APIs, SDKs and Libraries

JUNE 29, 2021

APIs, SDKs and Libraries

- How are they different?

APIs, SDKs and Libraries

- Library – reusable pre-written code "chunks" callable from your own code to do things quickly

- API – Application Programming Interface. "Face" of the library; logical representation of what is in the library e.g.. Java API, Google Maps API for data exchange

- SDK – Software development kit. Integration of APIs, libraries, including IDEs, documentation etc. into a unit for SDE on a specific platform e.g., JDK

APIs, SDKs and Libraries

- Framework – code you can insert your own code into. Your code gets called by the framework e.g. app development (Compare to library which is the inverse – building blocks vs. Beams, posts to build onto)

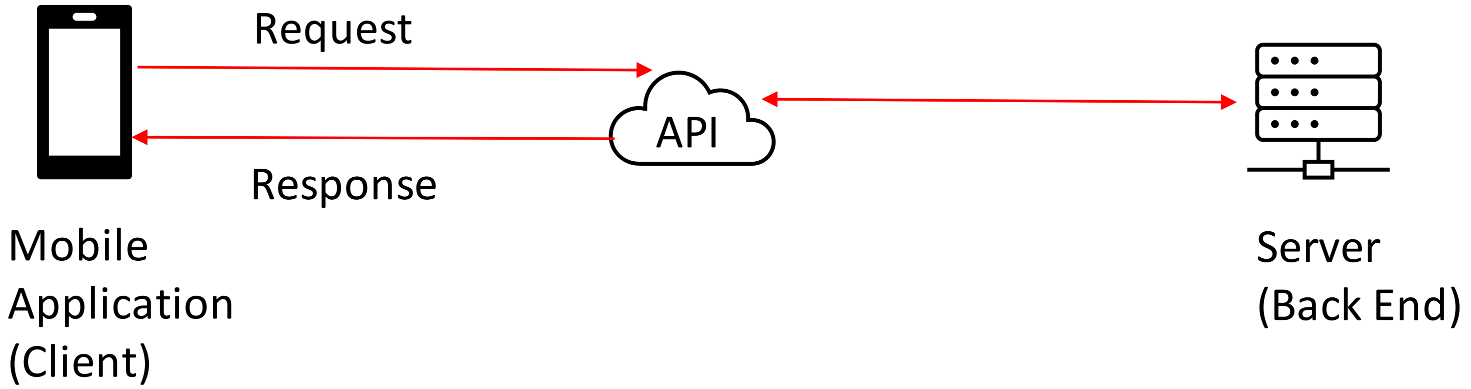
Have we used libraries before?

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

APIs

- API stands for *Application Programming Interface*
- Sets of rules that allow interaction between applications
- Similar to libraries, APIs let you add functionality without programming it from scratch mainly for data access

APIs


Request
API
Response
Mobile Application (Client)
Server (Back End)

APIs

- When are they used
  - Open or Public APIs
  - Partner APIs - for interfacing between products
  - Internal or Private APIs

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.

API Architectures and Protocols

- *Rest (representational state transfer)*

- Client-server architecture: separation of interface from backend and data storage. Flexible, allows different components to evolve independent of each other

- Statelessness: client context is not stored on the server between requests

API Architectures and Protocols

- *Rest (representational state transfer)*

- Cacheability: REST API response must explicitly state whether it can be cached or not (clients can cache)

- Layered system: API works irrespective of whether it is in direct communication with server, or through an intermediate load balancer.

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

API Architectures and Protocols

- *JSON-RPC and XML-RPC*

- Aim: Encoding RPC (remore procedure calls) 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.

Q&A