APIs, SDKs and Libraries

JUNE 27, 2022

APIs, SDKs and Libraries

- How are they different?

Libraries

- Library – reusable pre-written code "chunks" callable from your own code to do things quickly
- Libraries run in the same context of your program, i.e. runs on on the same system your program is running and is considered part of your program by the operating system
- Libraries are typically **include**'d or **import**'d.
- You can write your own libraries.
- Libraries have to be distributed with your program when deployed.

Application Programming Interface (API)

- API's work with running programs. The code executes in a different context than your program code. API's allow your program to communicate and use other RUNNING programs.
- An API is a contract stating all the capabilities a program can perform for another program. Each capability will typically options or parameters.
- Sometimes API's will need certain capabilities called in a particular sequence.
- API's are typically language agnostic

API's and Libraries

- Often you use libraries in your code to communicate with an API.
- Often an API library will have library methods that map directly onto an API method.
- In certain cases, a library can even be an API.
- While API's are language agnostic, libraries are not. Sometimes you may have trouble finding an API language library for your language of choice.

Software Development Kit (SDK)

- Complex software will often have many API's organized very basic levels of functionality.
- SDK's assemble collections of the library API's for a particular software or even set of software (often referred to as a software stack.)
- SDK's typically support multiple languages.
- SDK's will typically have extensive documentation and samples as well as test cases for development.

Frameworks

- Framework's are a collection of SDK's, API Libraries, templates, and other necessary code bits to write programs for specific use cases.
- Django and Flask are two examples of web-application development frameworks.
- Frameworks often will make many assumptions and define additional standard ways of using SDK's and API's. These are called *opinionated* frameworks.

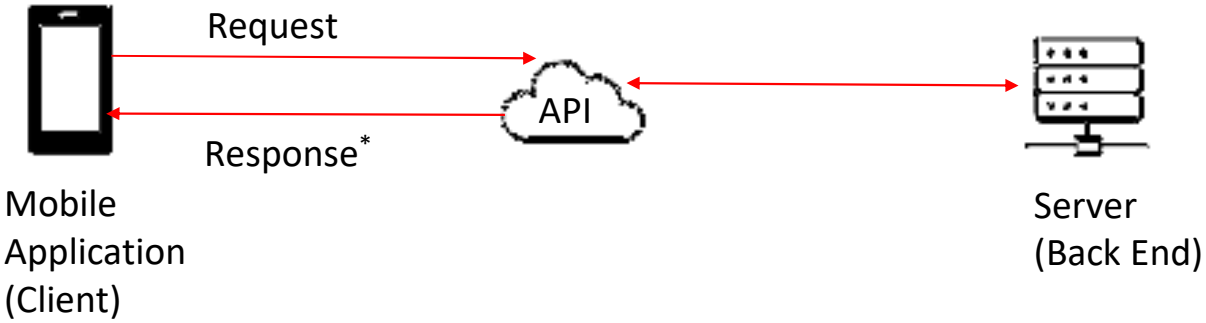
Related terms

- Client-server architecture: separation of interface from backend and data storage. Flexible, allows different components to evolve independent of each other
- Stateless: client context is not stored on the server between requests
- Cache or caching: Holding on to a response and re-using a prior response rather than requesting a new response.
- Layered architecture: A way to organize units of processing in such a way that work flows linearly between processing units. Each layer is responsible for only a small piece of the total work.

Have we used libraries before?

- Yes, just yesterday...*pandas, matplotlib*!

APIs



* Not all API's need to return a response. It depends on how the developer designed the API.

APIs

- When are they used
  - Open or Public APIs

  - Partner APIs - for interfacing between products

  - Internal or Private APIs

Using API's

- Most API's these days are designed to be run across a computer network.
- The API will focus on the content and structure of the request and responses but typically will have to say about how the request is delivered and the response received.
- Many API's rely on the HTTP network protocol to send API requests and receive API responses.

API Architectures and Protocols

- *JSON-RPC and XML-RPC*

- Aim: Encoding RPC (Remote Procedure Call) either in JSON or XML

- Designed to call methods, unlike REST protocols involve the transfer of documents (resource representations). Actions vs. Documents

- The URI identifies the server, but contains no information in its parameters, whereas in REST the URI contains details such as query parameters.

API Architectures and Protocols

- *SOAP (simple object access protocol)*

Aim: Independent, extensible, neutral

Specifies:

- Processing model: how to process a SOAP message
- Extensibility model: SOAP features and modules
- Protocol binding rules: how to use SOAP with an underlying protocol, such as HTTP
- Message construct: how to structure a SOAP message.
- SOAP specifies the structure of the message as XML

API Architectures and Protocols

- *Rest (representational state transfer)*

- Uniform Interface for consistent message formatting
- Most REST APIs use the common HTTP, or **Hyper-Text Transfer Protocol language**.
- HTTP wasn't created specifically for REST. REST adopted this communication protocol as the standard for applications that use it.
- To use HTTP with a REST API, the client sends a request in a specific format that might look familiar to you. For example, a request to the YouTube API for video data looks like this:

API Architectures and Protocols

- *Rest (representational state transfer)*

- To use HTTP with a REST API, the client sends a request in a specific format that might look familiar to you e.g.

GET https://www.googleapis.com/youtube/v3/channels?part=contentDetails

API Architectures and Protocols

- *Rest (representational state transfer)*

GET is the HTTP method. There four basic HTTP requests a client can make are:

GET: To retrieve a resource.
POST: To create a new resource.
PUT: To edit or update an existing resource.
DELETE: To delete a resource.

API Architectures and Protocols

- *Rest (representational state transfer)*

https://… is the URL. Contains the uniform resource identifier(URI) specifying the target resource.
- URL is also called an **endpoint** because it is the location where the API actually interacts with the client.
- After receiving and validating the request, the host returns information about the target resource. Usually, the information is back sent in a format called JSON, which stands for **JavaScript Object Notation**. JSON lays out all the contents of a resource in a lightweight format that humans can easily read.

API Architectures and Protocols

- *Rest (representational state transfer)*

- Response

- Head
- Body

- Rate Limit - number of requests you can make

- Authentication

- oAuth

Q&A