# REST API

\*\*REST (Representational State Transfer)\*\* is an architectural style for designing networked applications. It relies on a stateless, client-server, cacheable communications protocol — most commonly, HTTP. In a REST API, resources are identified by URLs, and the interaction with these resources is done using standard HTTP methods like GET, POST, PUT, DELETE, etc.

## Key Principles of REST:

1. Stateless: Each request from a client to a server must contain all the information needed to understand and process the request. The server does not store any state about the client session on the server between requests.

2. Client-Server Architecture: The client and server are independent; the client interacts with the server using an API, and the server provides resources as responses.

3. Cacheable: Responses must define themselves as cacheable or non-cacheable, so clients and intermediaries can store responses for reuse, improving efficiency.

4. Layered System: The architecture can be composed of hierarchical layers, where each layer has a specific function (e.g., security, load balancing, etc.).

5. Uniform Interface: REST APIs follow a consistent and uniform interface, typically based on HTTP methods. This uniformity allows independent evolution of different parts of the architecture.

6. Resource Representation: Resources are represented in various formats such as JSON, XML, or HTML, but JSON is the most commonly used.

## Common HTTP Methods in REST:

- \*\*GET\*\*: Retrieve a resource.

- \*\*POST\*\*: Create a new resource.

- \*\*PUT\*\*: Update an existing resource.

- \*\*DELETE\*\*: Delete a resource.

- \*\*PATCH\*\*: Partially update a resource.

# Types of APIs

APIs (Application Programming Interfaces) can be categorized based on their scope, usage, and the protocol they use. Below are the common types:

1. \*\*REST API\*\*:

- \*\*Protocol\*\*: HTTP

- \*\*Description\*\*: REST APIs are web-based APIs that conform to the principles of REST architecture. They are used to interact with web services and are stateless, allowing for easy scalability. They typically use JSON for data representation but can use other formats as well.

2. \*\*SOAP API\*\*:

- \*\*Protocol\*\*: HTTP, SMTP, TCP, etc.

- \*\*Description\*\*: SOAP (Simple Object Access Protocol) is a protocol that defines a set of rules for structuring messages. Unlike REST, SOAP is a protocol itself and is more rigid, requiring messages to follow a strict XML structure. It’s commonly used in enterprise environments for its robust security and transaction compliance features.

3. \*\*GraphQL API\*\*:

- \*\*Protocol\*\*: HTTP

- \*\*Description\*\*: GraphQL is a query language for APIs that allows clients to request only the data they need, reducing the amount of data transferred over the network. It provides more flexibility than REST by allowing clients to specify exactly what data they want.

4. \*\*gRPC\*\*:

- \*\*Protocol\*\*: HTTP/2

- \*\*Description\*\*: gRPC (Google Remote Procedure Call) is a high-performance RPC (Remote Procedure Call) framework that uses HTTP/2 for transport, Protocol Buffers as the interface definition language, and supports multiple programming languages. It’s efficient for connecting services in microservices architecture.

5. \*\*WebSocket API\*\*:

- \*\*Protocol\*\*: WebSocket

- \*\*Description\*\*: WebSocket is a communication protocol that provides full-duplex communication channels over a single TCP connection. WebSocket APIs allow for real-time, bidirectional communication between a client and a server, making it suitable for applications like chat apps and live updates.

6. \*\*OpenAPI/Swagger\*\*:

- \*\*Protocol\*\*: Not tied to a specific protocol

- \*\*Description\*\*: OpenAPI is a specification for defining RESTful APIs, while Swagger is a suite of tools that work with the OpenAPI specification to help design, build, document, and consume RESTful APIs. They are not a type of API themselves but are used to standardize and document RESTful APIs.

7. \*\*JSON-RPC/XML-RPC\*\*:

- \*\*Protocol\*\*: HTTP

- \*\*Description\*\*: These are remote procedure call (RPC) protocols encoded in JSON or XML, respectively. They allow a client to call methods on a server and receive the results. They are simpler than SOAP and provide a lightweight communication method, but they are less commonly used than REST.

8. \*\*Local APIs\*\*:

- \*\*Protocol\*\*: Varies (e.g., D-Bus, COM)

- \*\*Description\*\*: Local APIs are used to interact with services on the same machine, typically within an operating system or between applications on the same system. They enable deep integration with the host system or application.

# Summary

- \*\*REST APIs\*\* are popular for their simplicity, scalability, and use of standard HTTP methods.

- \*\*SOAP APIs\*\* offer strong security and transactional features but are more complex.

- \*\*GraphQL APIs\*\* provide flexibility in querying, allowing clients to request exactly what they need.

- \*\*gRPC APIs\*\* are efficient for microservices, supporting multiple languages and high performance.

- \*\*WebSocket APIs\*\* enable real-time, bidirectional communication.

- \*\*OpenAPI/Swagger\*\* standardizes RESTful APIs.

- \*\*JSON-RPC/XML-RPC\*\* offers a lightweight RPC method.

- \*\*Local APIs\*\* are used for system-level or inter-application communication on the same machine.

# HTTP Response Coade:

**2xx: Success**

1. **200 OK**: The request was successful, and the response contains the requested resource.
2. **201 Created**: The request was successful, and a new resource was created as a result.
3. **204 No Content**: The request was successful, but there is no content to send in the response.

**3xx: Redirection**

1. **301 Moved Permanently**: The resource has been moved to a new URL permanently.
2. **302 Found**: The resource has been temporarily moved to a different URL.
3. **304 Not Modified**: The resource has not been modified since the last request, so the client can use the cached version.
4. **307 Temporary Redirect**: The resource is temporarily located at a different URL, and the same method should be used to access it.

**4xx: Client Errors**

1. **400 Bad Request**: The server could not understand the request due to invalid syntax.
2. **401 Unauthorized**: Authentication is required, and the request was not provided with valid credentials.
3. **403 Forbidden**: The client does not have permission to access the resource.
4. **404 Not Found**: The requested resource could not be found on the server.
5. **405 Method Not Allowed**: The request method is not allowed for the requested resource.
6. **408 Request Timeout**: The server timed out waiting for the request from the client.
7. **429 Too Many Requests**: The user has sent too many requests in a given amount of time ("rate limiting").

**5xx: Server Errors**

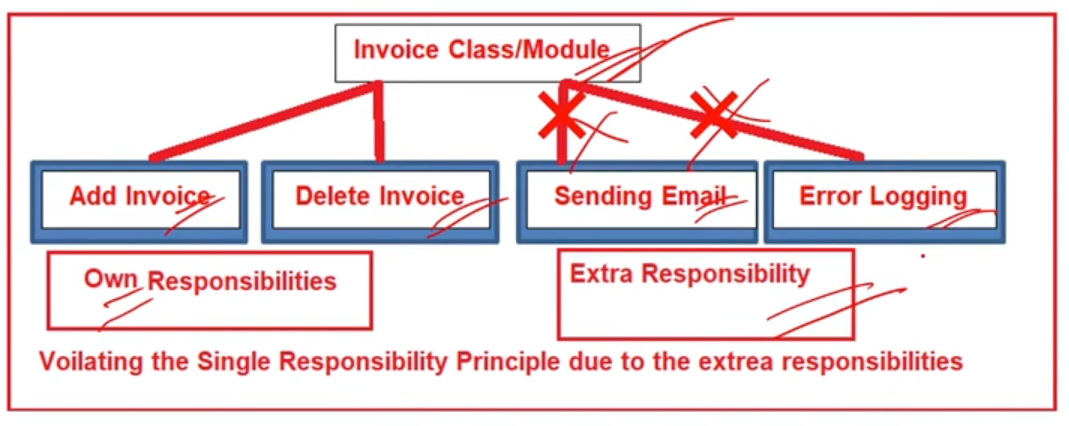
1. **500 Internal Server Error**: The server encountered an unexpected condition that prevented it from fulfilling the request.
2. **502 Bad Gateway**: The server received an invalid response from an inbound server it accessed while attempting to fulfill the request.
3. **503 Service Unavailable**: The server is currently unable to handle the request due to temporary overloading or maintenance.
4. **504 Gateway Timeout**: The server acted as a gateway or proxy and did not receive a timely response from the upstream server.

Top of Form

# SOLID Principals

## **S -** Single Responsibility Principle (SRP):

* **SRP** states that a class should have only one responsibility.
* A Class should have only one reason to change.
* When a class has only one responsibility, it becomes easier to change and test. If a class has responsibilities, changing one responsibility may impact others and more testing efforts will be required then.



## **O** - Open Closed Principal (OCP):

* **OCP** states that software entities (classes, modules) should be open for extension, but closed for modification.
* SRPis prerequisite for OCP.
* The benefit is simple testing is required to test individual classes, but if you will keep on adding and modifying in one class. Then even for the smallest modification, the whole class needs to be tested.

## **L** – Liskov Substitution Principal (LSP):

* **LSP** states that an object of a child class must be able to replace an object of the parent class without breaking the application.
* The Methods in base class must be applicable in the derived class.

## **I** - Interface Segregation Principle (ISP):

* **ISP** states that a class should not be forced to implement interface that it does not use.
* It is better to have multiple smaller interfaces than Larger Interfaces.

## D – Dependency Inversion Principle (DIP):

* **DIP** states that a High Level Class must not depend upon a lower level Class.
* E.g ILogger injection .

Bottom of Form