# **REST APIs**

## **1. Introduction to REST APIs**

### **What is a REST API?**

* REST = Representational State Transfer.
* An architecture style for designing networked applications.
* REST APIs allow communication between client and server using standard HTTP methods.

### **Characteristics of REST:**

* **Stateless**: Each request from client to server must contain all necessary information.
* **Client-Server Architecture**: Clear separation between client (frontend) and server (backend).
* **Cacheable**: Responses must define themselves as cacheable or not.
* **Uniform Interface**: Consistent way to access resources.

## **2. Core Concepts**

### **Resources**

* A resource is an object or representation of something (e.g., User, Post, Product).
* Resources are accessed using URIs (Uniform Resource Identifiers).

### **HTTP Methods**

| **Method** | **Purpose** | **Example** |
| --- | --- | --- |
| GET | Retrieve a resource | GET /users/ |
| POST | Create a new resource | POST /users/ |
| PUT | Update an entire resource | PUT /users/123/ |
| PATCH | Update part of a resource | PATCH /users/123/ |
| DELETE | Delete a resource | DELETE /users/123/ |

### **HTTP Status Codes**

* **200 OK**: Request succeeded.
* **201 Created**: Resource created successfully.
* **204 No Content**: Successful request but no content to return.
* **400 Bad Request**: Client error (malformed request).
* **401 Unauthorized**: Authentication required.
* **403 Forbidden**: Client authenticated but does not have permission.
* **404 Not Found**: Resource not found.
* **500 Internal Server Error**: Server encountered an error.

## **3. Data Format**

* Commonly exchanged in **JSON**.
* Example of a JSON body for creating a user:

{

"username": "john\_doe",

"email": "john@example.com",

"password": "securepassword"

}

## **4. Best Practices**

* **Use nouns**, not verbs, in URIs: /users/, not /createUser/.
* **Version your API**: /api/v1/users/
* **Use proper HTTP methods**.
* **Paginate large results**: /users?page=2&size=20
* **Secure with HTTPS**.
* **Return proper status codes and error messages**.
* **Authentication & Authorization**: Use standards like OAuth2, JWT.

## **5. Advanced Topics**

### **Authentication**

* **API Keys**: Simple but not very secure.
* **OAuth2**: Industry-standard for authorization.
* **JWT (JSON Web Tokens)**: Secure tokens carrying user identity information.

### **Rate Limiting**

* Protect API from abuse.
* Example: 1000 requests per hour per user.

### **Caching**

* Improves performance by storing responses temporarily.
* HTTP Headers: Cache-Control, ETag.

### **HATEOAS**

* Hypermedia As The Engine Of Application State.
* Responses include links to other actions available.
* Example:

{

"postId": 123,

"title": "REST APIs",

"links": [

{"rel": "self", "href": "/posts/123"},

{"rel": "comments", "href": "/posts/123/comments"}

]

}

### **API Documentation**

* Always document your APIs.
* Tools like **Swagger/OpenAPI** automate API documentation.

## **6. Security Tips**

* Always use **HTTPS**.
* Validate all inputs on the server side.
* Use proper **authentication and authorization**.
* Prevent **CORS issues** with proper headers.
* Implement **throttling and rate limiting**.
* Log and monitor suspicious activities.

## **7. Example REST API Flow**

### **Create a new user**

1. Client sends:

POST /api/v1/users/

Content-Type: application/json

{

"username": "newuser",

"password": "password123"

}

1. Server responds:

201 Created

Location: /api/v1/users/45

{

"id": 45,

"username": "newuser"

}

### **Fetch user profile**

GET /api/v1/users/45

Authorization: Bearer <token>

Server responds with the user details.

## **8. Tools for Working with REST APIs**

* **Postman**: For testing APIs easily.
* **curl**: Command-line HTTP client.
* **Swagger / OpenAPI**: For API documentation and testing.
* **Insomnia**: API design and debugging tool.