# **Introduction to API (Application Programming Interface)**

An **API (Application Programming Interface)** is a **set of rules** that allows one software application to communicate with another. It defines how requests should be sent and how responses should be formatted. APIs enable **integration** between different systems, allowing applications to share data and functionality.

## **1. API Terminologies**

* **Endpoint**: A specific URL where an API service can be accessed (/users, /products).
* **Request**: Data sent from the client to the API (e.g., fetching a user’s details).
* **Response**: Data returned by the API after processing a request.
* **Headers**: Metadata sent with requests and responses (e.g., authentication tokens).
* **Payload**: The actual data sent in a request, usually in **JSON** format.
* **Rate Limiting**: Restriction on the number of API calls per time frame.

## **2. Types of APIs**

1. **Open (Public) API** – Available to everyone (e.g., OpenWeather API).
2. **Private API** – Used within an organization for internal applications.
3. **Partner API** – Shared with business partners (e.g., Stripe API).
4. **Composite API** – Combines multiple API calls into one request.

## **3. Secure vs Unsecured APIs**

* **Secured API**: Requires authentication (API keys, OAuth, JWT) to access.
* **Unsecured API**: Open access without authentication, prone to security risks.

## **4. Private vs Public APIs**

* **Public API**: Available for external developers to use (e.g., Twitter API).
* **Private API**: Used only within an organization (e.g., a company's internal HR API).

## **5. RESTful API (Representational State Transfer)**

A **REST API** is a web API that follows REST principles:  
 ✅ **Stateless** – Each request is independent; no session is stored.  
 ✅ **Client-Server** – The frontend and backend are separated.  
 ✅ **Resource-Based** – Uses URIs to identify resources (/users, /products).  
 ✅ **Uses HTTP Methods** – Standard methods like GET, POST, PUT, DELETE.

### **REST API Flow**

1. **Client** sends a request (GET /users/1).
2. **Server** processes the request.
3. **Database** fetches the required data.
4. **Server** returns a response (usually in JSON).
5. **Client** displays the data.

## **6. API Documentation Tools**

* **Swagger (OpenAPI)** – Interactive API documentation.
* **Postman** – API testing and documentation.
* **Redoc** – OpenAPI documentation generator.

## **7. API Integration Steps**

1. **Read API Documentation** – Understand endpoints, authentication, and response formats.
2. **Obtain API Keys or Tokens** – Required for authenticated APIs.
3. **Make API Requests** – Use tools like **Postman** or code (cURL, JavaScript).
4. **Handle API Responses** – Process and display the data.

## **8. HTTP Request Methods**

* **GET** – Retrieve data.
* **POST** – Create a new resource.
* **PUT/PATCH** – Update an existing resource.
* **DELETE** – Remove a resource.

## **9. HTTP Response Codes**

* **200 OK** – Request successful.
* **201 Created** – New resource successfully created.
* **400 Bad Request** – Invalid request data.
* **401 Unauthorized** – Missing authentication.
* **403 Forbidden** – No permission.
* **404 Not Found** – Resource does not exist.
* **422 Unprocessable Entity** – Validation error.
* **500 Internal Server Error** – Server-side issue.

## **11. API Naming Conventions (Best Practices)**

✅ Use **plural nouns** for resource names (/users instead of /user).  
 ✅ Use **hyphens** in URL paths (/user-profile instead of /userProfile).  
 ✅ Use **lowercase** letters in endpoints (/orders instead of /Orders).  
 ✅ Use **HTTP verbs** instead of action words:

* ✅ GET /users (instead of POST /getUsers)
* ✅ DELETE /users/1 (instead of POST /deleteUser)  
   ✅ Use **query parameters** for filtering (GET /products?category=electronics).

## **12. API Testing**

API testing ensures endpoints work as expected.

### **Testing Tools**

✅ **Postman** – Manual testing of API requests.  
 ✅ **Jest + Supertest** – Automated API testing in Node.js.  
 ✅ **Newman** – Run Postman tests in CI/CD pipelines.

### **Example API Test (Jest + Supertest in Node.js)**

javascript

CopyEdit

const request = require("supertest");

const app = require("../app");

describe("GET /users", () => {

it("should return a list of users", async () => {

const response = await request(app).get("/users");

expect(response.status).toBe(200);

expect(response.body).toBeInstanceOf(Array);

});

});

## **13. Authentication & Authorization in APIs**

**Authentication** verifies *who* the user is.  
 **Authorization** checks *what* the user is allowed to do.

### **Different Authentication Methods**

✅ **API Keys** – A secret key sent in headers (Authorization: Bearer <API\_KEY>).  
 ✅ **JWT (JSON Web Token)** – Encoded token with user details.  
 ✅ **OAuth2** – Used for third-party login (Google, Facebook).  
 ✅ **Session-based Authentication** – Uses cookies for tracking sessions.

## **14. Example: JWT Authentication in Node.js**

### **Generating a Token**

javascript

CopyEdit

const jwt = require('jsonwebtoken');

const user = { id: 1, username: "john\_doe" };

const token = jwt.sign(user, "your-secret-key", { expiresIn: "1h" });

console.log(token);

### **Protecting Routes**

javascript

CopyEdit

const verifyToken = (req, res, next) => {

const token = req.headers["authorization"];

if (!token) return res.status(401).send("Access Denied");

try {

req.user = jwt.verify(token.split(" ")[1], "your-secret-key");

next();

} catch (err) {

res.status(403).send("Invalid Token");

}

};

app.get("/protected", verifyToken, (req, res) => {

res.send("This is a protected route");

});