**Application Programming Interfaces**

Diiferent **applications** communicate with each other through APIs

These APIs are written **programmatically** in which we mention what we want to send/receive from other application, or we need other app to perform some computation for this app.

This APIs are an abstract layer like an **interface** where, one app requesting/ sending data does not know how the responding app does the work.

Ex: Instagram Post button is un-aware of how a photo is posted by Instagram in backend.

APIs provide a freedom of implementation.

Ex: An Instagram API “getFollowers( )” on frontend wants to get current number of followers.

It is independent of how getFollowers() is handled by Instagram server applications.

Hence this API remains the same, and way getFollowers( ) is handled in backend ca be improved (by bug fixing) or refactored anytime.

APIs are platform Agnostic. Application fetching data can be written in JS, application sending data can be written in C#. APIs are not dependent on platform.

**Examples of APIs**

1. **Private APIs**

These APIs hidden APIs that only an application running in device has access to.

1. **Public APIs**

Certain APIs are publically available like Google Maps API, Google Fonts, Weather APIs, FB Developer API, etc.

1. **Web APIs**

These are superset of public and private APIs. These APIs are for applications running on cloud to interact with each other. Like: getFollowers( ), postPhoto( ) on instagram

1. **SDK/ Library APIs**

Example: A threading library which has certain APIs like lock(), unlock( ), fork( ), join( ), sleep( ).

These are APIs which we will use in program when we want to use threading in our application. Advantage: abstraction, no need to worry how they work.

**API Factors**

**There are certain factors to be aware about while designing and developing an API**

1. **API Contacts**
   1. We communicate with the developer of the API, who gives us certain contacts stating the endpoints which you must call in application. In what structure to send data.
   2. All these contracts are communicated between the two parties which will use each other’s APIs, then these are documented.
2. **Documentation**

In case of public APIs, documentation is written by developers and publicly available over internet.

1. **Data Formats**

It’s important to decide in what data format will exchange over APIs

1. **Security**

By providing API access to another developer, you are exposing a way to enter into your system. Hence endpoint security is of utmost importance.

If someone gets access to your API, they can send wrong information, perform SQL Injection/ DDOS Attack to bring your system and lead to business loss.

Need to consider factors like rate limiting, throttling while developing APIs.

Rate Limiting and Throttling policies are designed to limit API access, but have different intentions: Rate limiting protects an API by applying a hard limit on its access. Throttling shapes API access by smoothing spikes in traffic.

The **Rate Limiting** policy limits the number of requests an API accepts within a window of time. The API rejects requests that exceed the limit. You can configure multiple limits with window sizes ranging from milliseconds to years.

The **Throttling policy** queues requests that exceed limits for possible processing in a subsequent window. The API eventually rejects the request if processing cannot occur after a certain number of attempts. You can configure a delay between retries, as well as limit the number of retries.

**Industry Standards for writing an API**

* RPC
* SOAP
* REST
* GraphQL