CPAN 214: High Level Programming Languages

Week 13

Django REST Framework

Why DRF Exists

Traditional Django is optimized for server-side rendered HTML.

Modern systems need:

- API endpoints for mobile, web, and microservices
- Stateless interactions
- JSON serialization
- Authentication for tokens, sessions, Oauth
- Easy integration with frontend frameworks

DRF solves all of this with a clear, consistent architecture.

Django Built-In Authentication Functions

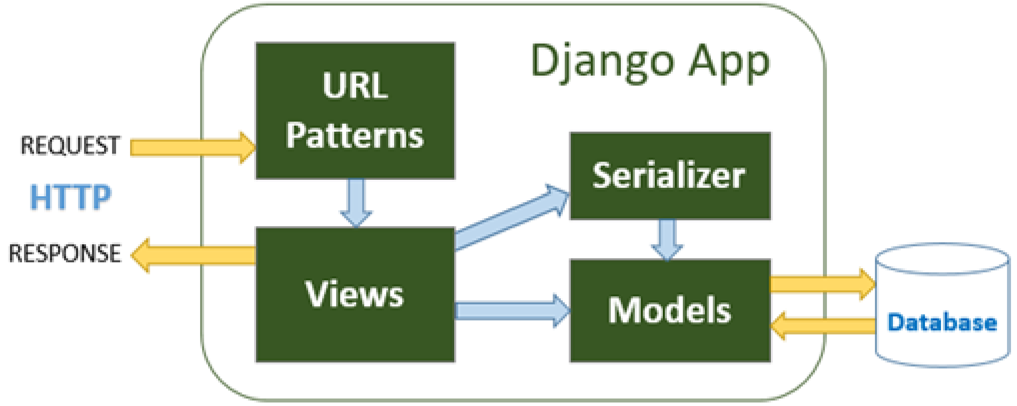
DRF sits on top of Django and provides:

- Request parsing (JSON, form, files)
- Response rendering (JSON, Browsable API)
- Serializers for model <-> JSON conversion
- Generic views to avoid boilerplate
- Authentication and permissions
- Pagination, filtering, throttling

Django Built-In Authentication Functions

DRF sits on top of Django and provides:

- Request parsing (JSON, form, files)
- Response rendering (JSON, Browsable API)
- Serializers for model <-> JSON conversion
- Generic views to avoid boilerplate
- Authentication and permissions
- Pagination, filtering, throttling



Serializers

Serializers convert Django model instances into JSON and back.

```
from rest_framework.serializers import ModelSerializer
class BookSerializer(ModelSerializer):
    class Meta:
        model = Book
        fields = ['id', 'title', 'year’]
```

Why this matters:

- Validation logic stays clean
- Works similarly to Django forms
- Enables nested resources

Views (Function Based API)

- Used for simpler cases or quick prototypes.
- But not ideal for large systems.

Example:

```
@api_view(['GET'])
def list_books(request):
    books = Book.objects.all()
    serializer = BookSerializer(books, many=True)
    return Response(serializer.data)
```

Views (Class Based API)

Advantages:

- Cleaner structure
- Override behavior in small functions
- Good for enterprise codebases

Example:

```
class BookList(APIView):
    def get(self, request):
        books = Book.objects.all()
        serializer = BookSerializer(books, many=True)
        return Response(serializer.data)
```

Generic Views (Best Practice)

DRF gives powerful built-ins:

- ListAPIView
- CreateAPIView
- RetrieveAPIView
- UpdateAPIView
- DestroyAPIView
- Combined views (ListCreate, RetrieveUpdateDestroy)

Production teams use these most of the time.

Example:

```
class BookListCreate(generics.ListCreateAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

ViewSets (Router Driven)

Generates:

- GET /books/
- POST /books/
- GET /books/3/
- PUT /books/3/
- DELETE /books/3/

Best for modular microservices.

Production teams use these most of the time.

Example:

```
class BookViewSet(ModelViewSet):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

Router:

```
router.register('books', BookViewSet)
```

Example: Blog API Model

```
class Blog(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    posted_by = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
```

This supports:

- Auth-bound content
- Permissions
- Filtering by user
- Pagination

Example: Blog Serializer

```
class BlogSerializer(ModelSerializer):
    posted_by_name = serializers.CharField(
        source='posted_by.username', read_only=True
    )

    class Meta:
        model = Blog
        fields = ['id', 'title', 'content', 'posted_by_name', 'created_at']
```

- Key idea: expose minimal fields for security.
- Never expose unnecessary user info.

Blog ViewSet

```
class BlogViewSet(ModelViewSet):
    serializer_class = BlogSerializer

    def get_queryset(self):
        return Blog.objects.all()

    def perform_create(self, serializer):
        serializer.save(posted_by=self.request.user)
```

- This enforces author tracking automatically.
- Perfect for real blog or LMS systems.

Authentication Options

DRF supports:

- Django session auth
- Token authentication
- JWT authentication (djangorestframework-simplejwt)
- OAuth via django-allauth
- Custom authentication classes

Most production APIs use JWT.

JWT Example

Install:

```
pip install djangorestframework-simplejwt
```

Settings:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
            'rest_framework_simplejwt.authentication.JWTAuthentication’,
     )
}
```

Login endpoint returns:

- access token
- refresh token.

Pagination

Global Settings:

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

Good for blogs, product lists, dashboards.

Filtering and Search

Install:

```
pip install django-filter
```

Example:

```
class BlogViewSet(ModelViewSet):
    filter_backends = [filters.SearchFilter, filters.OrderingFilter]
    search_fields = ['title', 'content']
    ordering_fields = ['created_at’]
```

We can search blog posts like: `/blogs/?search=django`

Throttling (Rate Limiting)

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

- Used to prevent abuse.
- Essential for public APIs.

Testing your API

Libraries

- `pytest`
- `pytest-django`

Example test:

```
def test_blog_list(api_client):
    response = api_client.get('/blogs/')
    assert response.status_code == 200
```

Production rule: never deploy without API tests.

.

Summary

- Core DRF components
- Serializers and ViewSets
- JWT authentication
- Permissions and throttling
- Pagination, filtering, search
- Designing scalable APIs
- Testing and documenting APIs

HUMBER
Faculty of Applied Sciences & Technology

Thank you