Data Collection and Modeling

Courser 5 - API

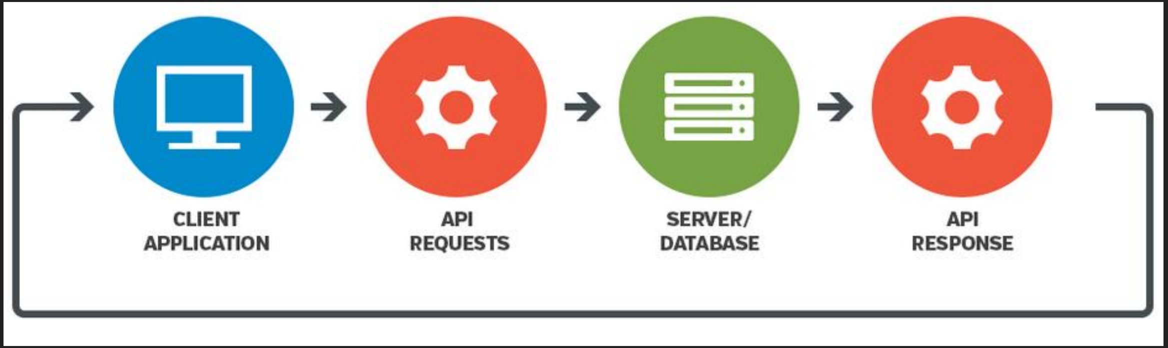
Definition

“An application programming interface, or API, enables companies to open up their applications’ data and functionality to external third-party developers and business partners, or to departments within their companies. This allows services and products to communicate with each other and leverage each other’s data and functionality through a documented interface. Programmers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services.” (IBM)

“API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software.” (RedHat)

How API Works

APIs enable apps or services to communicate with other apps or services without knowing how are they implemented.
An API is a set of rules that state how apps or services communicate with one another (API specification).
APIs sit between an application and a server, acting as an intermediary layer that processes data transfer between systems.



source: Calameo

How API Works

- An app makes an API call (request) to retrieve information.
- Upon receiving a valid request, the API makes a call to the external application or server.
- The server sends a response to the API with the requested information.
- The API transfers the data to the initial requesting application.

APIs enable the abstraction of functionality between interacting systems. An API endpoint decouples the consuming application from the infrastructure that provides a service. As long as the specification for what the service provider is delivering to the endpoint remains unchanged, the alterations to the infrastructure behind the endpoint should not be noticed by the applications that rely on that API.

History of APIs

- Commercial APIs: early 2000s, startups made available products and services but also enabled partners and resellers to extend reach of their platforms. Salesforce, Amazon, eBay
- Social media APIs: mid 2000s; Flickr, Facebook, Twitter
- Cloud APIs: 2006 AWS S3, EC2; enables the use of web APIs to deploy infrastructure
- Mobile apps APIs: after iPhone and Android in 2007; Twilio, Instagram
- API for connected devices: 2010, connect cameras, speakers, sensors to cloud; Fitbit, Nest, Alexa

Types of API

- By management strategies
  - Open API: open source; also known as public APIs
  - Internal API (aka Private API): not accessible to external users; built for company’s internal use to improve integration of internal systems
  - Partner API: exposed only to business partners; usually require some form of authentication
  - Composite API: combine several APIs and allow the access of several endpoints in a call
- By purpose
  - Process API: combine and orchestrate APIs for a specific business purpose
  - System API: provide access to core systems
  - Experience API: exposes services to intended audience

Architectural Styles

- RPC (XML/JSON): Remote Procedure Call; XML or JSON, simple and lightweight
- SOAP: Simple Object Access Protocol; based on XML
- REST: Representational State Transfer; web API architectural principles
- GraphQL: query language for the API; multiple sources in a API call

RPC

XML-RPC (Remote Procedure Call) is a protocol for cross-platform communication. It makes procedure calls by using HTTP as transport and XML as the encoder.

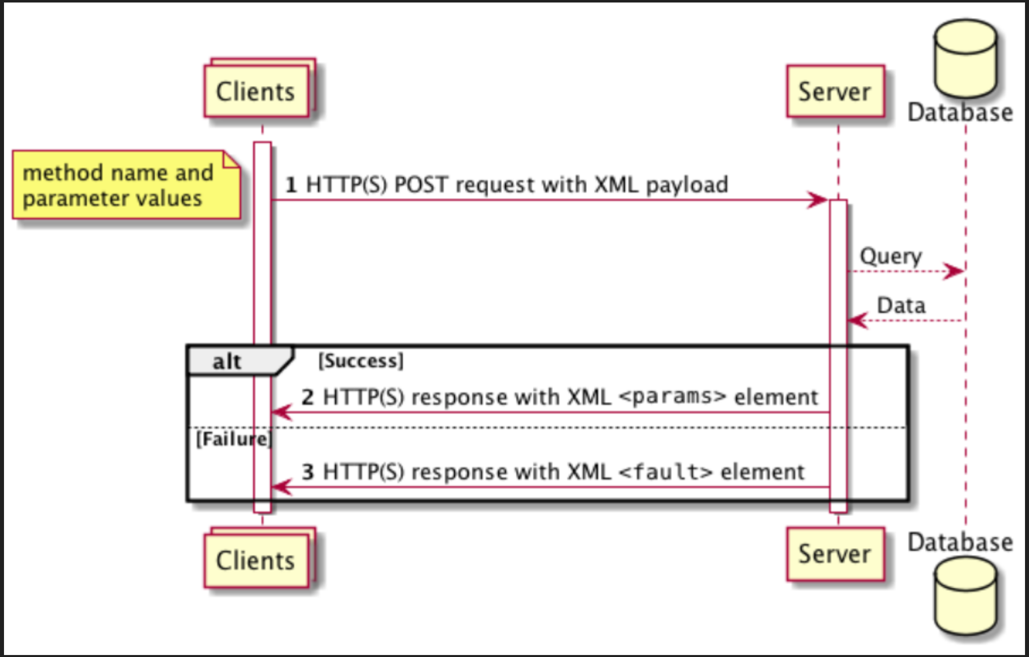
The client makes a call by sending an HTTP request to the server and receives the HTTP response in return.

XML-RPC invokes functions via HTTP request, the functions perform some actions and send hard-coded responses in return.

It does not provide asynchronous model, streaming or security.

RPC

- A client sends an XML document to the server using an HTTP request
- The server either returns
  - HTTP status and XML containing return value
  - HTTP status and XML containing an error


Clients
Server
Database
method name and parameter values
1 HTTP(S) POST request with XML payload
Query
Data
alt
[Success]
2 HTTP(S) response with XML <params> element
[Failure]
3 HTTP(S) response with XML <fault> element
Clients
Server
Database

source: Papercut

JSON-RPC

- stateless, lightweight remote procedure call (RPC) protocol.
- defines data structures and the rules for processing them.
- can be used within the same process over sockets, HTTP, or many other message passing environments.
- uses JSON (RFC 4627) as data format.

JSON-RPC

Request

Method: the string that denotes a method. There is a set of reserved names with prefix ‘rpc’, meant for internal RPC calls

Params: can be an object or an array, the value to be passed to method.

Id: A number that identifies the request. In the absence of a response towards a request, the ID will be removed automatically.

```
{"id": 1, "method": "sum", "params": [1,2]}
```

Response

Result: the data returned by invoked method.

Error: the second part, if a problem occurs. It has a code and a message.

Id: refers to the request for which response relates.

```
{"id": 1, "result": 3}
```

SOAP

The web service architecture is based on components that play different roles in the interaction between them

Service provider: the collection of software that provides the service

- application
- middleware
- platform

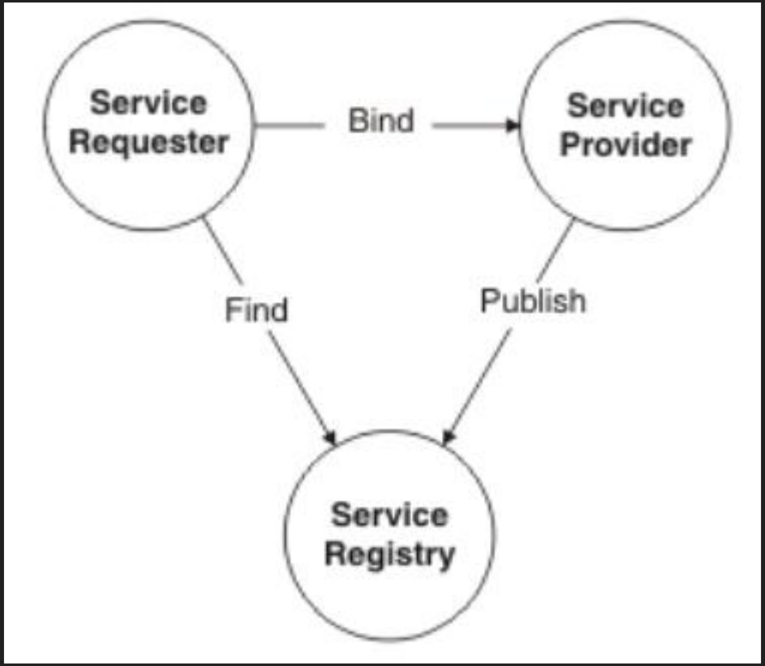
Service requester: the collection of software that requests a service

- application
- middleware
- platform

Service registry: a central location where providers publish service descriptions and requesters can find them

- optional
- allows versioning

SOAP



source: IBM

- Publish: the provider publishes service description in the registry so it can be found
- Find: the requester finds service by description in registry
- Bind: the requester binds with the provider and interacts with the implementation

SOAP

API is a form of agreement between web services on how they are exchanging data.

Service description: a document that service provider uses to communicate the specification to requesters. It is expressed in XML but referred as WSDL (Web Service Description Language). It defines and describes endpoints, data types and actions available.

Strongly follows standards related to message structure, encoding rules, to provide platform independent and language independent communication.

It offers a good level of security.

SOAP Message

- Envelope: the root element in every message; can contain an optional <Header> and a mandatory <Body>
- Header: is used to transmit information related to application to be processed by nodes along the path, extra requirements (authentication)
- Body: contains the request or the response
- Fault: (optional) information about errors

It does not enforce the content of <Header> or <Body>

SOAP Specification

Common web service specifications include:

- Web services security (WS-security): Standardizes how messages are secured and transferred through unique identifiers called tokens.
- WS-ReliableMessaging: Standardizes error handling between messages transferred across unreliable IT infrastructure.
- Web services addressing (WS-addressing): Packages routing information as metadata within SOAP headers.
- Web services description language (WSDL): Describes what a web service does, and where that service begins and ends.

SOAP Example - Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
               <HelloRequest xmlns="http://learnwebservices.com/services/hello">
               <Name>John Doe</Name>
               </HelloRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

https://apps.learnwebservices.com/services/hello

SOAP Example - Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
     <HelloResponse xmlns="http://learnwebservices.com/services/hello">
       <Message>Hello John Doe!</Message>
     </HelloResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP Example - Error

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

   <soap:Body>

                 <soap:Fault>

                 <faultcode>soap:Client</faultcode>

                 <faultstring>Unmarshalling Error: Unexpected '&lt;' character in element (missing
closing '>'?)

 at [row,col {unknown-source}]: [6,13]</faultstring>

                 </soap:Fault>

   </soap:Body>

</soap:Envelope>
```

SOAP Extensibility

SOAP has been augmented with standard protocols (WS-*) that enhance it by specifying how things are done:

- WS-Security
- WS-Messaging
- WS-Transactions
- WS-MetadataExchange
- WS-Atomic-Transactions (ACID Compliant)

REST

Representational State Transfer is a software architecture style that uses a stateless communications protocol, usually HTTP.

Most often REST uses JSON to structures data, but also XML, YAML, or any other format that is machine-readable can be used.

It uses the object-oriented programming paradigm of noun-verb and is data-driven

The components of a REST system are:

- client: makes a request for a resource
- server: owner of the requested resources

REST

6 criteria (constraints) for a RESTful API:

- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)

REST

Uniform Interface: only one way of interacting with a server to request a resource so that information is transferred in a standard form.

- Resource-Based: resources are identifiable and separate from the representations sent to the client
- Manipulation of Resources Through Representations: client can manipulate the resource by means of representation of resource as it contains enough information
- Self-descriptive Messages: messages include enough information to describe how the messages should be processed
- Hypermedia as the Engine of Application State (HATEOAS): after accessing a resource the client should be able to use hyperlinks to find all other currently available actions it can take

REST

Stateless: no client information is stored between requests, the necessary state to handle the request is contained within the request itself; the server doesn't store anything related to the session.

The client can include all information for the server as a part of query params, headers or URI.

Cacheable: the response should include information that states if it is cacheable or not and, when yes, the duration it can be cached at the client. For that period, the client will return the data from cache without sending the request again to the server.

A good caching policy can eliminate some client–server interactions, improving availability and performance.

REST

Client-Server: application has a client-server architecture. Client and server can evolve/change independently.

Layered system: the architecture is composed of multiple layers. A layer knows only about the immediate layer; there can be many intermediate servers (invisible to the client) between client and the end server. These servers may be responsible for load-balancing, security shared caches.

Code on demand: server can send executable code to the client, extending client functionality.

REST

Rules for creating REST API endpoints:

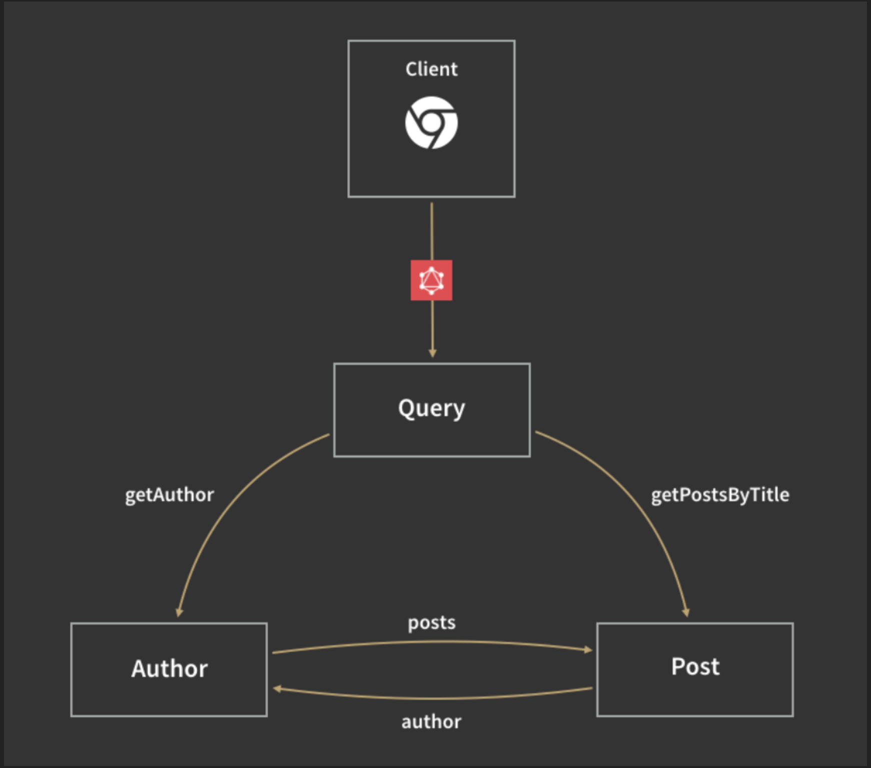
- REST is based on the noun (resource) instead of verb (action). A URI should always end with a noun: /api/users
- HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, GET, PATCH.
- Application should be organized into resources and use HTTP verbs to modify the resources. By looking at the endpoint and HTTP method used one should know what needs to be done.
- Use plurals in URL as naming standard to keep API URI consistent.
- Send a proper HTTP code to indicate a success or error status.

GraphQL

“GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.”

- Is a query language and server-side runtime for APIs; provides complete and understandable description of data
- A GraphQL query defines needed resources and returns predictable results
- Allows retrieval of several resources in one request by following references between them
- GraphQL APIs don't use endpoints but types and fields. All data is accessed using a single endpoint

GraphQL



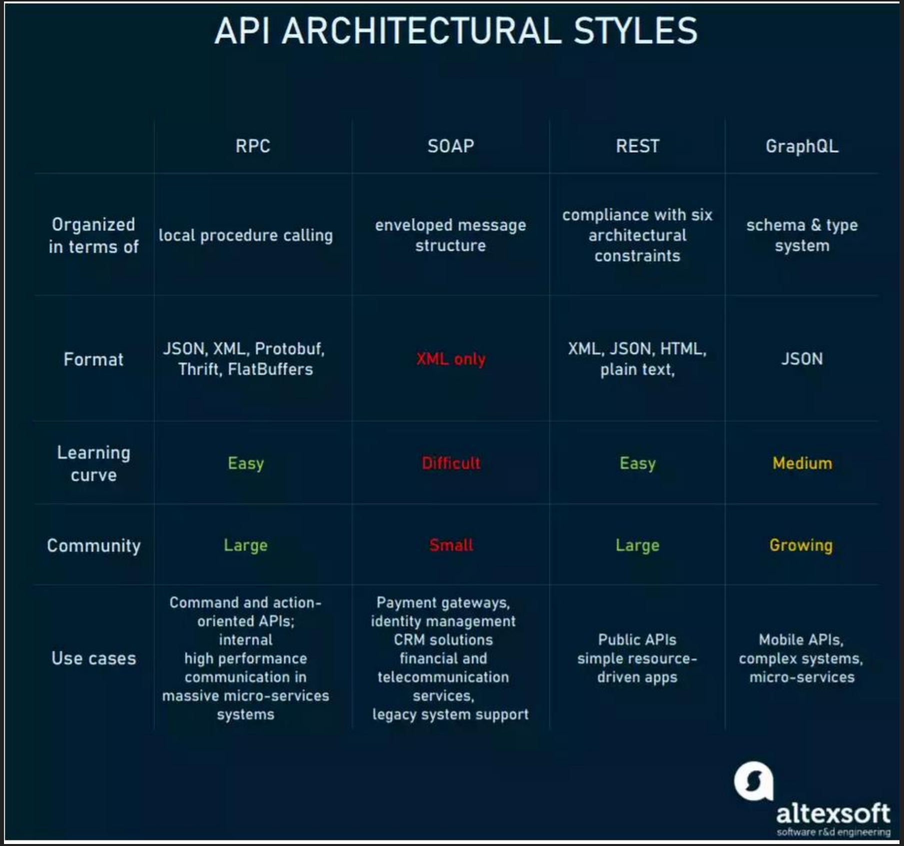
```
type Author {
  id: Int
  name: String
  posts: [Post]
}type Post {
  id: Int
  title: String
  text: String
  author: Author
}type Query {
  getAuthor(id: Int): Author
  getPostsByTitle(titleContains: String): [Post]
}schema {
  query: Query
}
```

source: [Apollographql]

GraphQL

```
{
  getAuthor(id: 5){
              name
              posts {
              title
              author {
              name # this will be the same as the name above
              }
              }
  }
}
```

RPC vs SOAP vs REST vs GraphQL



API ARCHITECTURAL STYLES

	RPC	SOAP	REST	GraphQL
Organized in terms of	local procedure calling	enveloped message structure	compliance with six architectural constraints	schema & type system
Format	JSON, XML, Protobuf, Thrift, FlatBuffers	XML only	XML, JSON, HTML, plain text,	JSON
Learning curve	Easy	Difficult	Easy	Medium
Community	Large	Small	Large	Growing
Use cases	Command and action-oriented APIs; internal high performance communication in massive micro-services systems	Payment gateways, identity management CRM solutions financial and telecommunication services, legacy system support	Public APIs simple resource-driven apps	Mobile APIs, complex systems, micro-services

altexsoft
software r&d engineering

Free Web Services

https://dog.ceo/dog-api/

https://swapi.co/

https://graphical.weather.gov/xml/

https://swagger.io/

https://www.predic8.de/public-soap-web-services.htm

Resources

JSON-RPC

SOAP

Services Articles

REST

REST Architecture

GraphQL

GraphQL (Apollo)