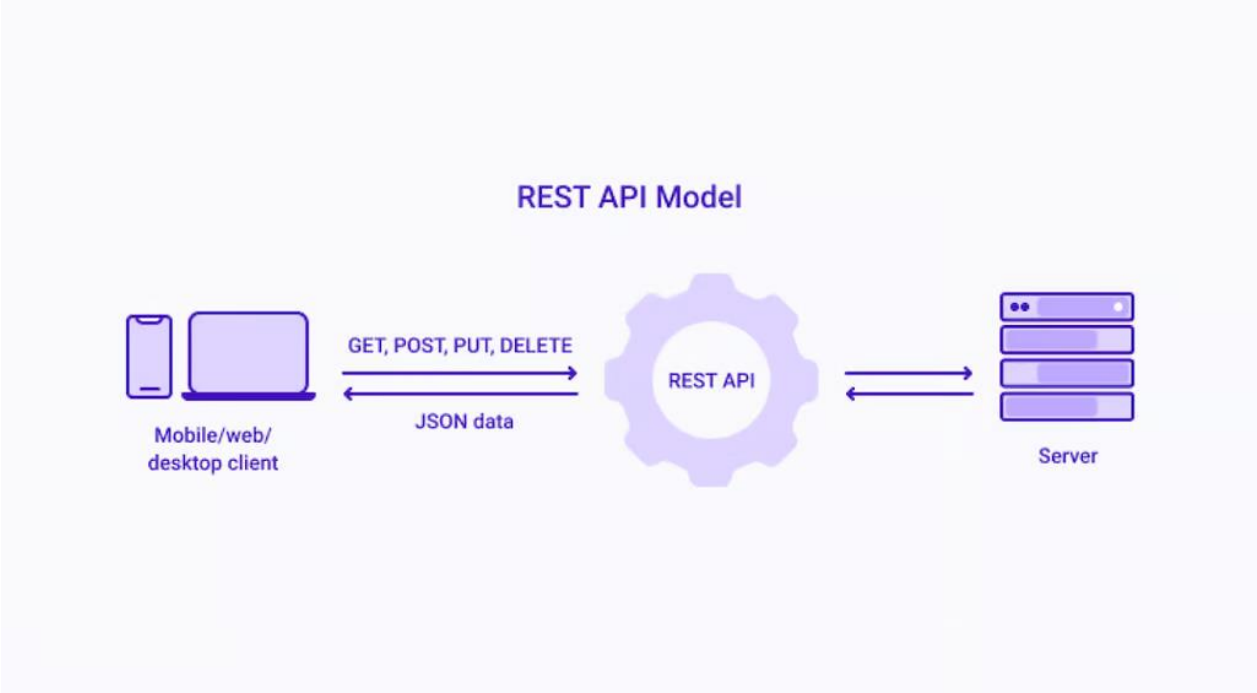
What is REST and Its Real Life Applications

An architectural technique for creating networked apps is called REST. It offers a set of guidelines and rules that, when followed to, produce web services that are loosely coupled, scalable, and stateless. When creating APIs (Application Programming Interfaces) for web applications, REST is frequently utilized.

Designing APIs that facilitate communication between clients and servers in web applications is a typical application of RESTful concepts. Because HTTP (Hypertext Transfer Protocol) is so widely used and supports all necessary actions (GET, POST, PUT, DELETE), it is frequently chosen as the communication protocol for RESTful services. Servers reply to requests from clients to particular URIs by presenting representations of the requested resources in the predetermined format.



Key principles and concepts of REST

1. **Client-Server Architecture:** REST uses a client-server model, in which the client and server are independent entities that communicate across a network. This separation provides for the independence and scalability of both the client and server components.
2. **Statelessness:** Since RESTful services are stateless, every request a client sends to a server needs to include all the details required for the server to comprehend and respond to it. Between requests, the server does not keep track of the client's state. This statelessness lowers the possibility of errors, improves scalability, and simplifies server implementation.
3. **Cacheability:** HTTP caching techniques are used by RESTful services to lower server load and increase performance. Clients can reuse previously fetched data when appropriate by designating whether or not a server response is cacheable.

4. **Uniform Interface:** One of the main constraint of REST is the uniform interface, which offers a consistent means of communication between clients and servers. These four principles define this interface:
    a. Resource Identification: URIs, are unique codes that uniquely identify resources, including data entities and services. RESTful services identify resources using URIs. Each resource is assigned a unique URI, enabling clients to interact with specific entities. For example, in a blogging platform, a URI like /posts/123 might represent a blog post with the ID 123.
    b. Resource Representation: JSON or XML are the most common formats used to represent resources. Instead of interacting directly with the resources, clients engage with representations of the resources.
    c. Stateless Communication: Every request a client sends to a server needs to include all the necessary data in order for the server to respond to it. Between requests, the server does not keep track of the client's state.
    d. Hypermedia as the Engine of Application State (HATEOAS): Application servers' dynamically provided hypermedia is the only way that clients interact with the application. HATEOAS improves the API's self-descriptiveness and discoverability.

Real life examples:

1. **GitHub API:** GitHub's API is one of the example of RESTful implementation. RESTful endpoints enable developers to manage issues, repositories, and user data. The flexibility of RESTful web services in a version control system is demonstrated by the actions that users can take by utilizing normal HTTP methods, such as fetching a list of issues, updating user profiles, or creating a new repository.
2. **Twitter API:** The Twitter API is a well-known illustration of RESTful architecture in practical applications. RESTful endpoints allow developers to work with Twitter's massive dataset, which includes user profiles, trends, and tweets. Developers can create, retrieve, or update tweets by using the right methods to make HTTP requests to specific URIs. This shows the simplicity and power of RESTful design. Twitter's decision to use REST architecture is rooted in several advantages that align with the characteristics of RESTful services so there are many advantages of REST for Twitter API but there are also some disadvantages too.

    Advantages:

    1. Simplicity and Ease of Use: RESTful APIs are known for their simplicity and ease of use. The use of standard HTTP methods (GET, POST, PUT, DELETE) makes it straightforward for developers to understand and interact with the Twitter API.
    2. Scalability: The statelessness of RESTful services facilitates horizontal scaling, which is crucial for a platform like Twitter with millions of active users generating a continuous stream of data.
    3. Caching: RESTful APIs often leverage HTTP caching mechanisms, such as ETags and Last-Modified headers. Caching helps reduce the load on the server by allowing clients to reuse previously fetched data, improving performance and responsiveness.

    Disadvantages:

1. Security Considerations: While RESTful APIs can be secured using standard mechanisms, the security of the API depends on the proper implementation of these measures. Additionally, as RESTful APIs often rely on URL parameters, securing sensitive information in the URL can be challenging.

Why not use pub-sub for Twitter API ?

1. Real-Time Updates: Twitter is a platform where real-time updates are critical. Users expect to receive immediate notifications about new tweets, mentions, and other activities. Pub-Sub architectures can introduce latency due to the inherent asynchronous nature of message delivery.
2. Scalability Challenges: While Pub-Sub systems are inherently scalable, achieving efficient and elastic scaling can be challenging in some scenarios. Twitter experiences a massive volume of concurrent users and tweets, and ensuring that Pub-Sub infrastructure can scale horizontally to handle these volumes might introduce complexities.
3. Guaranteed Message Delivery: In a Pub-Sub model, there might be scenarios where messages are not guaranteed to be delivered to all subscribers.
4. Resource State Representation: Twitter's RESTful API allows clients to retrieve representations of specific resources, such as tweets or user profiles. This resource-centric approach aligns well with the nature of Twitter data. Pub-Sub, on the other hand, focuses more on event-driven messaging without necessarily exposing the underlying resource state.