ABES
Estd. 2000
ABES
Engineering College
College Code-032

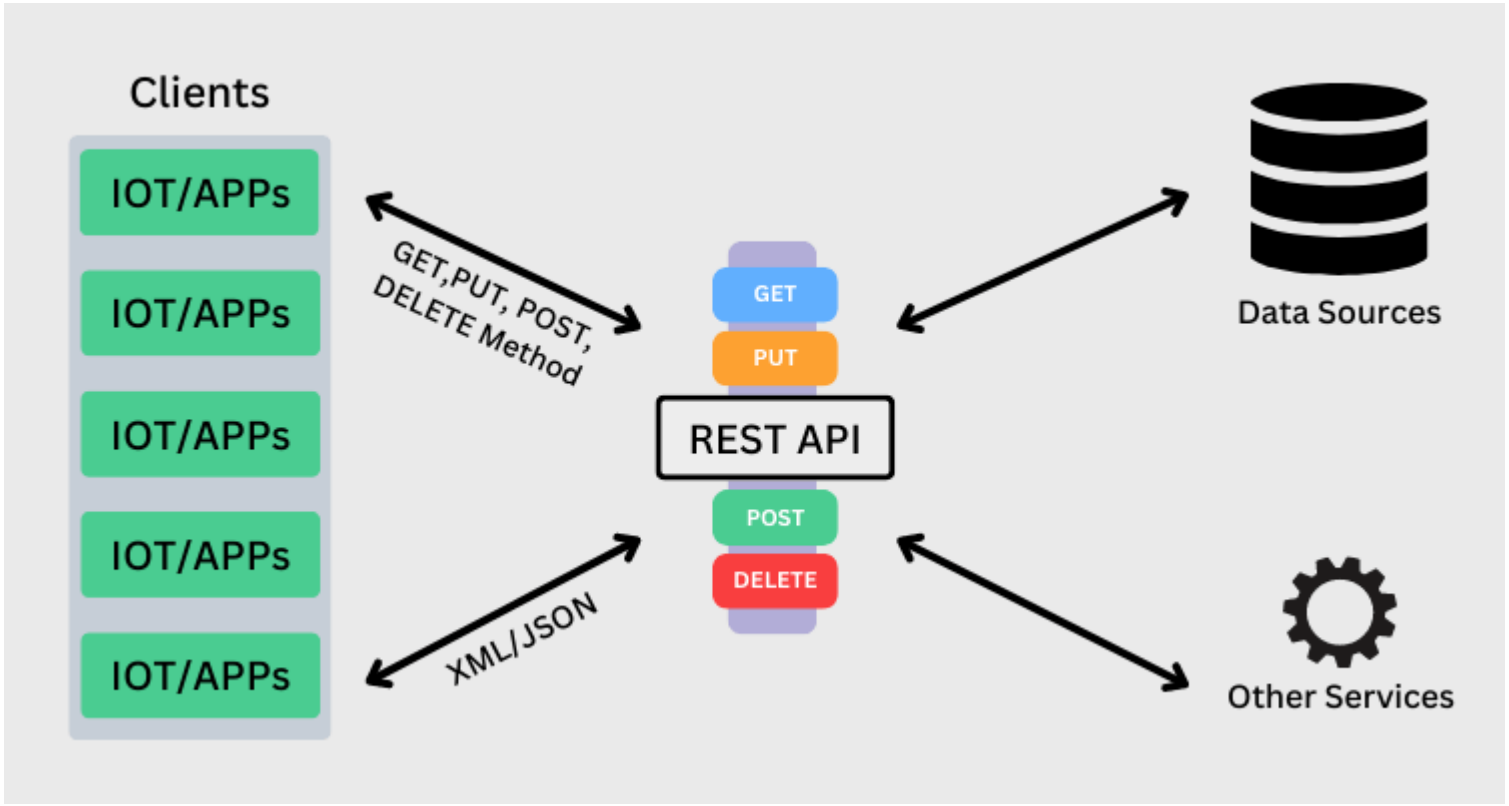
REST API
and
Web Architecture

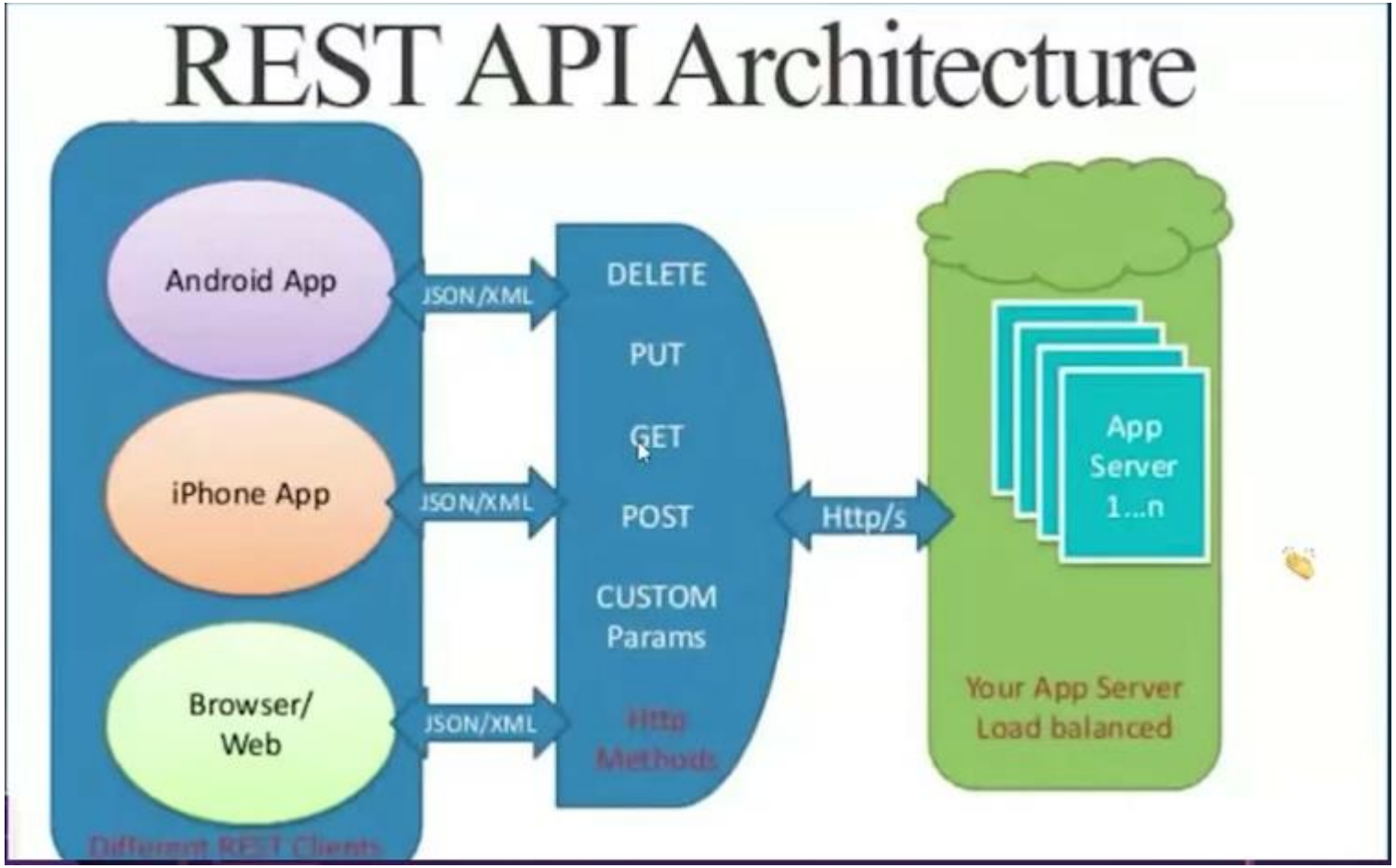
General Guideline

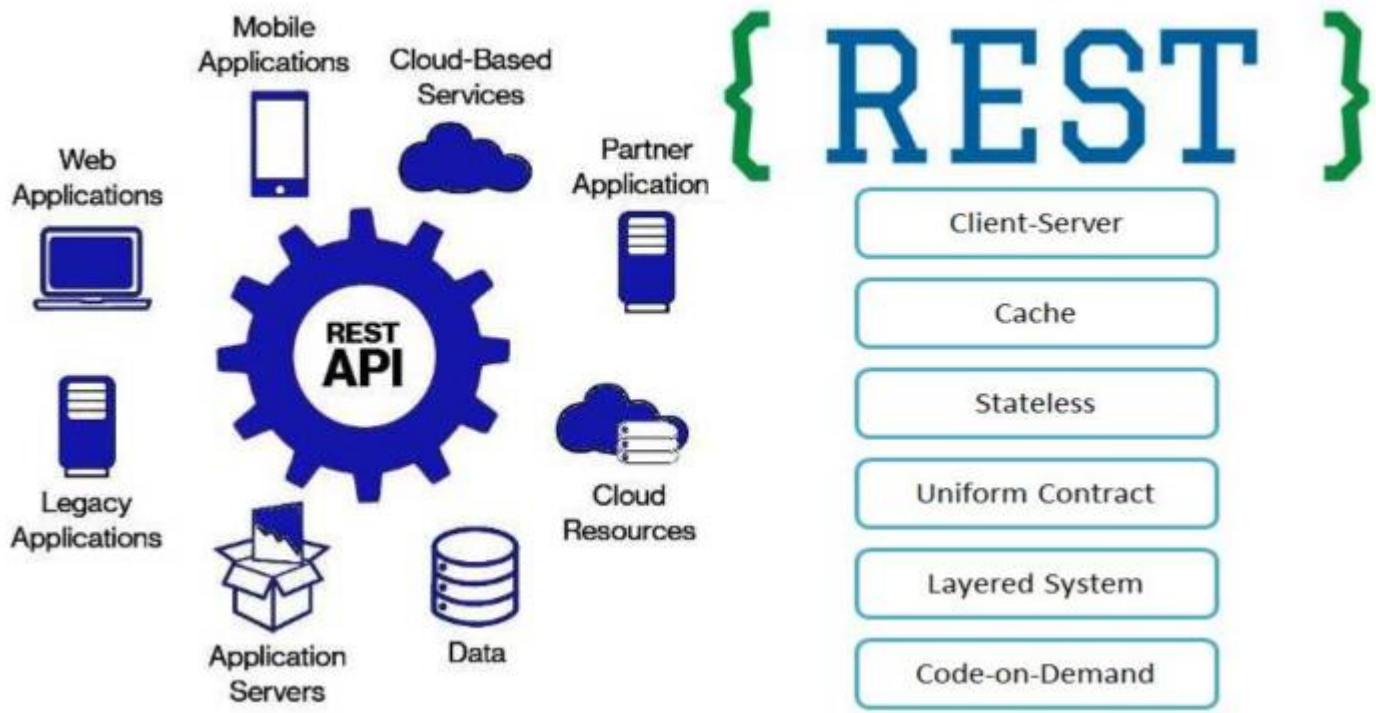
Table of Content

REST architecture

- SOAP and REST are two internet data exchange mechanisms.
- For example, imagine that your internal accounts system shares data with your customer's accounting system to automate invoicing tasks.
- The two applications share data by using an API that defines communication rules. SOAP and REST are two different approaches to API design.
- The SOAP approach is highly structured and uses XML data format.
- REST is more flexible and allows applications to exchange data in multiple formats.







Rest Advantages

- Scalability
- Simplicity
- Modifiability
- Reliability
- Portability
- Visibility

- To build applications, you can use many different programming languages, architectures, and platforms.
- It's challenging to share data between such varied technologies because they have different data formats.
- Both SOAP and REST emerged in an attempt to solve this problem.
- You can use SOAP and REST to build APIs or communication points between diverse applications.
- The terms *web service* and *API* are used interchangeably.
- However, APIs are the broader category. Web services are a special type of API.

SOAP API

SOAP is a protocol that defines rigid communication rules. It has several associated standards that control every aspect of the data exchange. For example, here are some standards SOAP uses:

- Web Services Security (WS-Security) specifies security measures like using unique identifiers called tokens
- Web Services Addressing (WS-Addressing) requires including routing information as metadata
- WS-Reliable Messaging standardizes error handling in SOAP messaging
- Web Services Description Language (WSDL) describes the scope and function of SOAP web services

When you send a request to a SOAP API, you must wrap your HTTP request in a SOAP envelope. This is a data structure that modifies the underlying HTTP content with SOAP request requirements. Due to the envelope, you can also send requests to SOAP web services with other transport protocols, like TCP or Internet Control Message Protocol (ICMP). However, SOAP APIs and SOAP web services always return XML documents in their responses.

Rest API

REST is a software architectural style that imposes six conditions on how an API should work. These are the six principles REST APIs follow:

- *Client-server architecture*. The sender and receiver are independent of each other regarding technology, platforming, programming language, and so on.
- *Layered*. The server can have several intermediaries that work together to complete client requests, but they are invisible to the client.
- *Uniform interface*. The API returns data in a standard format that is complete and fully useable.
- *Stateless*. The API completes every new request independently of previous requests.
- *Cacheable*. All API responses are cacheable.
- *Code on demand*. The API response can include a code snippet if required.

You send REST requests using HTTP verbs like *GET* and *POST*. Rest API responses are typically in JSON but can also be of a different data format.

Uniform interface

The following four constraints can achieve a uniform REST interface:

- Identification of resources – The interface must uniquely identify each resource involved in the interaction between the client and the server.
- Manipulation of resources through representations – The resources should have uniform representations in the server response. API consumers should use these representations to modify the resource state in the server.
- Self-descriptive messages – Each resource representation should carry enough information to describe how to process the message. It should also provide information of the additional actions that the client can perform on the resource.
- Hypermedia as the engine of application state – The client should have only the initial URI of the application. The client application should dynamically drive all other resources and interactions with the use of hyperlinks.

Uniform interface(Cont ...)

- In simpler words, REST defines a consistent and uniform interface for interactions between clients and servers. For example, the HTTP-based REST APIs make use of the standard HTTP methods (GET, POST, PUT, DELETE, etc.) and the URIs (Uniform Resource Identifiers) to identify resources.

Client-server

- The client-server design pattern enforces the separation of concerns, which helps the client and the server components evolve independently.
- By separating the user interface concerns (client) from the data storage concerns (server), we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
- While the client and the server evolve, we have to make sure that the interface/contract between the client and the server does not break.

Statelessness

- Statelessness mandates that each request from the client to the server must contain all of the information necessary to understand and complete the request.
- The server cannot take advantage of any previously stored context information on the server.
- For this reason, the client application must entirely keep the session state.

Cacheability

- The cacheable constraint requires that a response should implicitly or explicitly label itself as cacheable or non-cacheable.

- If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period.

Layered system

- The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior.
- In a layered system, each component cannot see beyond the immediate layer they are interacting with.
- A layman's example of a layered system is the *MVC pattern*.
- The MVC pattern allows for a clear separation of concerns, making it easier to develop, maintain, and scale the application.

Code on demand(optional)

- REST also allows client functionality to extend by downloading and executing code in the form of applets or scripts.

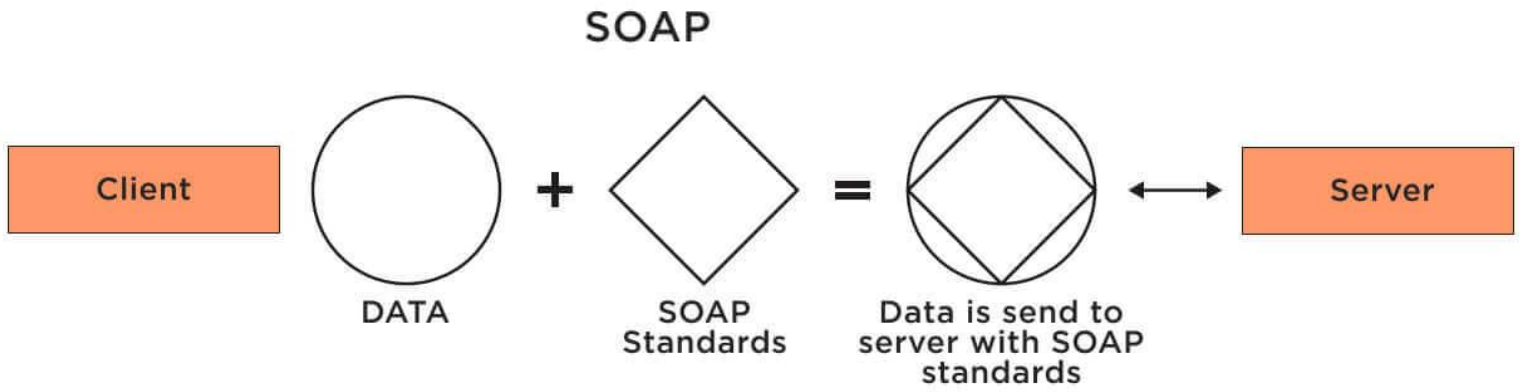
- The downloaded code simplifies clients by reducing the number of features required to be pre-implemented. Servers can provide part of features delivered to the client in the form of code, and the client only needs to execute the code.

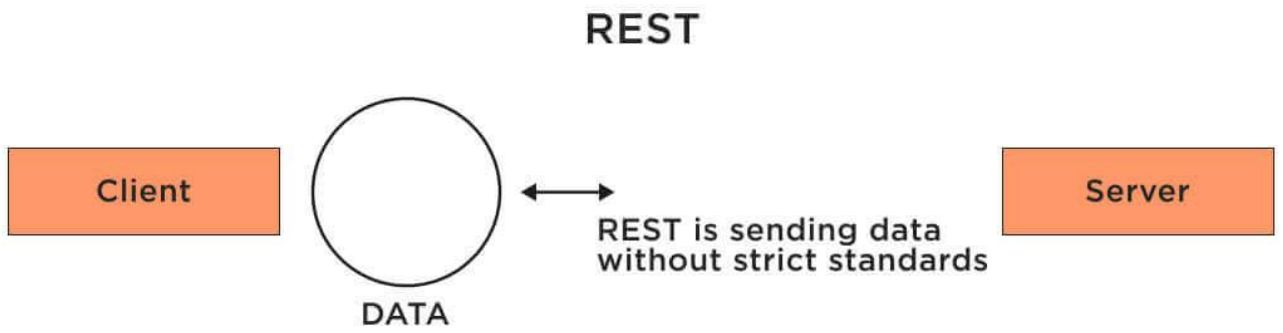
similarities between SOAP and REST

- They both describe rules and standards on how applications make, process, and respond to data requests from other applications
- They both use HTTP, the standardized internet protocol, to exchange information
- They both support SSL/TLS for secure, encrypted communication
- You can use either SOAP or REST to build secure, scalable, and fault-tolerant distributed systems.

Key differences: SOAP vs REST

	SOAP	REST
Stands for	Simple Object Access Protocol	Representational State Transfer
What is it?	SOAP is a protocol for communication between applications	REST is an architecture style for designing communication interfaces.
Design	SOAP API exposes the operation.	REST API exposes the data.
Transport Protocol	SOAP is independent and can work with any transport protocol.	REST works only with HTTPS.
Data format	SOAP supports only XML data exchange.	REST supports XML, JSON, plain text, HTML.
Performance	SOAP messages are larger, which makes communication slower.	REST has faster performance due to smaller messages and caching support.
Scalability	SOAP is difficult to scale. The server maintains state by storing all previous messages exchanged with a client.	REST is easy to scale. It's stateless, so every message is processed independently of previous messages.
Security	SOAP supports encryption with additional overheads.	REST supports encryption without affecting performance.
Use case	SOAP is useful in legacy applications and private APIs.	REST is useful in modern applications and public APIs.



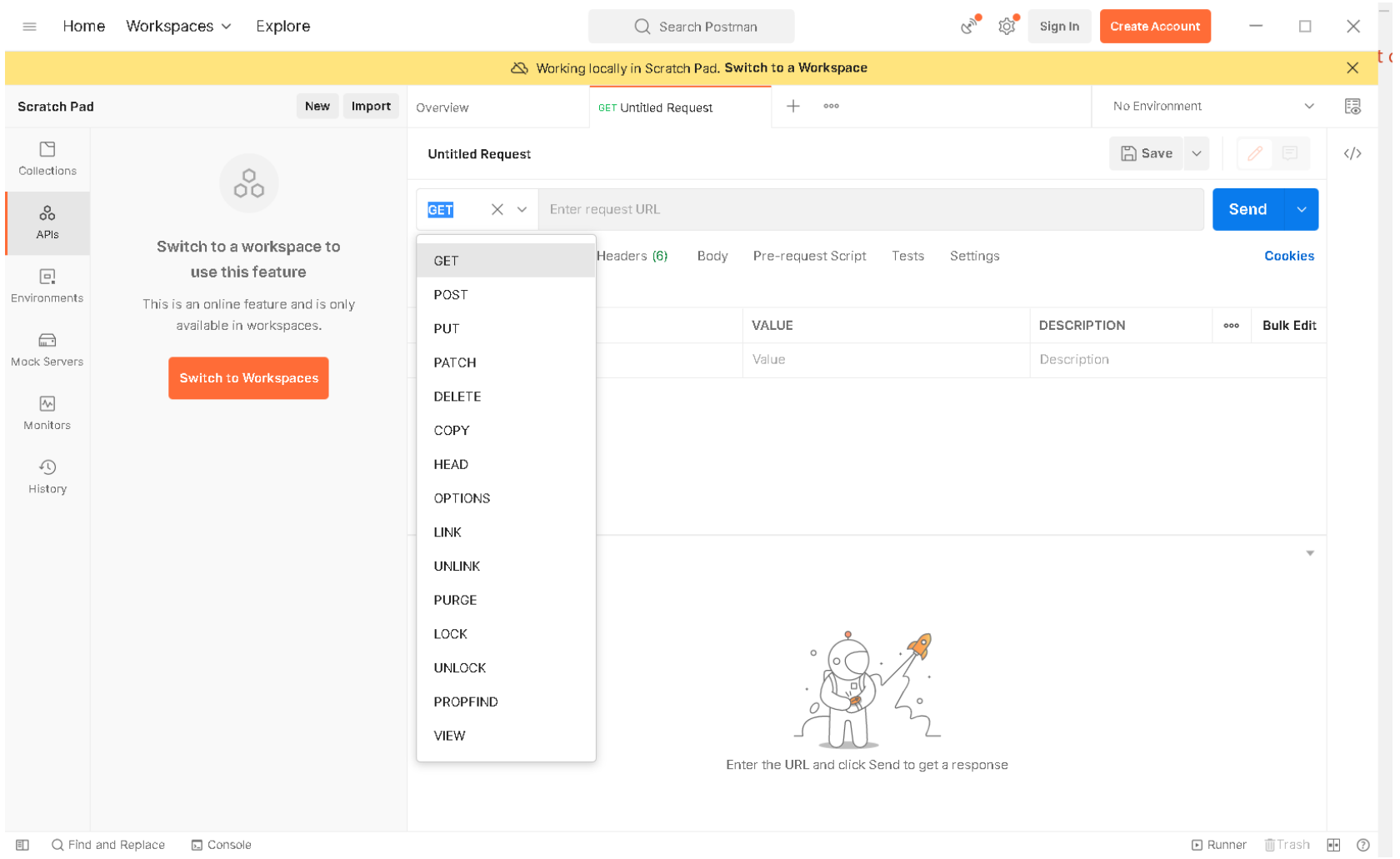


Jelvix

Source: https://media.geeksforgeeks.org/wp-content

jelvix.com

Postman API Development Tool



JSON Based user Data

```
{
   "user1" : {
      "name" : "mahesh",
      "password" : "password1",
      "profession" : "teacher",
      "id": 1
   },

   "user2" : {
      "name" : "suresh",
      "password" : "password2",
      "profession" : "librarian",
      "id": 2
   },

   "user3" : {
      "name" : "ramesh",
      "password" : "password3",
      "profession" : "clerk",
      "id": 3
   }
}
```

CRUD API for User

Sr.No.	URI	HTTP Method	POST body	Result
1	/	GET	empty	Show list of all the users.
2	/	POST	JSON String	Add details of new user.
3	/:id	DELETE	JSON String	Delete an existing user.
4	/:id	GET	empty	Show details of a user.
5	/:id	PUT	JSON String	Update an existing user

REST API authentication

Four common authentication methods include:

HTTP authentication

HTTP provides authentication schemes for REST API implementation. Two common schemes are:

- **Basic authentication:** HTTP basic authentication (BA) is a simple technique for controlling access to web resources. It doesn't require cookies, session identifiers, or login pages. Instead, it uses standard fields in the HTTP header.
- **Bearer authentication:** Bearer authentication, also known as token authentication, is an HTTP authentication scheme that involves the use of bearer tokens for security.

API keys

- One way to authenticate REST APIs is with API keys. When a client connects to a server for the first time, it is given a unique identifier. This unique API key is then utilized for authentication on every subsequent request to retrieve resources. It’s important to note that API keys have security risks because they must be transmitted with each request and can therefore be intercepted.

OAuth

- OAuth is a security protocol that offers highly secure login access to any system by combining passwords and tokens. The authorization process starts with the server requesting a password, followed by an additional token to complete the process.

REST API security

Even though RESTful APIs provide a simpler way to access and manipulate your application, security issues can still happen. For example, a client can send thousands of requests every second and crash your server. Other REST API security challenges include:

- Lack of proper authentication
- Absence of rate limiting and throttling
- Failure to encrypt payload data
- Incorrect implementation of HTTPS
- Weak API keys that are easily compromised

Common HTTP Code

Common HTTP status codes include:

- 200 OK: Indicates that the request has succeeded and the response body contains the requested resource.
- 404 Not Found: Indicates that the server cannot find the requested resource, often due to a mistyped URL or a resource that has been moved or deleted.
- 400 Bad Request: Signifies that the server cannot process the request due to a client error, such as invalid syntax or missing required parameters in the request.
- 500 Internal Server Error: Indicates that the server encountered an unexpected condition that prevented it from fulfilling the request, typically a server-side issue beyond the client's control.
- 302 Found (Moved Temporarily): Indicates that the requested resource has been temporarily moved to a different URL. The client should issue another request to the new URL provided in the response header.
- 401 Unauthorized: Signifies that the request requires user authentication, but the client has not provided valid credentials or has not yet authenticated.

Web Architecture

- Web architecture is the conceptual structure of the World Wide Web. The WWW or internet is a constantly changing medium that enables communication between different users and the technical interaction between different systems and subsystems.

Types of Web Architecture

- Client-server Architecture
- Three-tier Architecture
- **Service-oriented architectures (SOA)**
  - ✓ SOAP
  - ✓ REST

Web Application Architecture

- A web application architecture diagram presents a layout with all the software components (databases, applications, and middleware) and how they interact.
- It defines how the data is delivered through HTTP and ensures that the client-side and backend servers can understand.
- Moreover, it also ensures that valid data is present in all user requests.
- It creates and manages records while providing permission-based access and authentication.
- Choosing the right design defines your company's growth, reliability and interoperability

Types of Web Application Architecture

- The architecture of web applications can be classified into different categories based on the software development and deployment patterns.

Monolithic Architecture

- A monolithic architecture is a traditional software development model —also known as web development architecture— wherein the entire software is developed as a single piece of code going through the traditional waterfall model.
- It means all the components are interdependent and interconnected;
- every component is required to run the application.
- To change or update a specific feature, you need to change the entire code to be rewritten and compiled.

Microservices Architecture

- Microservices architecture solves several challenges that are encountered in a monolithic environment. In a microservice architecture, the code is developed as loosely coupled, independent services that communicate via RESTful APIs.
- Each microservice contains its own database and operates a specific business logic, so you can easily develop and deploy independent services.

Containers

- Container technology is the best option when it comes to deploying microservices.
- A container encapsulates a lightweight runtime environment for an application that can run on a physical or a virtual machine.
- As such, applications run consistently from the developer's device to the production environment.
- By abstracting execution at the OS level, containerization allows you to run multiple containers inside a single OS instance.
- While it reduces overhead and processing power, it increases the efficiency as well.

Serverless Architecture

- Serverless architecture is a model for developing software applications. In this structure, an infrastructure service provider manages the underlying infrastructure's provisioning.
- It means you only pay for the infrastructure when it's in use and not for the idle CPU time or unused space.
- Serverless computing lowers costs as resources are only used when the application is in execution.
- The cloud provider handles the scaling tasks. Moreover, the backend code gets simplified. It reduces development efforts, costs and brings faster time to market.

Reference

Amazon Web Service (https://aws.amazon.com/)

Thank You