Comprehensive Theory Notes on APIs and Django REST Framework

# 1. What is an API (Application Programming Interface)?

An API acts as a bridge between two software applications, enabling them to communicate without knowing how they are implemented. APIs expose certain functionalities as endpoints so that clients (frontend or other servers) can access or modify resources.

Key Concepts:

* - Endpoints: Specific URL paths used to access resources (e.g., /api/users)
* - Request/Response Cycle: Clients send requests, and APIs return structured responses (usually JSON or XML)
* - Status Codes: APIs use HTTP status codes (e.g., 200 OK, 404 Not Found) to indicate result

Real-World Uses:

* - Online payments (PayPal, Stripe API)
* - Login with Google or Facebook (OAuth-based APIs)
* - Weather updates (OpenWeatherMap API)
* - Maps and navigation (Google Maps API)

# 2. Types of APIs: REST, SOAP, and more

REST (Representational State Transfer):

* - Stateless, cacheable architecture over HTTP
* - Uses standard HTTP methods: GET, POST, PUT, DELETE
* - Returns JSON (lightweight and readable)
* - Easy to integrate and scale
* - Widely used in modern web and mobile apps

SOAP (Simple Object Access Protocol):

* - Protocol for exchanging structured XML-based information
* - Strict and standardized with built-in security (WS-Security)
* - Requires more overhead than REST
* - Used in legacy enterprise systems (banking, finance)

Other types:

* - GraphQL: Allows clients to request exactly the data they need
* - gRPC: High-performance RPC framework, often used in microservices

# 3. Importance of APIs in Web Development

* - Frontend-backend communication: APIs allow the frontend (React, Vue) to fetch data from backend (Django, Node.js).
* - Microservices architecture: APIs help different services communicate independently.
* - Mobile app support: APIs serve mobile apps with necessary backend data.
* - Automation: APIs are essential for integrating tools (CI/CD, bots, schedulers).

Best Practices:

* - Use versioning (/api/v1/doctors)
* - Secure APIs with authentication (Token, JWT, OAuth)
* - Return meaningful error messages
* - Document your APIs (Swagger, Postman)

# 4. Requirements for Web Development Projects

Requirement Analysis:

* - User stories, system architecture
* - List of features/modules (e.g., login, search, profile)
* - API endpoints required
* - External APIs to be used
* - Role-based access control

Environment Setup:

* - Use virtual environments (python -m venv env)
* - Install dependencies using pip
* - Maintain a requirements.txt file using pip freeze > requirements.txt

Common Python Packages:

* - django: Core framework
* - djangorestframework: REST API support
* - requests: For making API calls
* - python-decouple, dotenv: For managing environment variables

# 5. Serialization in Django REST Framework

Serialization is a core concept of Django REST Framework (DRF). It handles the conversion between:

* - Python objects ↔ JSON data

When Serialization is used:

* - Sending data to frontend
* - Receiving and validating input from frontend
* - Saving input as model objects

Types in DRF:

* - Serializer: Custom serialization logic
* - ModelSerializer: Auto-generates fields based on model

Advanced Features:

* - Field-level validation (validate\_<fieldname>)
* - Object-level validation (validate)
* - Nested serializers (for related models)

# 6. Requests and Responses in DRF

Every API interaction involves a request and a response.

HTTP Methods:

* - GET: Retrieve data
* - POST: Create new resource
* - PUT/PATCH: Update resource
* - DELETE: Remove resource

Request Object:

DRF enhances Django’s request object with .data (parsed input data), .query\_params, .user, etc.

Response Object:

Return data using Response() from rest\_framework.response, with proper status codes (status.HTTP\_200\_OK, etc.)

# 7. Views in Django REST Framework

Views handle business logic and route processing.

Function-Based Views (FBV):

* - Simple and readable
* - Good for small APIs

Class-Based Views (CBV):

* - Use Django OOP concepts
* - Reusable and scalable

DRF View Classes:

* - APIView: Basic class-based view
* - GenericAPIView: Adds querysets and serializers
* - ModelViewSet: Full CRUD in one class

# 8. URL Routing in DRF

URL routing connects endpoints to views.

Manual Routing:

path('doctors/', DoctorList.as\_view())

Router-based (automatic):

router = DefaultRouter()

router.register(r'doctors', DoctorViewSet)

Best Practices:

* - Use versioned URLs: /api/v1/doctors/
* - Keep route names RESTful and meaningful
* - Group routes by module/app

# 9. Pagination in DRF

Pagination controls the number of records returned in a response, improving performance and usability.

Types in DRF:

* - PageNumberPagination
* - LimitOffsetPagination
* - CursorPagination

Configure in settings.py:

REST\_FRAMEWORK = {

'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.PageNumberPagination',

'PAGE\_SIZE': 10

}

# 10. Django Settings Configuration

Settings configure app behavior, security, and integrations.

Key Settings:

* - DATABASES: SQLite, MySQL, PostgreSQL
* - STATIC\_URL, MEDIA\_URL: File management
* - EMAIL\_BACKEND: Email configurations
* - INSTALLED\_APPS: App registration
* - .env or decouple: Keep sensitive info secure

Tips:

* - Use .env files for environment variables
* - Turn off DEBUG in production
* - Set ALLOWED\_HOSTS properly

# 11. Project Setup for Django + DRF

Steps:

1. Install Django and DRF

2. Start a new project: django-admin startproject

3. Create a new app: python manage.py startapp

4. Add the app to INSTALLED\_APPS

5. Define models, serializers, views, and routes

6. Run migrations and create superuser

Common Structure:

* - /models.py: Database structure
* - /serializers.py: JSON conversion
* - /views.py: API logic
* - /urls.py: Routing

# 12. Social Authentication, Email, and OTP

Social Login:

Use django-allauth or social-auth-app-django to add OAuth logins (Google, Facebook).

Email Sending:

Configure SMTP (Gmail, SendGrid) or APIs (Mailchimp) to send:

* - Welcome emails
* - Password reset links
* - OTPs

OTP via SMS:

Use Twilio to send SMS-based OTPs with custom templates.

# 13. RESTful API Design

Core REST Principles:

* - Statelessness: Each request is independent
* - Resource-based URLs: /users/, /products/
* - Standard HTTP methods
* - Structured responses: Mostly JSON
* - Status codes: Reflect request result

Good Practices:

* - Use plural nouns for endpoints
* - Avoid unnecessary nesting
* - Follow naming consistency

# 14. CRUD API Design

CRUD is the backbone of most web applications.

CRUD Operations:

* - Create: POST /doctors/
* - Read: GET /doctors/, /doctors/1/
* - Update: PUT/PATCH /doctors/1/
* - Delete: DELETE /doctors/1/

Use ModelViewSet to handle all of these with minimal code.

# 15. Authentication and Authorization

Authentication:

Confirms identity (login, tokens)

Authorization:

Grants or denies access based on roles

Types in DRF:

* - TokenAuth
* - JWT (via djangorestframework-simplejwt)
* - OAuth2

Security Tips:

* - Use HTTPS
* - Store tokens securely
* - Restrict access by roles and permissions

# 16. OpenWeatherMap API Integration

Use API key to fetch weather for any city.

Returned Data:

* - Temperature (°C/°F)
* - Wind speed
* - Weather status
* - Sunrise/sunset

Used in weather dashboards, travel apps, or agriculture tools.

# 17. Google Maps Geocoding API

Converts:

* - Address → Coordinates
* - Coordinates → Address (Reverse Geocoding)

Applications:

* - Food delivery
* - Cab services
* - Location-based marketing

# 18. GitHub API Integration

Use for:

* - Fetching user/public repo data
* - Creating issues or pull requests
* - Analyzing contributions

Great for building developer portfolios or dashboards.

# 19. Twitter API Integration

Enables:

* - Fetching tweets by user/hashtag
* - Posting automated tweets
* - Sentiment analysis

Use cases:

* - News apps
* - Marketing tools
* - Trend analysis

# 20. SMS and Email APIs (Twilio, SendGrid)

Twilio:

* - Send OTPs, alerts, messages
* - Verify user phone numbers
* - Integrate with workflows

SendGrid/Mailchimp:

* - Transactional and marketing emails
* - HTML templates and delivery tracking