The Beginner’s Guide to APIs

Raise your hand if you’ve heard the term “API” before! Now keep your hand up if you know not only what “API” stands for, but also what it means? Did you lower your hand? No worries! Read on for a comprehensive overview of Application Programming Interfaces!

1. Web Services

Before we begin to understand what exactly the modern Web API is, we must first understand what a web service is. What is a web service? A web service is a cloud application that runs on a web server and is used by web clients. Let’s break down this definition to truly understand the meaning of a web service.

First off, a web service is a cloud application. This means that the application is being hosted and managed on a remote server, as opposed to being downloaded and ran locally on individual users’ machines.

Secondly, a web service runs on a web server. Consider the World Wide Web, which is really a distributed collection of nodes that share data between each other. Nodes on the web may act as either servers or clients. Servers are the nodes that listen for requests and utilize protocols (like HTTP, FTP, etc.) in order to serve a response in return.

And finally, a web service is used by web clients. On the web, clients are the nodes that make requests to servers. They then display, manipulate, and/or consume the data for end users.

In a nutshell, web services are made to provide functionality to web clients on behalf of a web server.

Keep In Mind!
A web server is not the same as a physical server humming along in some air conditioned basement! However, a web server may run on a physical server.

Types of Web Services

In the world of cloud computing, companies offer various technologies to others as a service. There are three main types of web services:

- First is Data as a Service (DaaS), in which a type of data (financial, historical, geographical, etc) is offered by a server. An example of this is the geographical information system company Esri’s ArcGIS service, which gives users access to geographical mapping data and analytics.
- Second is Platform as a service (PaaS), in which a company offers infrastructure for developers so that they can focus on the developing and testing phases of the Software Development Life Cycle. This is the category that cloud platforms like microsoft azure, google cloud platform, heroku fall under.
- Third, we have Software as a Service (SaaS), which is what most people refer to as cloud apps. This category includes sites, apps, and tools that we use every day like spotify, youtube and discord.

2. API Characteristics

The Motivation Behind APIs

The existence of millions of servers and clients is complicated, to say the least, and brings up quite a few questions for developers of both servers and clients.

- **How should the world-wide web be organized?** Beyond just server and client nodes, developers need to know how each type of node should be designed.
- **What format should requests be made in?** Client-side developers need to know how to format their requests for each server their code interacts with.
- **How do developers know how to interact with web services?** As we saw, there are so many types of aaS's out there!
- **How do clients know how to make requests to a specific server?** The accepted protocols and procedures may vary from server-to-server!
- **What should a client expect in return from a server?** Client code for handling the server's response can't be written until the expected format is known.
- And finally, **how should clients and servers interact with each other?**

This is where APIs come in. APIs, or Application Programming Interfaces define how the client-side/front-end can load data from the server-side/back-end. They specify how web services should be used, how requests and responses should be formatted, as well as how clients and servers should interact with each other.

The Four Characteristics of the Modern Web API

First off is **encapsulation**. A web API should encapsulate a web service's functionality into component-ized systems. Since the web is so complex, it's important to minimize interdependencies among components.

Next is **scalability**. The network of nodes that make up the web must be able to handle the addition of new nodes as more and more of the physical world moves to the cloud. APIs help make the web more flexible.

Third, web APIs should be made to **handle large amounts of data**. The networks that all servers and clients run on must be capable of coping with server/client demands and heavy network traffic.

The fourth and final characteristic is **abstraction** via interfaces—the servers and clients on the web should be able to work with each other, regardless of the specific API they're using.

3. Modern Architectural Styles

With those characteristics in mind, when designing and building an API, it is important that the developer takes into consideration which architecture he or she would like to use.

Architectures focus on splitting a web service into components, defining how they interact with data, and how they interact with each other.

REST

Representational State Transfer, or REST, was introduced by Roy Thomas Fielding in his 2000 dissertation in response to SOAP. It’s designed to transfer media from server to client.

REST has a server-based **architectural focus**. A server will expose resources (aka any media, such as images, code, videos etc) through URLs, which are called resource identifiers.

Data is represented by any media type supported by HTTPS . This can be anything ranging from an image to a PowerPoint to a javascript module.

REST **operations** are also CRUD based, which means it supports operations to create, read, update, and delete server data. These operations are performed via HTTP (or HTTPS to be more secure) verbs, specified by the client when a request is made.

Caching is built-in to help improve server response time.

GraphQL

More recently, GraphQL was created internally by Facebook in 2012, and then later open-sourced in 2015. Starting in November 2018, GraphQL now runs as a nonprofit under the Linux Foundation. It acts as a Querying Language for clients to get data from a server.

Its **architectural focus** is client-based. Data from the server is exposed to the client via a schema, which is a data structure that "defines what queries are allowed to be made, what types of data can be fetched, and the relationships between these types." Clients are able to request any data from the server, which is exposed through an endpoint at the root node of an underlying graph

When it comes to **data representation**, queries are structured in a uniform interface, exactly like their responses, but on top of that, responses only contain the data that was requested in the query, nothing more, nothing less.

GraphQL supports QMS **operations**, or the ability to query, mutate, and subscribe to updates in data.

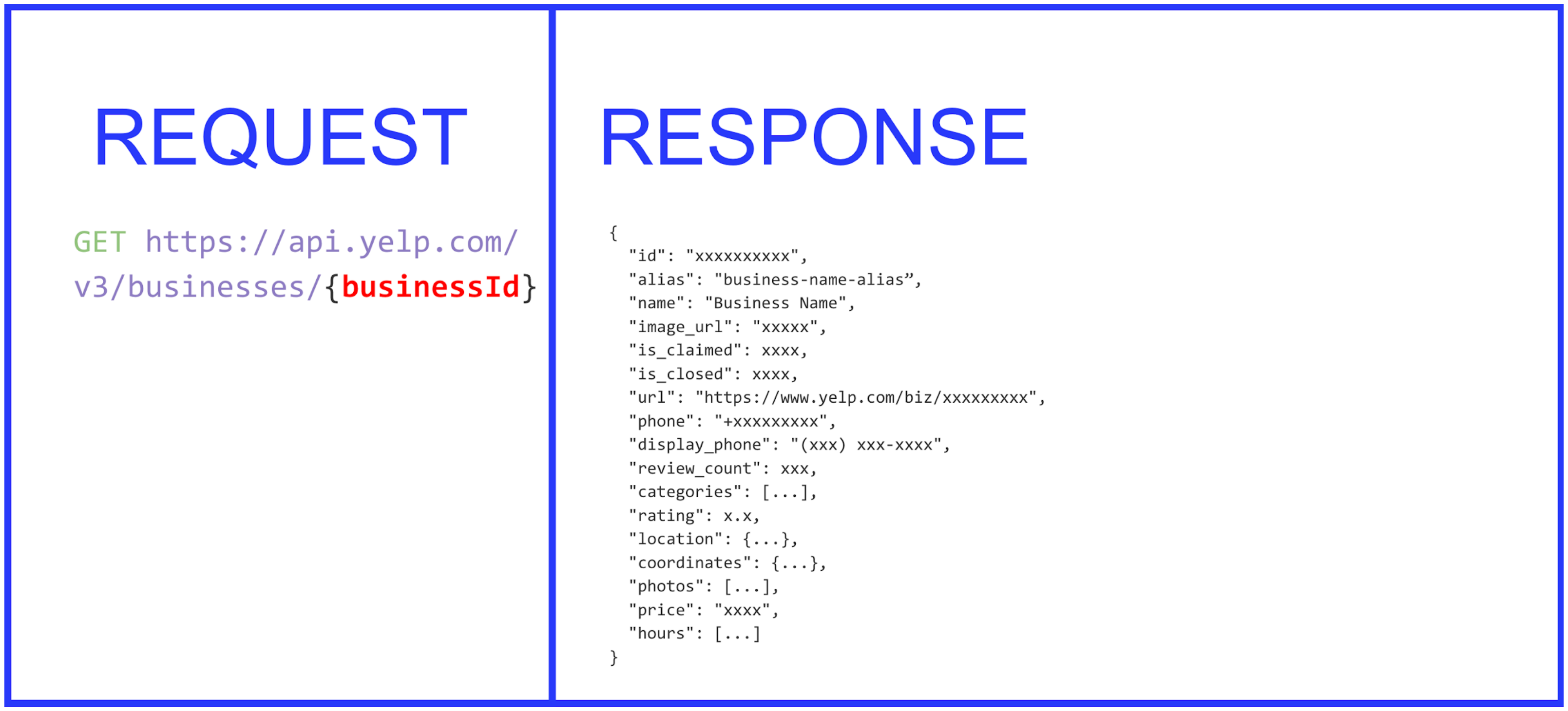
On the other hand, **caching** is not built-in, but is available through 3rd party libraries since caching can be complicated when it comes to servers whose data is updated frequently.

4. Case Study: Yelp | REST vs GraphQL

A case study of Yelp's developer tools can help demonstrate the differences between a RESTful API and a GraphQL-based API.
Yelp offers developers access to information on local businesses and their reviews in the form of two APIs. The legacy (aka original) API, Yelp Fusion, is the RESTful API, and the newer API is the Yelp GraphQL API.

REST APIs are designed to transfer media from server to client.

The RESTful Yelp Fusion API provides developers with the resource identifier https://api.yelp.com/v3/businesses/{businessId} to grab data about a business.



```
GET https://api.yelp.com/
v3/businesses/{businessId}
```

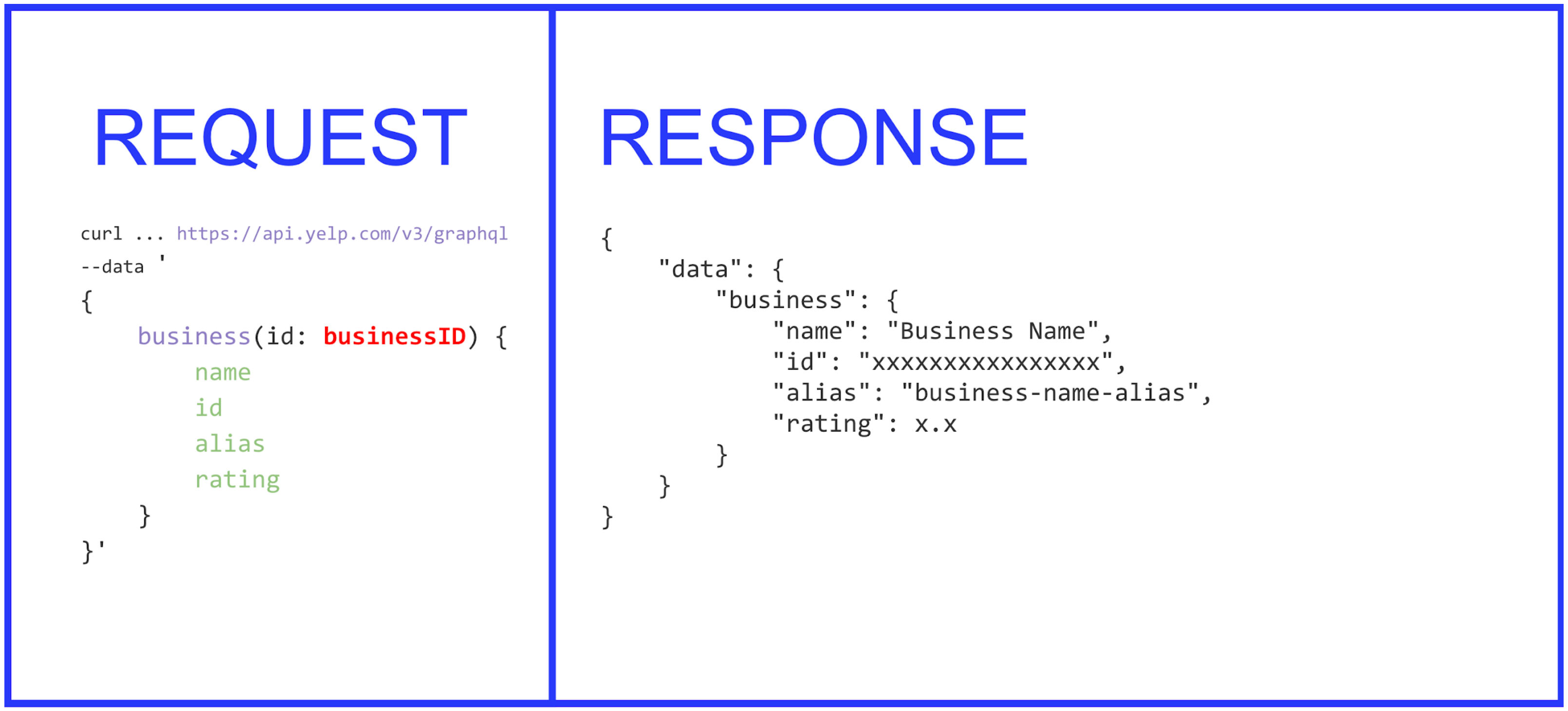
```
{
  "id": "xxxxxxxxxx",
  "alias": "business-name-alias”,
  "name": "Business Name",
  "image_url": "xxxxx",
  "is_claimed": xxxx,
  "is_closed": xxxx,
  "url": "https://www.yelp.com/biz/xxxxxxxxx",
  "phone": "+xxxxxxxxx",
  "display_phone": "(xxx) xxx-xxxx",
  "review_count": xxx,
  "categories": [...],
  "rating": x.x,
  "location": {...},
  "coordinates": {...},
  "photos": [...],
  "price": "xxxx",
  "hours": [...]
}
```

On the left, we have a client making a request to the server. The HTTP GET verb specifies that we would like to perform a “read” operation on the business with the given businessID.

On the right, we see that the server has responded with the info for that specific business.

The response in this scenario would return all of the info about the business with the given ID. While the client may only need to grab the business’s name and address, the server will return not only the name and address of the business but also the phone number, photos, Yelp rating, price levels and hours of operation. This is where the GraphQL API comes into play.

The GraphQL-based API provides developers with a single endpoint, https://api.yelp.com/v3/graphql. This can then be used by clients to make any request in the form of a query.



```
curl ... https://api.yelp.com/v3/graphql
--data '
{
    business(id: businessID) {
        name
        id
        alias
        rating
    }
}'
```

```
{
    "data": {
        "business": {
            "name": "Business Name",
            "id": "xxxxxxxxxxxxxxxx",
            "alias": "business-name-alias",
            "rating": x.x
        }
    }
}
```

On the left, we have a client making a request to the server. This request is performing a “query” operation on the business with the given bussinessID. Contrary to the RESTful Yelp Fusion API, the GraphQL API allows for more customizations when it comes to what data the server should respond with.

On the right, we see that the server has responded with only the info we requested for that specific business.

5. API-First Development

Nowadays, it's common to build an API and test it before it's even finished! Tools like Postman can help get the job done.

POSTMAN PRESENTS
LIFECYCLE OF AN API
DESIGN & MOCK • DEBUG • AUTOMATED TESTING
DOCUMENT • MONITOR • PUBLISH

Long Story Short

- As a result, the web services that run on these nodes are
    - flexible: API architectures provide different "flavors" while still maintaining the four Web API requirements.
    - efficient: Since APIs act as a contract between nodes, both clients and servers know what to expect from each other.
    - and modular: Web services that follow an architecture are able to run separately and undergo code changes and updates without affecting the web as a whole.

therefore making the web a better experience for developers and users alike!