# 

[**API (Application Programming Interface) 4**](#_vu71lmj4fnk5)

[Why is an API Used? 4](#_ul1e4jvlxi4y)

[Examples of APIs in Action: 4](#_si1beo7wbxgh)

[**Different Types of API based on Structure 5**](#_u5p5itu19cwc)

[1. REST API (Representational State Transfer) 5](#_tagcr73plld9)

[REST API – Like Ordering Food Online 🍕 5](#_yxwph7k71lfg)

[2. SOAP API (Simple Object Access Protocol) 5](#_y8p2d5cdacp6)

[SOAP API – Like a Bank Transaction 🏦 5](#_45o4ndajwxhx)

[3. GraphQL API 6](#_dy6hqb9g7bov)

[GraphQL API – Like a Restaurant Menu 🍽️ 6](#_615ksqeweth2)

[4. gRPC API (Google Remote Procedure Call) 6](#_8pbfu4ket5bo)

[gRPC API – Like a Phone Call 📞 6](#_p2tyqwbswehq)

[**Dissection of a REST API URL 7**](#_5jsderoaduwp)

[1. HTTP Protocol (http or https) 7](#_zi2tzn1rsih7)

[2. Base URL (baseUrl) 8](#_thz50hrct0t4)

[3. Controller (controller) 8](#_52yn3dm2or0)

[4. Action (action) 9](#_rgnptnq094rv)

[5. Parameter (parameter) 9](#_9frwis7swt6f)

[Path Parameter (Fetching Specific Data Directly in the URL) 10](#_2b4l111qrofc)

[Query Parameter (Filtering Data with Key-Value Pairs) 10](#_egr5amr5jieg)

[**API Endpoint 12**](#_nccb5crztikp)

[**API Gateway 13**](#_hvf2ahd8w7wt)

[Activities of an API Gateway 14](#_jwvnpxrm43ww)

[**Software System Architecture 15**](#_fjnm3hygql8s)

[Monolithic Architecture 🏛️ 15](#_tyoij0q51rdh)

[Client-Server Architecture 🖥️🔁📱 16](#_v3ujs4vaa0ga)

[Microservices Architecture 🏗️ 16](#_xlcuy8ww6y04)

[Event-Driven Architecture 🔄 16](#_5x2tv6k6301k)

[Serverless Architecture ☁️ 17](#_tsohstav3cew)

[**Different Types of APIs 17**](#_dp91d47u2ezn)

[1. Public API (Open API) 🌎 17](#_f17vlagie6q1)

[2. Partner API 🤝 17](#_trdg3zkaqbaj)

[3. Internal API (Private API) 🏢 18](#_f3da7lwp9cop)

[4. Composite API 🔄 18](#_yuyvkc38j5hr)

[5. Private API 🔒 18](#_a0mtoz11zkzm)

[**API HTTP Methods 19**](#_yofhhko9d2qi)

[**API Authorization Methods 21**](#_6npdyzldys8t)

[1. API Keys (Stateless) 🔑 21](#_xm65r3keip73)

[2. Basic Authentication (Base64) 🔐 21](#_i3rarabze7xy)

[3. Session-Based Authentication (Stateful) 🔄 22](#_1hithd2utvzo)

[4. Bearer Token / JWT (JSON Web Token) 🔑📜 22](#_lgi85tx2fxii)

[5. OAuth 2.0 (Token-Based Authorization) 🔄🔐 23](#_s9hdtjn9cite)

[6. AWS Signature (AWS SigV4) 🌐🔑 24](#_nivlzciuc8lp)

[**List of HTTP Status Codes & Their Meanings 24**](#_v8cstw3wu77l)

[✅ Success (2xx) 24](#_4r6zn9fgubje)

[🔀 Redirection (3xx) 24](#_xlm2fayhcr4h)

[❌ Client Errors (4xx) 25](#_ijjlvzhj2cik)

[💥 Server Errors (5xx) 25](#_6k5bjsl8h96d)

[**Key Terms in Postman & Their Real-Life Examples 26**](#_k9etbslt52uz)

[1. Environment 26](#_exhqvi5yzlv1)

[2. Token 31](#_frcmk0svgad0)

[3. Session Cookie 31](#_o8ox7rvpljyh)

[4. Base URL 31](#_a34yame50n44)

[5. Random Variables 32](#_fvt0iyq8dfld)

[6. Secret Authorization Key 32](#_o5msusuxrha8)

[7. Pre-Processor (Pre-Request Script) 33](#_bgi0g2j7xyfp)

[8. Post-Processor (Tests Tab in Postman) 33](#_dts3o5r2t0mb)

# 

# 

# 

# 

# 

# 

# 

# **API (Application Programming Interface)**

An **API (Application Programming Interface)** is a **software intermediary** that enables two applications to communicate with each other. It acts as a bridge, allowing different software systems to exchange data and functionality in a seamless manner.

## Why is an API Used?

APIs are commonly used to:

* Develop the **backend of a service** by enabling communication between a frontend and a database or server.
* Allow **third-party applications** to access specific functionalities without exposing the entire system.

### Examples of APIs in Action:

1. **Google/Facebook Login API** – Allows users to log in to websites or apps using their existing Google or Facebook credentials.
2. **Google Maps API** – Enables businesses to integrate maps and location services into their applications.
3. **SMS Provider API** – Used by apps to send automated SMS notifications, such as OTPs for verification.

# **Different Types of API based on Structure**

APIs can be categorized based on their **structure** or **architecture** into the following types:

## 1. REST API (Representational State Transfer)

* **Most commonly used API type** for web services.
* Follows a **stateless** architecture, meaning each request from the client must contain all necessary information.
* Uses **HTTP methods** (GET, POST, PUT, DELETE) for communication.
* Returns data mostly in **JSON or XML** format.
* **Example:** Twitter API, Google Maps API.

✅ **Real-Time Example:**

### REST API – Like Ordering Food Online 🍕

* You open a food delivery app and search for "Pizza."
* The app sends a request to the restaurant's backend (API).
* The API returns a list of available pizzas.
* You select one, place an order (POST request), and get a confirmation.

✅ Key Feature: Uses URLs and HTTP methods (GET, POST, etc.).

## 

## 2. SOAP API (Simple Object Access Protocol)

* A **protocol-based** API that follows strict standards.
* Uses **XML** for message formatting.
* Requires **higher security** and is often used in **banking and financial services**.
* Works with multiple transport protocols like **HTTP, SMTP, TCP**.
* **Example:** PayPal API, Banking Transaction APIs.

✅ **Real-Time Example:**

### SOAP API – Like a Bank Transaction 🏦

* You transfer money from one bank to another.
* The banking system follows strict security rules (SOAP format).
* Data is sent in XML format to ensure safety.
* Once verified, the transaction is completed.

✅ Key Feature: Highly secure, used in financial services.

## 

## 3. GraphQL API

* Developed by **Facebook**, provides more **flexibility** than REST.
* Clients can request **specific data fields**, reducing unnecessary data transfer.
* Uses **a single endpoint** instead of multiple endpoints like REST.
* **Example:** GitHub API, Shopify API.

✅ **Real-Time Example:**

### GraphQL API – Like a Restaurant Menu 🍽️

* Instead of getting the entire menu, you ask only for what you need (e.g., "Show me only vegetarian dishes").
* The server sends back only the requested items (not the entire menu).

✅ Key Feature: You request only the needed data, reducing unnecessary data transfer.

## 4. gRPC API (Google Remote Procedure Call)

* Developed by **Google**, designed for **high-performance** communication between microservices.
* Uses **Protocol Buffers (protobuf)** instead of JSON or XML, making it **faster**.
* Supports **bi-directional streaming**, enabling real-time communication.
* **Example:** Kubernetes API, Netflix internal APIs.

✅ **Real-Time Example:**

### gRPC API – Like a Phone Call 📞

* You and a friend talk back and forth in real-time (bi-directional streaming).
* Instead of waiting for each person to finish, you can speak simultaneously (faster interaction).
* This is useful for video calls, live chat, and streaming services.

✅ Key Feature: Super-fast and supports real-time communication.

# Dissection of a REST API URL

A REST API URL is structured as:

{http protocol}/{baseUrl}/{controller}/{action}/{parameter}

## 1. HTTP Protocol (http or https)

* **http (Hypertext Transfer Protocol)**
  + ✅ **Real-Time Example:**Defines how information is transmitted over the **World Wide Web**.
  + Provides a **set of rules** for communication between web browsers and servers.
  + **Default port:** 80.

**✅ Real-Time Example:**

* If you visit a website using HTTP, the communication between your browser and the server is not encrypted.
* Example Scenario:
  + You open<http://oldwebsite.com> in your browser.
  + Your request is sent in plain text, making it vulnerable to hacking.
  + Hackers can intercept and steal login credentials or personal data.
* **https (Hypertext Transfer Protocol Secure)**
  + A **secure** version of HTTP that encrypts data using **SSL/TLS**.
  + Ensures **safe communication** between the client and the server.
  + **Default port:** 443.

✅ **Real-Time Example:**

* If you log in to a bank website using HTTPS, all data is encrypted for security.
* Example Scenario:
  + You open<https://securebank.com>.
  + Your login details (username/password) are encrypted before being sent.
  + Even if hackers intercept the data, they cannot read it.

🔹 **Key Difference:** **HTTP is like sending a postcard (anyone can read it), while HTTPS is like sending a sealed letter (only the recipient can read it)**

## 2. Base URL (baseUrl)

* Represents the **main domain** or the **API endpoint**.

Example:  
  
<https://api.example.com>

* + It acts as the **entry point** to access different services of the API.

## 

## 3. Controller (controller)

A **Controller** in an API is like a **manager** that handles client requests and decides how to process them.  
It **connects** the front-end (user request) with the **backend services** (database, logic, etc.).

✅ **Example:**Imagine an **e-commerce website** 🛒.

* You have different **sections** like **Users, Products, and Orders**.
* Each section has its **own controller** to handle requests.

**API Structure Example:**

https://api.example.com/users

https://api.example.com/products

https://api.example.com/orders

Here,

* **users** → UserController
* **products** → ProductController
* **orders** → OrderController

## 

## 

## 

## 4. Action (action)

An **Action** is a specific **task** or **operation** that a Controller can perform. It defines **what should happen** when a client sends a request to an API.

✅ **Example Actions in a User Controller:**

| **Action** | **API Endpoint** | **Purpose** |
| --- | --- | --- |
| getUser | /users/getUser/123 | Fetch details of user with ID 123 |
| createUser | /users/create | Add a new user |
| updateUser | /users/update/123 | Update user with ID 123 |
| deleteUser | /users/delete/123 | Remove user with ID 123 |

🔹 Key Takeaways

📌 **Controller** = A **manager** for handling requests (like Users, Products, Orders).  
📌 **Action** = A **specific operation** (like getUser, updateUser, deleteUser).

📍 **Analogy:**

* **Controller** is like a department in a company (HR, Sales, Support).
* **Actions** are tasks that department performs (Hire Employee, Process Salary, Handle Complaints).

## 

## 5. Parameter (parameter)

A parameter is used in a REST API URL to specify which specific data you need. Instead of fetching all data, parameters help refine your request to get only the relevant information.

### 

Types of Parameters in REST API

1️⃣ **Path Parameter** (Used inside the URL path)  
2️⃣ **Query Parameter** (Used after a ? in the URL)

### Path Parameter (Fetching Specific Data Directly in the URL)

* Used to request data about a **specific item** by including it in the **URL path**.

Example: If you want details of a user **with ID 123**, the URL would be:  
 https://api.example.com/users/getUser/123

✅ **Real-Life Example:** **Amazon Product Page** 🛒

When you visit a product on Amazon, the URL might look like:  
 https://www.amazon.com/product/12345

* Here, 12345 is the product **ID**.
* Amazon’s API fetches **only that specific product's details** instead of all products.

### Query Parameter (Filtering Data with Key-Value Pairs)

* Used when you **filter, sort, or search for data**.
* Appears **after a ? in the URL**, and multiple parameters are **separated by &**.

Example: If you want to **search for a user named "John" who is 30 years old**, the URL would be:  
 https://api.example.com/users?name=John&age=30

📌 **Breakdown:**

* + https://api.example.com/users → **Base URL & Controller**
  + ?name=John → **Query Parameter (Search for "John")**
  + &age=30 → **Another Query Parameter (Filter users aged 30)**

✅ **Real-Life Example:** **Google Search** 🔍

When you search for "best smartphones under $500" on Google, the URL might look like:  
 <https://www.google.com/search?q=best+smartphones+under+500>

* Here, q=best+smartphones+under+500 is the **query parameter** that tells Google **what to search for**.

Key Differences Between Path & Query Parameters

| **Feature** | **Path Parameter (URL Path)** | **Query Parameter (?)** |
| --- | --- | --- |
| **Used for** | **Specific item** (ID, product, user) | **Filtering/searching data** |
| **Example** | /users/getUser/123 | users?name=John&age=30 |
| **Real-World Use** | Amazon product page | Google search |
| **Can it be multiple?** | No (Usually 1 value) | Yes (Separated by &) |

# API Endpoint

An API Endpoint is a specific URL (web address) where a client (user or system) can send a request to get or modify data from a server. **It is the "entry point" where two systems communicate**—one system makes a **request**, and the other provides a **response**.

📍 **Real-Life Example (Weather API ☀️🌧️)**

Let’s say you want to check the weather in **New York** using a Weather API.

**API Endpoint:**

https://api.weather.com/v1/current?city=NewYork

**Breakdown:**

* https://api.weather.com → **Base URL** (Main API address)
* /v1/current → **Endpoint** (Fetches current weather)
* ?city=NewYork → **Query Parameter** (Gets weather for New York)

✅ **Request:** "Tell me the weather in New York!"  
✅ **Response:** { "city": "New York", "temperature": "25°C", "condition": "Sunny" }

**🔹Key Takeaways**

📍 **API Endpoint** = A specific URL where requests are sent to interact with a system.  
📍 Every API has multiple endpoints for different actions (GET, POST, UPDATE, DELETE).  
📍 API Endpoints are like **doors** to access different services in a system.

💡 **Analogy:** Think of an API as a **restaurant**, and API Endpoints are **different menu items** you can order. 🍕🍔🥗

# API Gateway

An **API Gateway** is like a **traffic manager** that handles API requests, decides where they should go, and ensures everything runs smoothly.

* It acts as a **middleman** between clients (users) and backend services (databases, microservices, etc.).
* It ensures **efficient routing, security, and monitoring** of API requests.

📍 **Think of it as a receptionist in a company** 🏢.

* When visitors come (API requests), the receptionist (API Gateway) **directs them to the right department (backend services)**.
* The receptionist also **checks visitor ID (authentication), records visitor logs (monitoring), and controls access (security)**.

**Why Use an API Gateway?**

An API Gateway is important for **managing multiple APIs** efficiently. **Main Benefits:**

* **Request Routing** → Sends requests to the right backend service.
* **Security** → Controls authentication and access management.
* **Load Balancing** → Distributes traffic to avoid overload.
* **Rate Limiting** → Prevents excessive API requests from slowing down services.
* **Monitoring & Logging** → Tracks API usage and performance.
* **Protocol Translation** → Converts different request formats (e.g., HTTP → WebSocket).

**📌 How an API Gateway Works? (Real Example)**

Example: E-Commerce API Gateway 🛒

Imagine you have an **e-commerce platform** with different backend services:

| **Service** | **API Endpoint** | **Function** |
| --- | --- | --- |
| **User Service** | /users/login | Manages user authentication |
| **Product Service** | /products/list | Fetches product details |
| **Order Service** | /orders/create | Places an order |
| **Payment Service** | /payments/checkout | Processes payments |

📍 Without an API Gateway:

* The client has to **directly interact** with multiple APIs (User, Product, Order, Payment).
* Managing these connections becomes **complex** and **insecure**.

📍 With an API Gateway:

* The client **only calls the API Gateway** → /api/gateway
* The API Gateway **forwards requests to the correct backend service**.

**Example API Call (with Gateway):**

https://api.ecommerce.com/gateway/products/list

* The **API Gateway** receives the request.
* It forwards it to the **Product Service** (/products/list).
* The **Product Service** returns the data to the client.

## Activities of an API Gateway

| **Activity** | **Description** |
| --- | --- |
| **Protocol Translation** | Converts different API request formats (e.g., REST → gRPC) |
| **Basic Business Logic** | Handles request validation and response modification |
| **Authentication & Security** | Manages login, token verification, and access control |
| **Load Balancing** | Distributes traffic to prevent overload on one server |
| **Monitoring & Analytics** | Tracks API usage and logs errors for analysis |

📍 Real-World Example (Netflix API Gateway 🎬)

Netflix uses an **API Gateway** to manage millions of requests per second. When you open Netflix, the API Gateway:

* **Handles authentication** → Verifies user login.
* **Fetches personalized recommendations** → Calls the recommendation service.
* **Streams content** → Routes requests to video servers.

📍 **Without an API Gateway, users would have to connect to multiple APIs separately!**

🔹 Key Takeaways

**API Gateway** is like a **smart traffic manager** that efficiently routes API requests.It **improves security, scalability, and performance**.Major companies like **Netflix, Amazon, and Facebook** use API Gateways to handle millions of requests.

# 

# Software System Architecture

Software System Architecture is the high-level blueprint that defines how different components of a software system interact.

* It shows how **data flows** between different parts of the system.
* It includes **frontend, backend, databases, APIs, security, and infrastructure**.
* Helps developers, designers, and stakeholders **understand the system's structure** before building it.

Types of Software System Architectures

## Monolithic Architecture 🏛️

All components (UI, business logic, database) are bundled together in a **single codebase**.

* **Best For:** Small applications with simple logic.
* **Downside:** Difficult to scale; one failure can crash the entire system.

📍 **Example:**

* A **basic e-commerce website** where everything (user login, product catalog, orders) is in one application.

## Client-Server Architecture 🖥️🔁📱

Divides the system into:

* **Client:** Sends requests (Frontend – Web/App).
* **Server:** Processes requests and sends responses (Backend).
* **Best For:** Web applications, mobile apps.
* **Downside:** Server overload if too many users access it.

📍 **Example:**

* A **banking app** 🏦 where the app (client) requests transactions from the bank's servers.

## Microservices Architecture 🏗️

Breaks down an application into **independent, small services** that communicate via APIs.

* **Best For:** Large, scalable systems.
* **Downside:** Complex to manage multiple services.

📍 **Example:**

* **Netflix & Amazon** → Each service (authentication, video streaming, recommendations) runs separately.

## Event-Driven Architecture 🔄

Works on real-time events where components react to **triggers** instead of continuous requests.

* **Best For:** Real-time applications.
* **Downside:** Requires event management tools.

📍 **Example:**

* **Stock trading apps** 📈 where price updates happen instantly based on market events.

## 

## Serverless Architecture ☁️

No need to manage servers! Code runs on **cloud services** like AWS Lambda, Azure Functions.

* **Best For:** Lightweight applications & startups.
* **Downside:** Limited control over infrastructure.

📍 **Example:**

* **Chatbots, real-time notifications, image processing.**

🔹 Key Takeaways

📌 Software architecture defines how an application is structured.  
📌 Different architectures serve different use cases.  
📌 Modern applications use microservices or serverless for better scalability.

# Different Types of APIs

APIs (Application Programming Interfaces) can be classified into different types based on **accessibility and usage**.

## 1. Public API (Open API) 🌎

These APIs are **available to everyone** and can be accessed by developers without restrictions.

* **Best For:** Third-party integrations, open-source projects.
* **Downside:** Security concerns as anyone can use them.

📍 **Example:**

* **Google Maps API** → Developers use this to show maps in apps.
* **Weather API (OpenWeatherMap)** → Provides weather updates.

## 2. Partner API 🤝

These APIs are **shared with specific business partners** under agreements.

* **Best For:** Business collaborations, B2B services.
* **Downside:** Requires authentication and approval to access.

📍 **Example:**

* **Uber API** → Uber allows partners (e.g., restaurants) to integrate ride-booking.
* **PayPal API** → Used by e-commerce platforms to process payments.

## 3. Internal API (Private API) 🏢

APIs used **within an organization** to improve internal communication between services.

* **Best For:** Secure, internal software development.
* **Downside:** Not accessible to external developers.

📍 **Example:**

* **Banking Systems** → A bank's internal API to check user balances between departments.
* **HR Software API** → Used to manage employee details within a company.

## 4. Composite API 🔄

This API **combines multiple API calls** into one request, reducing response time.

* **Best For:** Speed optimization, batch processing.
* **Downside:** Can be complex to manage.

📍 **Example:**

* **E-Commerce Checkout API** → One request retrieves product details, user info, and applies discounts in one go.
* **Social Media Dashboard** → Fetches posts, comments, and likes in one response.

## 5. Private API 🔒

A **restricted** API used for internal purposes within an organization, offering the **highest security**.

* **Best For:** Internal business applications, security-sensitive tasks.
* **Downside:** Cannot be accessed externally.

📍 **Example:**

* **Netflix Backend API** → Manages user recommendations internally.
* **Finance APIs** → Used by banks to manage transactions securely.

🔹 Key Takeaways

📌 **Public APIs** are open and widely used in third-party integrations.  
📌 **Partner APIs** enable business collaborations securely.  
📌 **Internal & Private APIs** are strictly used inside organizations.  
📌 **Composite APIs** optimize performance by combining multiple requests.

# API HTTP Methods

**1. GET – Retrieve Data :** Used to fetch data from a server without modifying anything.  
**Example:** GET /users/123 → Fetch user details.

2. POST – Create a Resource : Used to add new data to the server.  
**Example:** POST /users → Create a new user.

3. PUT – Update Entire Resource: Replaces an existing resource with a new one.  
**Example:** PUT /users/123 → Update user completely.

4. PATCH – Partial Update: Modifies only specific fields of a resource.  
**Example:** PATCH /users/123 → Update just the email.

5. DELETE – Remove a Resource : Deletes a resource permanently.  
**Example:** DELETE /users/123 → Remove user.

6. HEAD – Retrieve Headers Only: Similar to GET but fetches only metadata (headers), not the response body.  
**Example:** HEAD /users/123 → Get metadata for user.

7. OPTIONS – Get Supported Methods: Used to check which HTTP methods are allowed on a resource.  
**Example:** OPTIONS /users → Returns allowed methods: GET, POST, DELETE.

8. CONNECT – Establish a Secure Connection: Used to establish a tunnel to the server, commonly used for **HTTPS proxies**.  
**Example:** CONNECT example.com:443 → Connect securely.

9. TRACE – Debugging Request Path: Returns the exact request sent to the server for debugging.  
**Example:** TRACE /users → Debug API request.

10. PURGE – Clear Cached Data: Used to forcefully remove cached content from a CDN or caching system.  
**Example:** PURGE /users/123 → Clear cache for user data.

11. LINK – Establish a Relationship: Used to associate one resource with another.  
**Example:** LINK /users/123/posts/456 → Link post 456 to user 123.

12. UNLINK – Remove a Relationship: Removes an established link between resources.  
**Example:** UNLINK /users/123/posts/456 → Unlink post 456 from user 123.

13. LOCK – Secure a Resource : Prevents modifications by locking a resource.  
**Example:** LOCK /document/123 → Lock document 123.

14. UNLOCK – Release a Locked Resource: Unlocks a previously locked resource.  
**Example:** UNLOCK /document/123 → Unlock document 123.

15. PROPFIND – Retrieve Properties : Fetches metadata about a resource, mainly used in WebDAV.  
**Example:** PROPFIND /files/document.pdf → Get file properties.

16. PROPPATCH – Update Properties: Modifies metadata of a resource.  
**Example:** PROPPATCH /files/document.pdf → Update file properties.

17. MKCOL – Create a Collection: Creates a new directory (collection) on a server.  
**Example:** MKCOL /photos/albums → Create a new album.

18. MOVE – Transfer a Resource 🚚

Moves a resource to a new location.  
**Example:** MOVE /photos/old.jpg TO /photos/new.jpg → Move file.

19. COPY – Duplicate a Resource : Copies a resource to another location.  
**Example:** COPY /photos/old.jpg TO /photos/backup.jpg → Copy file.

20. SEARCH – Query Data : Allows searching within resources.  
**Example:** SEARCH /users?q=John → Find users named "John".

# API Authorization Methods

APIs use authorization methods to ensure that only authenticated users or systems can access specific data or actions. Here’s a breakdown of the common API authorization techniques:

## 1. API Keys (Stateless) 🔑

* A **unique key** is assigned to each user/application.
* The client includes the API key in requests.
* **No session is maintained**, making it **stateless**.
* **Example Usage:** Public APIs, Google Maps API.

✔ **Pros:** Easy to use, widely adopted.  
❌ **Cons:** Can be shared or exposed if not secured properly.

**Example Request:**

GET /users HTTP/1.1

Host: api.example.com

Authorization: Api-Key 123456abcdef

## 2. Basic Authentication (Base64) 🔐

* Uses a **username and password** encoded in **Base64**.
* Sent in the Authorization header with each request.
* **Example Usage:** Simple authentication for internal APIs.

✔ **Pros:** Easy to implement.  
❌ **Cons:** Less secure (credentials sent in every request).

**Example Request:**

GET /users HTTP/1.1

Host: api.example.com

Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=

## 3. Session-Based Authentication (Stateful) 🔄

* User **logs in** and gets a **session ID (cookie)**.
* The session is **stored on the server** (stateful).
* **Example Usage:** Web applications with user logins.

✔ **Pros:** Good for web apps that require session tracking.  
❌ **Cons:** Not scalable for large distributed systems.

**Example Request:**

GET /dashboard HTTP/1.1

Host: api.example.com

Cookie: sessionid=abc123xyz

## 4. Bearer Token / JWT (JSON Web Token) 🔑📜

* A **signed token** (JWT) is issued after authentication.
* The client includes the **token** in the Authorization header.
* **Stateless**: No session is stored on the server.
* **Example Usage:** Mobile & web apps, OAuth-based APIs.

✔ **Pros:** Secure, scalable, no need to store sessions.  
❌ **Cons:** Cannot revoke tokens unless you store them in a database.

**Example JWT Token Format:**

{

"sub": "user123",

"iat": 1628500000,

"exp": 1628600000

}

**Example Request:**

GET /users HTTP/1.1

Host: api.example.com

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI...

## 5. OAuth 2.0 (Token-Based Authorization) 🔄🔐

* **Industry standard** for secure API access.
* Uses an **authorization server** to issue access tokens.
* **Example Usage:** Google, Facebook, GitHub API authentication.

✔ **Pros:** Secure, supports third-party authentication.  
❌ **Cons:** More complex than API keys or JWT.

**OAuth 2.0 Grant Types:**

* **Authorization Code Flow** → Best for web apps.
* **Implicit Flow** → Used for frontend apps (less secure).
* **Client Credentials Flow** → Used for machine-to-machine communication.
* **Resource Owner Password Flow** → Used in trusted environments.

**Example OAuth Flow:**1️⃣ User logs in via Google → 2️⃣ Google issues **access token** → 3️⃣ API verifies token → 4️⃣ User accesses resources.

**Example Request:**

GET /profile HTTP/1.1

Host: api.example.com

Authorization: Bearer abc123xyz

## 6. AWS Signature (AWS SigV4) 🌐🔑

* Used for **Amazon Web Services (AWS) APIs**.
* Signs requests using an **HMAC-SHA256 signature**.
* Ensures **authentication and integrity**.
* **Example Usage:** AWS S3, EC2, Lambda.

✔ **Pros:** Highly secure, prevents request tampering.  
❌ **Cons:** More complex setup.

**Example Signature Headers:**

Authorization: AWS4-HMAC-SHA256 Credential=accessKey/20240208/region/service/aws4\_request,

SignedHeaders=host;x-amz-date,

Signature=abcdef123456

🚀 **Tip:** OAuth 2.0 & JWT are the most **modern** and **secure** choices for most web applications!

# List of HTTP Status Codes & Their Meanings

## ✅ Success (2xx)

* 200 – OK → Request processed successfully.
* 201 – Created → New resource successfully created.
* 202 – Accepted → Request received but processing is pending.
* 203 – Non-Authoritative Information → Response received from another source.
* 204 – No Content → Successful request, but no content to return.
* 205 – Reset Content → Request processed, instructs client to reset view.
* 206 – Partial Content → Partial response for large requests.
* 207 – Multi-Status → Multiple status codes in one response.
* 208 – Already Reported → Resource already included in a response.
* 226 – IM Used → Server fulfilled the request using instance manipulations.

## 🔀 Redirection (3xx)

* 300 – Multiple Choices → Multiple options for the resource.
* 301 – Moved Permanently → Resource permanently moved to a new URL.
* 302 – Found → Resource temporarily moved.
* 303 – See Other → Redirect to another resource.
* 304 – Not Modified → Resource not changed since the last request.
* 305 – Use Proxy → Must access resource through a proxy.
* 306 – (Unused) → Previously used, now deprecated.
* 307 – Temporary Redirect → Temporary URL change, method remains the same.
* 308 – Permanent Redirect → Permanent move, method unchanged.

## ❌ Client Errors (4xx)

* 400 – Bad Request → Invalid request syntax.
* 401 – Unauthorized → Authentication required.
* 402 – Payment Required → Reserved for future use.
* 403 – Forbidden → Server refuses to fulfill the request.
* 404 – Not Found → Resource not found.
* 405 – Method Not Allowed → HTTP method not supported.
* 406 – Not Acceptable → Request format not acceptable.
* 407 – Proxy Authentication Required → Authentication required via proxy.
* 408 – Request Timeout → Client took too long to send a request.
* 409 – Conflict → Request conflicts with the current state of the resource.
* 410 – Gone → Resource permanently deleted.
* 411 – Length Required → Missing Content-Length header.
* 412 – Precondition Failed → Server conditions not met.
* 413 – Payload Too Large → Request entity is too big.
* 414 – URI Too Long → URL is too long for the server to process.
* 415 – Unsupported Media Type → Media type not supported.
* 416 – Range Not Satisfiable → Requested range cannot be fulfilled.
* 417 – Expectation Failed → Expect header conditions not met.
* 418 – I'm a Teapot → Easter egg (RFC 2324).
* 421 – Misdirected Request → Request sent to the wrong server.
* 422 – Unprocessable Entity → Request format correct, but invalid data.
* 423 – Locked → Resource is locked.
* 424 – Failed Dependency → Dependent request failed.
* 425 – Too Early → Prevent replay attacks.
* 426 – Upgrade Required → Client must switch protocols.
* 428 – Precondition Required → Preconditions required for request.
* 429 – Too Many Requests → Rate-limiting reached.
* 431 – Request Header Fields Too Large → Header fields are too large.
* 451 – Unavailable For Legal Reasons → Content blocked due to legal reasons.

## 💥 Server Errors (5xx)

* 500 – Internal Server Error → General server error.
* 501 – Not Implemented → Feature not supported.
* 502 – Bad Gateway → Server received an invalid response from upstream.
* 503 – Service Unavailable → Server overloaded or under maintenance.
* 504 – Gateway Timeout → Upstream server did not respond in time.
* 505 – HTTP Version Not Supported → HTTP version not supported.
* 506 – Variant Also Negotiates → Content negotiation loop.
* 507 – Insufficient Storage → Server lacks space.
* 508 – Loop Detected → Infinite request loop detected.
* 510 – Not Extended → Extensions required for processing.
* 511 – Network Authentication Required → Authentication required to access the network.

Most Popular API HTTP Response Codes

| **Category** | **Status Codes** | **Usage** |
| --- | --- | --- |
| ✅ **Success** | 200, 201, 202, 204, 208 | When a request is successfully processed. |
| 🔀 **Redirection** | 301, 302, 308 | When resources have moved. |
| ❌ **Client Errors** | 400, 401, 403, 404, 405, 413, 415, 429 | When the client sends an invalid request. |
| 💥 **Server Errors** | 500, 502, 503, 504, 507, 508 | When something goes wrong on the server side. |

# Key Terms in Postman & Their Real-Life Examples

## 1. Environment

* An environment in Postman allows you to define variables that can be reused across different requests.
* Instead of hardcoding values, you can create variables like base URLs, API keys, and tokens to make testing flexible.

🔹 Example:

* You have Development, Staging, and Production environments.Instead of manually changing the API URL for each environment, you can create an environment variable for {{base\_url}} and switch between them.

📌 Real-life Example:

* Similar to how you switch Wi-Fi networks based on location (home, office, café), Postman environments allow you to switch API setups easily.

**Types of Environments in Postman**

| **Type** | **Description** | **Example Use Case** |
| --- | --- | --- |
| **Local Environment** | Variables specific to an individual Postman request or session. | Testing a new API on your machine. |
| **Global Environment** | Variables accessible across all requests, collections, and workspaces. | Setting API keys or common headers used in all requests. |
| **Workspace Environment** | Variables shared within a specific Postman workspace. | When working on a project with a team. |
| **Collection-Level Variables** | Variables defined within a specific collection (applies only to that collection). | Running a set of related API tests. |

**Environment Variables in Postman**

**1. Global Variables**

* Available everywhere in Postman (across all environments).
* Used for API keys, authentication tokens, and frequently used constants.
* Global variables take priority over environment variables**.**
* Used in the Authorization header across multiple requests.

🔹 **Example:**

{

"global\_auth\_token": "abcd1234"

}

**📌 Real-life Example:**

* **Your email address**: You use it for multiple logins, whether it’s Gmail, Facebook, or Twitter.

**2. Environment Variables**

* Specific to an environment (e.g., Development, Staging, Production).
* Allows switching between API setups easily.

🔹 **Example:**

{{base\_url}} variable for different environments:  
  
{

"base\_url": "https://api-dev.example.com"

}

* This **avoids manually changing URLs** in API calls.

📌 **Real-life Example:**

* **Switching between different Wi-Fi networks** at home, office, and public places.

**3. Collection-Level Variables**

* Variables **limited to a specific collection** (API group).
* Useful for **scoping variables to related API requests**.

🔹 **Example:**

* Setting {{user\_id}} for all requests in a **User Management API collection**.

📌 **Real-life Example:**

* **Project-based logins** (one set of credentials for school projects, another for office work).

**4. Local Variables**

* **Temporary variables** valid only in a single request or script execution.
* These **override global and environment variables** but are NOT stored.

🔹 **Example:**

var temp\_token = "xyz987";

pm.environment.set("auth\_token", temp\_token);

📌 **Real-life Example:**

* **Disposable email addresses** used for temporary sign-ups.

**Using Variables in Postman**

You can use **double curly braces** {{variable\_name}} to refer to a variable.

🔹 Example:

GET {{base\_url}}/users/{{user\_id}}

* **base\_url = "https://api-dev.example.com"**
* **user\_id = "12345"**

Resulting request:

GET https://api-dev.example.com/users/12345

📌 **Real-life Example:**

* **Filling in placeholders in a document template** (e.g., "Dear {{name}}, welcome to {{company}}!").

**Setting Variables via Scripts (Pre-Request & Tests Tab)**

You can dynamically set variables using **JavaScript scripts** in Postman.

🔹 Setting Environment Variables (Pre-Request Script)

javascript

CopyEdit

pm.environment.set("auth\_token", "abcd1234");

📌 **Real-life Example:**

* **Getting a token before logging into a website.**

**Retrieving Environment Variables**

var token = pm.environment.get("auth\_token");

console.log("Current Token:", token);

**Clearing or Removing Variables**

🔹 Remove an Environment Variable

pm.environment.unset("auth\_token");

📌 **Real-life Example:**

* **Logging out from a device to remove saved credentials.**

**Priority Order of Variables**

When multiple variables have the same name, **Postman resolves them in the following order**:

1. **Local Variables (highest priority)**
2. **Data Variables (from Collection Runner input files)**
3. **Environment Variables**
4. **Collection Variables**
5. **Global Variables (lowest priority)**

📌 **Real-life Example:**

* **If you have multiple credit cards, you first use your local wallet (Local), then look in your bag (Global) if it’s missing.**

Best Practices for Using Environments in Postman

✅ **Use separate environments** for **Development, Staging, and Production**.  
✅ **Keep sensitive information (API Keys, Auth Tokens) in environment variables**, not in request URLs.  
✅ **Use scripts to update dynamic values** like tokens, timestamps, and session cookies.  
✅ **Avoid hardcoding values**—always use {{variable\_name}}.

## 2. Token

* A **token** is an authentication key used to access secured APIs.
* Some APIs require a **Bearer Token, JWT, or API Key** in the request header.

🔹 **Example:**

* A login API returns an authentication **JWT Token**, which must be sent in every request for authorization.

📌 **Real-life Example:**

* When logging into **Netflix**, your session remains active using a token until you log out or it expires.

## 3. Session Cookie

* **Session cookies** store short-term user authentication details for APIs.
* Used for maintaining a user's **login state** without requiring a token for every request.

🔹 **Example:**

* When you log in to an API, it returns a **session cookie** that keeps your session active across multiple requests.

📌 **Real-life Example:**

* When you **order food online**, your cart stays saved even if you visit another page—this is managed by cookies.

## 4. Base URL

* The **base URL** is the common part of all API endpoints in a system.
* It avoids redundancy when calling multiple API endpoints.

🔹 **Example:**

* Instead of writing full URLs repeatedly, you use a base URL like:
  + https://api.example.com/
  + Then, append endpoints like:
    - /users → To get user details
    - /products → To fetch product data

## 5. Random Variables

* Random variables are generated dynamically in Postman for testing purposes.
* Examples: {{$randomInt}}, {{$randomEmail}}, {{$randomUUID}}.

🔹 **Example:**

* When testing user registration, you don’t want to **manually enter unique email addresses** every time.
* You can use:
  + {{$randomEmail}} → Generates a new email each time the request runs.

📌 **Real-life Example:**

* When filling out a **test survey**, fake/random data is used instead of real user data.

## 6. Secret Authorization Key

* Some APIs require a **Secret Key** for secure communication.
* Unlike public API keys, secret keys **must not be exposed** (used for sensitive actions like payments).

🔹 **Example:**

* Authorization: Bearer <SECRET\_KEY>
* Used for services like **Stripe, PayPal, and AWS**.

📌 **Real-life Example:**

* Similar to your **ATM PIN**, which should never be shared, as it provides access to your bank account.

## 7. Pre-Processor (Pre-Request Script)

* A **pre-request script** runs before an API request is sent.
* Used for setting up headers, tokens, or dynamically generating data.

🔹 **Example:**

Generate a **timestamp** before making a request:  
  
pm.environment.set("current\_timestamp", new Date().toISOString());

📌 **Real-life Example:**

* Before taking an **online exam**, you **sign in and verify identity** before starting the test.

## 8. Post-Processor (Tests Tab in Postman)

* A **post-processor** script runs **after** an API response is received.
* Used for validation, logging, and extracting response values.

🔹 **Example:**

Check if an API response status is 200 OK:  
  
pm.test("Status code is 200", function () {

pm.response.to.have.status(200);

});