API

API, REST, SOAP

Posted on February 18, 2023
Last updated on February 18, 2023

1. API

API stands for `Application Programming Interface`. Its a software interface that allows `two applications to interact` with each other without any user intervention.

It offers products or services to communicate with other products and services `without having to know how they're implemented.`

Protocals that can be used in APIs are TCP, SMTP, HTTP

1.2 Why would we need an API?

- API helps `two different software's to communicate` and exchange data with each other.
- It helps you to `embed content` from any site or application more efficiently.
- APIs can access app components. The delivery of services and information is more flexible.
- Content generated can be published automatically.
- It allows the user or a company to customize the content and services which they use the most.
- Software needs to change over time, and APIs help to anticipate changes.

1.3 What are the different types of APIs?

APIs are classified both according to their `architecture and scope` of use. We have already explored the main types of API architectures so let's take a look at the scope of use.

- **Private APIs**
  - These are `internal to an enterprise` and only used for connecting systems and data within the business.
- **Public APIs**
  - These are `open to the public` and may be used by anyone.
  - There may or not be some authorization and cost associated with these types of APIs.
- **Partner APIs**
  - These are only accessible by `authorized external developers` to aid business-to-business partnerships.
- **Composite APIs**
  - These `combine two or more different APIs` to address complex system requirements or behaviors.

1.4 How do APIs work?

API architecture is usually explained in terms of `client and server`. The application `sending the request` is called the client, and the application `sending the response` is called the server.

There are `four different ways` that APIs can work depending on when and why they were created.

- **SOAP APIs**
  - These APIs use Simple Object Access Protocol. Client and server exchange messages using XML.
  - This is a less flexible API that was more popular in the past.
- **RPC APIs**
  - These APIs are called Remote Procedure Calls. The client completes a function (or procedure) on the server, and the server sends the output back to the client.
- **Websocket APIs**

    - Websocket API is another modern web API development that uses JSON objects to pass data.
    - A WebSocket API supports `two-way communication` between client apps and the server.
    - The server can send callback messages to connected clients, making it more efficient than REST API.
- **REST APIs**
    - These are the most popular and flexible APIs found on the web today.
    - The client sends requests to the server as data.
    - The server uses this client input to start internal functions and returns output data back to the client.

1.5 SOAP API vs REST API

- SOAP API
    - Simple Object Access Protocol
    - Protocol Used: TCP or SMTP
    - Data Transfer: XML (WSDL)
    - More secure than REST API
    - Preffered Choise: Offline application
- REST API
    - Representational State Transfer
    - Protocol Used: HTTP
    - Data Transfer: JSON
    - Less secure than SOAP API
    - Preffered Choise: Web based application
    - Fxn: PUT, GET
    - Simple, Scalable and flexible

1.6 WEB API

Web API is an API as the name suggests, it can be accessed over the web using the `HTTP protocol`. It a subset of API.

A Web API or Web Service API is an application processing interface `between a web server and web browser`. All web services are APIs but not all APIs are web services. REST API is a special type of Web API that uses the standard architectural style.

The different terms around APIs, like Java API or service APIs, exist because historically, APIs were created before the world wide web. Modern web APIs are REST APIs and the terms can be used interchangeably.

1.7 REST API

`REpresentational State Transfer` API (Application Programming Interface)

Its an `architectural style` for providing standards between computer systems on the web, making it easier for systems to communicate with each other.

Provides a `set of principles` for building web services that used HTTP as a communication protocol.

1.7.1 Architectural Constraints

REST defines `6 architectural constraints` which make any web service – a true RESTful API.

- Client–server
- Stateless
- Cacheable
- Layered system
- Uniform interface
- Code on demand (optional)

Client–server

The `client and server` application are separated, allowing then `evolve independently` without any dependency on each other.

A client should know only resource URIs and that's all.

Stateless

Each request from a client must contain all the information required by the server to carry out the request. The server does not maintain any client context between the request.

Server will not store anything about latest HTTP request client made. It will treat each and every request as new. No session, no history.

Cacheable

Response from the server can be cached to improve performance. Caching can be implemented on the server or client side.

In today's world, caching of data and responses is of utmost important wherever they are applicable/possible. Caching brings `performance improvement` for client side, and better `scope for scalability` for a server because the load has reduced.

Uniform interface

The method of communication between a client and a server must be uniform. Like GET, POST, DELETE.

A resource in the system should have only one logical URI and that should provide a way to fetch related or additional data.

Any single resource should not be too large and contain each and everything in its representation. Whenever relevant, a resource should contain links (HATEOAS) pointing to relative URIs to fetch related information.

Layered system

REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C.

For example. A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.

Communication between a client and a server should be standardized in such a way that allows intermediaries to respond to requests instead of the end server, without the client having to do anything different.

Code on demand (optional)

Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Most of the time you will be sending the static representations of resources in form of XML or JSON. But when you need to, you are free to return executable code to support a part of your application e.g. clients may call your API to get a UI widget rendering code. It is permitted.

1.7.2 Methods of REST API

- REST API offers CRUD operations -

```
* C: Create     --> POST
* R: Read       --> GET
* U: Update     --> PUT
* D: Delete     --> DELETE
```

GET Request

GET requests is used to `retrieve resource` representation/information only – and not modify it in any way.As GET requests do not change the resource's state, these are said to be safe methods.

Additionally, GET APIs should be `idempotent` . Making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

POST Request

POST request is used to `create new` subordinate resources, e.g., a file is subordinate to a directory containing it or a row is subordinate to a database table.

POST is `neither safe nor idempotent` , and invoking two identical POST requests will result in two different resources containing the same information (except resource ids).

PUT Request

PUT request is used primarily to `update an existing resource` (if the resource does not exist, then API may decide to create a new resource or not).

Delete Request

DELETE request is used to delete the resources (identified by the Request-URI).

DELETE operations are idempotent. If you DELETE a resource, it's removed from the collection of resources.

Some may argue that it makes the DELETE method non-idempotent. It's a matter of discussion and personal opinion.

.

.

.

.

.

.

.

.

.

.

.

.