```

11. **What is the DRF browsable API?**
    Answer: An HTML interface for testing API endpoints.

```python
# urls.py
urlpatterns = [path('api/', include('myapp.urls'))]
```

12. **How do you handle GET requests in DRF?**
    Answer: Define a `get()` method in a view.

```python
# views.py
class PostList(APIView):
    def get(self, request):
        posts = Post.objects.all()
        serializer = PostSerializer(posts, many=True)
        return Response(serializer.data)
```

13. **What is deserialization in DRF?**
Answer: Converts JSON to Python objects with validation.

```python
# views.py
class PostCreate(APIView):
    def post(self, request):
        serializer = PostSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)
```

14. **How do you validate serializers?**
Answer: Use `validate()` or field-specific methods.

```python
# serializers.py
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']
```

15. **What are DRF generic views?**
Answer: Pre-built views for common API tasks.

```python
# views.py
from rest_framework.generics import ListAPIView
class PostListView(ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

16. **How do you implement pagination in DRF?**
Answer: Configure `PageNumberPagination` in `settings.py`.

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

17. **What is the `Response` class?**
Answer: Wraps API responses with data and status codes.

```python
# views.py
from rest_framework.response import Response
def my_view(request):
    return Response({'message': 'Hello'}, status=200)
```

18. **How do you handle nested serializers?**
Answer: Include serializers as fields with `many=True`.

```python
# serializers.py
class PostSerializer(serializers.ModelSerializer):
    tags = TagSerializer(many=True)
    class Meta:
        model = Post
        fields = ['id', 'tags']
```

19. **What is the `APIView` class?**
Answer: Base class for DRF views, handling HTTP methods.

```python
# views.py
from rest_framework.views import APIView
class MyView(APIView):
    def get(self, request):
        return Response({'data': 'Hello'})
```

20. **How do you test DRF APIs?**
Answer: Use `APITestCase` to simulate requests.

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

21. **What are DRF renderers?**
Answer: Convert responses to formats like JSON.

```python
# views.py
from rest_framework.renderers import JSONRenderer
class MyView(APIView):
    renderer_classes = [JSONRenderer]
    def get(self, request):
        return Response({'data': 'Hello'})
```

22. **How do you handle versioning in DRF?**
Answer: Use `URLPathVersioning` for versioned APIs.

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

23. **What is the `DefaultRouter`?**
Answer: Generates URLs for viewsets with optional suffixes.

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

24. **How do you implement filtering in DRF?**
Answer: Use `django-filter` with `DjangoFilterBackend` and specify `filterset_fields` in the view to filter querysets based on query parameters.

```python
# Install via terminal
pip install django-filter

# settings.py
INSTALLED_APPS = ['django_filters']

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

25. **How do you implement search in DRF?**
Answer: Use `SearchFilter` with `search_fields` in a DRF view to enable searching on specified model fields via query parameters.

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

4 Django REST Framework Advanced

1. **How do you implement token authentication in DRF?**
Answer: Use `rest_framework.authtoken` to generate tokens, add it to `INSTALLED_APPS`, configure `TokenAuthentication` in `settings.py`, and include the token authenti-

cation view in URLs.

```python
# settings.py
INSTALLED_APPS = [
    'rest_framework',
    '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,
       name='api_token_auth'),
]
```

2. **What is JWT authentication in DRF?**
   Answer: Uses JSON Web Tokens for stateless authentication.

```python
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES':
       ['rest_framework_simplejwt.authentication.JWTAuthentication']
}
```

3. **How do you create custom permissions in DRF?**
   Answer: Subclass `BasePermission` for custom access logic.

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

4. **How do you implement throttling in DRF?**
   Answer: Limit request rates with `UserRateThrottle`.

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

5. **How do you handle nested writes in DRF?**
   Answer: Override `create` or `update` in serializers.

```python
# serializers.py
class PostSerializer(serializers.ModelSerializer):
    tags = TagSerializer(many=True)
    class Meta:
        model = Post
        fields = ['id', 'tags']
    def create(self, validated_data):
        tags_data = validated_data.pop('tags')
        post = Post.objects.create(**validated_data)
        for tag_data in tags_data:
            Tag.objects.create(post=post, **tag_data)
        return post
```

6. **What are DRF parsers?**
   Answer: Process incoming data, like `JSONParser`.

```python
# views.py
from rest_framework.parsers import JSONParser
class MyView(APIView):
    parser_classes = [JSONParser]
    def post(self, request):
        return Response(request.data)
```

7. **How do you customize error responses?**
   Answer: Override `exception_handler` in a custom function to modify DRF error responses, then configure it in `settings.py`.

```python
# myapp/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':
       'myapp.exceptions.custom_exception_handler'
}
```

8. **How do you implement API documentation?**
   Answer: Use `drf-yasg` for Swagger/OpenAPI docs.

```python
# urls.py
from drf_yasg.views import get_schema_view
schema_view = get_schema_view()
urlpatterns = [path('swagger/',
   schema_view.with_ui('swagger'))]
```

9. **How do you handle background tasks in DRF?**
   Answer: Use Celery for asynchronous tasks.

```python
# tasks.py
from celery import shared_task
@shared_task
def process_post(post_id):
    post = Post.objects.get(id=post_id)
    print(f"Processed {post.title}")
# views.py
def my_view(request):
    process_post.delay(1)
    return Response({'status': 'Task started'})
```

10. **What is `ModelViewSet`?**
    Answer: Provides full CRUD operations for a model.

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

11. **How do you optimize DRF APIs?**
    Answer: Use caching, pagination, and query optimization.

```python
# views.py
class PostListView(ListAPIView):
    queryset = Post.objects.select_related('author')
    serializer_class = PostSerializer
```

12. **What is `ReadOnlyModelViewSet`?**
    Answer: Supports only read operations.

```python
# views.py
from rest_framework import viewsets
class PostReadOnlyViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

13. **How do you handle file uploads in DRF?**
    Answer: Use `FileField` and `MultiPartParser`.

```python
# serializers.py
class FileSerializer(serializers.ModelSerializer):
    file = serializers.FileField()
    class Meta:
        model = Document
        fields = ['file']
# views.py
from rest_framework.parsers import MultiPartParser
class FileUploadView(APIView):
    parser_classes = [MultiPartParser]
    def post(self, request):
        serializer = FileSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(status=201)
```

14. **What are viewset actions?**
    Answer: Custom endpoints defined with `@action`.

```python
# views.py
from rest_framework.decorators import action
class PostViewSet(viewsets.ModelViewSet):
    @action(detail=True, methods=['post'])
    def publish(self, request, pk=None):
        post = self.get_object()
        post.published = True
        post.save()
        return Response({'status': 'Published'})
```

15. **How do you implement sorting in DRF?**
    Answer: Use `OrderingFilter` with `ordering_fields` in a DRF view to enable sorting on specified model fields via query parameters.

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

16. **What is `HyperlinkedModelSerializer`?**
    Answer: Includes hyperlinks to related resources.

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

17. **How do you handle large datasets in DRF?**
    Answer: Use pagination or streaming responses.

```python
# views.py
from rest_framework.response import StreamingHttpResponse
def stream_posts(request):
    def generate():
        for post in Post.objects.iterator():
            yield f"{post.title}\n"
    return StreamingHttpResponse(generate())
```

18. **How do you secure a DRF API?**
    Answer: Use HTTPS, JWT, and permissions.

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES':
       ['rest_framework_simplejwt.authentication.JWTAuthentication'],
    'DEFAULT_PERMISSION_CLASSES':
       ['rest_framework.permissions.IsAuthenticated']
}
```

19. **What is `GenericAPIView`?**
    Answer: Extends `APIView` with mixins for common tasks.

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

20. **How do you implement rate limiting?**
    Answer: Configure throttling for specific views.

```
# views.py
from rest_framework.throttling import UserRateThrottle
class LimitedView(APIView):
    throttle_classes = [UserRateThrottle]
    def get(self, request):
        return Response({'data': 'Limited'})
```

21. **How do you handle API errors?**
    Answer: Raise `APIException` or customize error handling.

```
# views.py
from rest_framework.exceptions import APIException
class MyView(APIView):
    def get(self, request):
        raise APIException("Error occurred")
```

22. **How do you integrate Celery with DRF?**
    Answer: Define tasks and trigger them from views.

```
# tasks.py
from celery import shared_task
@shared_task
def send_notification(post_id):
    post = Post.objects.get(id=post_id)
    print(f"Notified for {post.title}")
# views.py
def my_view(request):
    send_notification.delay(1)
    return Response({'status': 'Task started'})
```

23. **What is `SerializerMethodField`?**
Answer: Computes dynamic field values.

```
# serializers.py
class PostSerializer(serializers.ModelSerializer):
    summary = serializers.SerializerMethodField()
    def get_summary(self, obj):
        return obj.title[:50]
    class Meta:
        model = Post
        fields = ['id', 'summary']
```

24. **How do you implement partial updates?**
Answer: Use `partial=True` for PATCH requests.

```
# views.py
class PostUpdateView(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)
```

25. **How do you deploy a DRF API?**
Answer: Use Gunicorn, Nginx, and set `DEBUG=False`.

```
# settings.py
DEBUG = False
ALLOWED_HOSTS = ['example.com']
# Run with Gunicorn
gunicorn myproject.wsgi:application --bind 0.0.0.0:8000
```

Conclusion

These 100 Django and DRF questions with code examples prepare you for 2025 interviews. Practice these snippets and explore Django/DRF documentation for deeper understanding.