COMPLETE REST API

FUNDAMENTALS

- ***Principles***: *Statelessness, Uniform Interface, Cacheability, Layered System.*
- ***HTTP Methods:*** *GET (read), POST (create), PUT/PATCH (update), DELETE (remove).*
- ***Status Codes:*** *200 (OK), 201 (Created), 400 (Bad Request), 401 (Unauthorized), 500 (Server Error).*

ENDPOINT DESIGN

- ***Resource Naming***: *Nouns (e.g., /users), not verbs.*
- ***Versioning***: *URL (/v1/users) vs Header (Accept: application/vnd.api.v1+json).*
- ***Pagination***: *limit, offset, or cursor-based.*

REQUEST & RESPONSE

- ***Headers***: *Content-Type, Authorization, Accept.*
- ***Body Formats***: *JSON (common), XML (legacy).*
- ***Query Params***: *Filtering (?status=active), sorting (?sort=-created_at).*

AUTHENTICATION & AUTHORIZATION

- ***JWT***: *Stateless tokens with Bearer scheme.*
- ***OAuth2***: *Flows (Authorization Code, Client Credentials).*
- ***API Keys***: *Simple but less secure.*

SECURITY

- ***HTTPS***: *Mandatory for production.*
- ***CORS***: *Control cross-origin requests.*
- ***Rate Limiting***: *Prevent abuse (e.g., X-RateLimit-Limit).*

ERROR HANDLING

- ***Standardized Errors***: *{ error: { code: 404, message: "Not Found" } }.*
- ***Validation Errors***: *400 with details (e.g., "email must be valid").*

PERFORMANCE

- ***Caching***: *Cache-Control headers, ETags.*
- ***Compression***: *Gzip/Deflate responses.*
- ***Lazy Loading***: *Partial responses (?fields=id,name).*

TESTING

- ***Tools***: *Postman, Swagger, supertest.*
- ***Test Cases***: *Happy path, edge cases, auth failures.*

DOCUMENTATION

- ***OpenAPI/Swagger***:

  Machine-readable specs.

- ***Examples***:

  Request/response samples.

ADVANCED TOPICS

- ***HATEOAS***: *Hypermedia-driven navigation.*
- ***GraphQL vs REST***: *When to use each.*

INTERVIEW QUESTIONS

BEGINNER-LEVEL QUESTIONS

1. **What are RESTful APIs, and what constraints do they follow?**
2. **Explain common HTTP methods and their idempotency.**
3. **How do you differentiate between PUT and PATCH?**
4. **What status code would you return after a successful resource creation?**
5. **How would you design a /users endpoint for CRUD operations?**
6. **What is the purpose of the Accept and Content-Type headers?**
7. **Why is HTTPS critical for REST APIs?**
8. **How do you handle a "Resource Not Found" scenario?**
9. **What are query parameters, and how are they used?**
10. **How would you version an API?**

INTERMEDIATE LEVEL QUESTIONS

1. **How do you implement pagination in a REST API?**
2. **Explain JWT authentication flow for APIs.**
3. **What is CORS, and how do you configure it?**
4. **How would you rate-limit an API endpoint?**
5. **Design an endpoint for bulk operations (e.g., delete multiple users).**
6. **How do you handle file uploads in a REST API?**
7. **What are ETags, and how do they optimize performance?**
8. **How would you document an API for developers?**
9. **Explain the trade-offs between API versioning strategies.**
10. **How do you validate request payloads?**

ADVANCE LEVEL QUESTIONS

1. **How would you design an API for a real-time collaboration tool?**
2. **Implement HATEOAS in a product catalog API.**
3. **Secure an API against SQL injection and XSS.**
4. **Optimize an API for high latency mobile clients.**
5. **Design a caching strategy for a read-heavy API.**
6. **How would you migrate an API from REST to GraphQL?**
7. **Implement idempotency for a payment processing API.**
8. **Handle partial failures in a batch API request.**
9. **Design a webhook system for event notifications.**
10. **How would you scale an API to 1M+ requests per minute?**