API Design Patterns

**Introduction**

API design patterns are critical components of software development, allowing developers to produce code that is dependable, scalable, and reusable. With the growing popularity of microservices and cloud computing, developing a successful API is more important than ever. APIs are contracts that specify how applications, services, and components interact. API Design Patterns describe the best practices for creating reliable, user-friendly APIs. These design patterns may be used to tackle common API difficulties and easily customized to meet your requirements.

**Types of API Design Patterns**

* RESTful API
* RPC API
* GraphQL API
* SOAP API
* Hypermedia API
* Event-driven API
* Message Queue API
* gRPC API

**RESTful API**

The RESTful API is a prominent and commonly used web API design paradigm. REST (Representational State Transfer) is a set of principles for structuring and manipulating server resources. A RESTful API operates on resources using HTTP methods (GET, POST, PUT, DELETE, etc.) and represents the data in standard forms (JSON, XML, etc.). A RESTful API is straightforward, consistent, and scalable, and it allows for caching security and performance optimization. However, there are certain limitations to using a RESTful API, such as the necessity for several queries to retrieve related data, the absence of statefulness, and the difficulties of versioning.

Advantages

* REST API allows you to structure complex applications and make resources easier to utilize.
* High load may be controlled with the use of an HTTP proxy server and cache.
* The REST API is simple to navigate and discover.
* It makes it easy for new clients to work on other apps, whether they are particularly developed for that purpose or not.
* To obtain data and make requests, use standard HTTP procedure callouts.
* REST API is code-based, and it may be used to synchronize data with a website without problems.
* Users can access the same standard objects and data models as SOAP-based web services.

Disadvantages

* Lack of state: Most web apps rely on stateful techniques. Suppose you buy a website with a shopping cart feature. Before making a purchase, you must first know how many products are in your shopping basket. This responsibility for preserving the state falls on the client, making the client application heavy and difficult to maintain.
* Last of security: like SOAP, imposes security requirements. That is why REST is acceptable for public URLs but not for transmitting sensitive data between client and server.

**RPC API**

RPC API stands for Remote Procedure Call Application Programming Interface. It's a protocol that enables one program to request a service from another program on a network without needing to comprehend the network's specifics. RPC APIs encapsulate network communication, allowing programs to communicate across several systems as if they were local procedures or function calls. This abstraction facilitates the creation of distributed systems.

**Advantages**

* RPC is ideal for applications that require complicated data structures and business logic.
* RPC can be quicker than REST in some circumstances because it avoids the overhead of HTTP.
* RPC may be more secure than REST since it may employ SSL or other encryption techniques to safeguard data in transit.
* RPC can handle a wide range of data types, including binary data, which might be more efficient in some situations.

**Disadvantages**

* RPC can be more difficult to build than REST since it requires a unique protocol and API architecture.
* RPC can be strongly tied to a programming language or platform, making it less adaptable than REST.
* RPC is less scalable than REST since it is stateful and may need extra infrastructure to manage high quantities of traffic.

**GraphQL API**

GraphQL is a query language and runtime system that enables clients to describe exactly what data they want from the server and how they want it formatted. A GraphQL API handles all requests through a single endpoint and defines the data's types and relationships using a schema. A GraphQL API is adaptable, efficient, and powerful, minimizing the amount of data sent and requests required. However, there are certain drawbacks to using a GraphQL API, such as the schema's complexity, the absence of caching, and the possibility of over or under-fetching data.

Advantages

* **Good fit for complex systems and micro services**
* **Validation and type checking out-of-the-box**
* **No over- and under-fetching problems**
* **Auto-generating API documentation**
* **API evolution without versioning**
* **Detailed error messages**

Disadvantages

* **Performance issues with complex queries.**
* **Overkill for small applications.**
* **Web caching complexity.**

**SOAP API**

The Simple Object Access Protocol (SOAP) is a messaging protocol for sharing data between systems and applications. In terms of application programming interfaces (APIs), a SOAP API is more organized and defined. SOAP's architectural patterns are less tightly connected with HTTP than REST's; it may also use other protocols. While REST is more of a style, SOAP provides considerably more information about the structure of the request and response, as well as message content and how it will be encoded. Simply said, while building APIs, SOAP focuses on the message, whereas REST focuses on the resources. SOAP uses XML as part of a standard communication protocol that enables the exchange of structured data in dispersed situations. SOAP allows apps that operate on multiple operating systems and programming languages to interact with one another.

Advantages

* SOAP is language, transport, and platform-independent.
* SOAP is very secure, making it ideal for systems handling sensitive data, such as financial services and online banking apps.
* SOAP performs effectively in dispersed corporate systems rather than relying on direct point-to-point communication.
* SOAP contains built-in error-handling tools that simplify determining what happens when a request fails.

Disadvantages

* SOAP does not support caching API calls.
* SOAP is complicated and can have performance implications.
* SOAP is much less adaptable.
* SOAP is usually slower.

**Hypermedia API**

Advantages

* Reduced Coding Errors – Client Side
* Detailed Application Upgradation Without Client Breakage
* Includes links between resources, allowing for dynamic discovery and navigation
* Can improve the flexibility and adaptability of an API

Disadvantages

* May require additional tooling and libraries
* Can be more difficult to implement and maintain
* Resource Hungry Architecture
* Reduced Flexibility
* Information Overload

**Event-driven API**

Advantages

* Clients receive alerts when specific events occur.
* Clients may no longer need to query for updates as frequently.
* Can be beneficial in real-time applications.

Disadvantages

* It can be more complex to adopt and manage.

**Message Queue API**

Advantages

* Enables apps to send and receive messages asynchronously.
* Can offer a dependable and scalable solution to process communications.
* Can be beneficial in distributed systems.

Disadvantages

* There may be a need for extra infrastructure and tools.

The choice of API design pattern will be determined by the project's unique requirements and system architecture. RESTful APIs are frequently an excellent choice for simple CRUD tasks, but GraphQL may be better suited for complicated queries. RPC API and SOAP API are more efficient for some sorts of tasks, but they are more difficult to develop and manage. Hypermedia API, event-driven API, and message queue API can be beneficial for specific systems and applications, although they may necessitate additional tooling and infrastructure.