



RapidAPI

comics

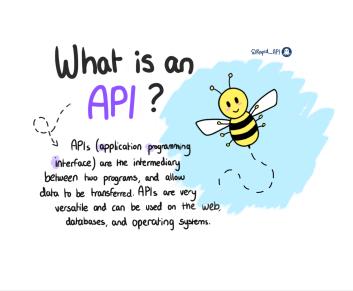
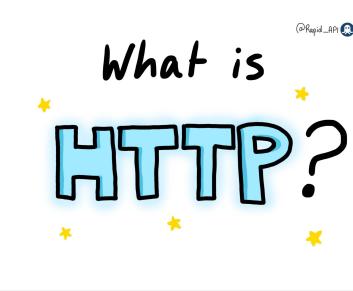
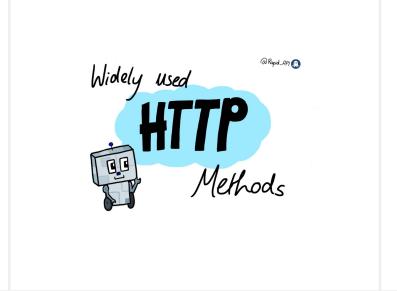
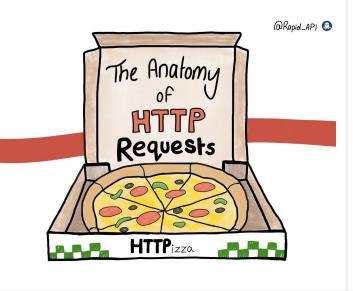
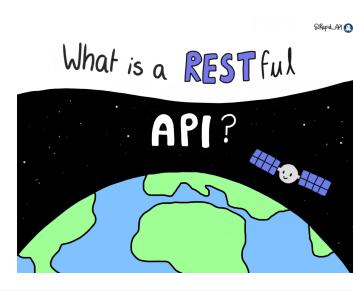
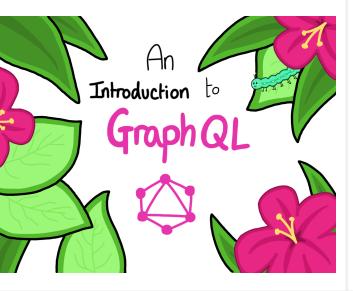
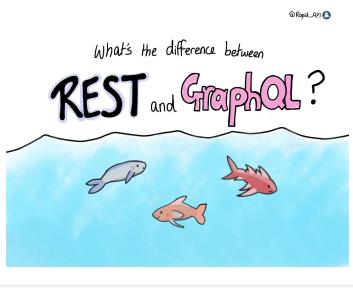
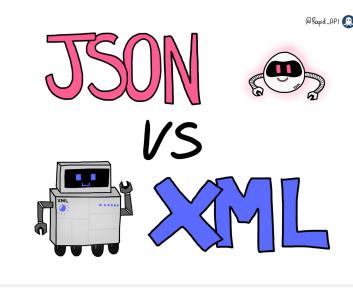
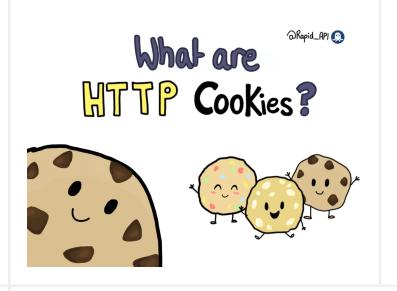
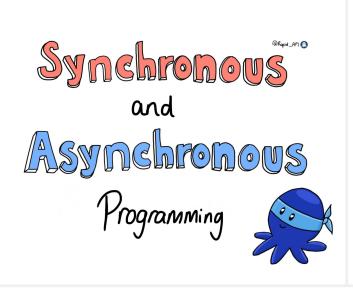
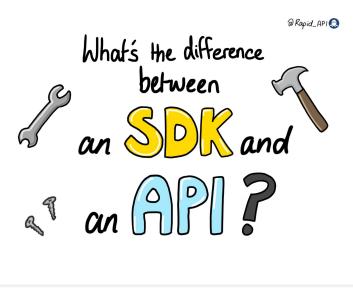
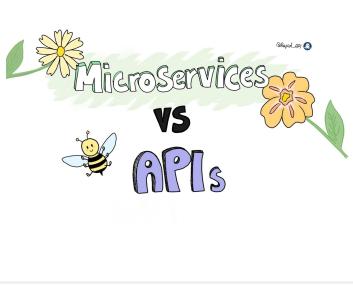
eBook

APIs, HTTP,
GraphQL, DNS,
Webhooks,
best practices,
and more!



Learn API and Web development topics through fun
illustrations and analogies!

Table of Comics

| | | | | |
|---|---|---|--|--|
|  |  |  |  | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | | | |

What is RapidAPI

[**RapidAPI**](#) is the world's largest API Hub, where over four million developers find, connect, build, and sell tens of thousands of APIs.

What is an API

What is an API?

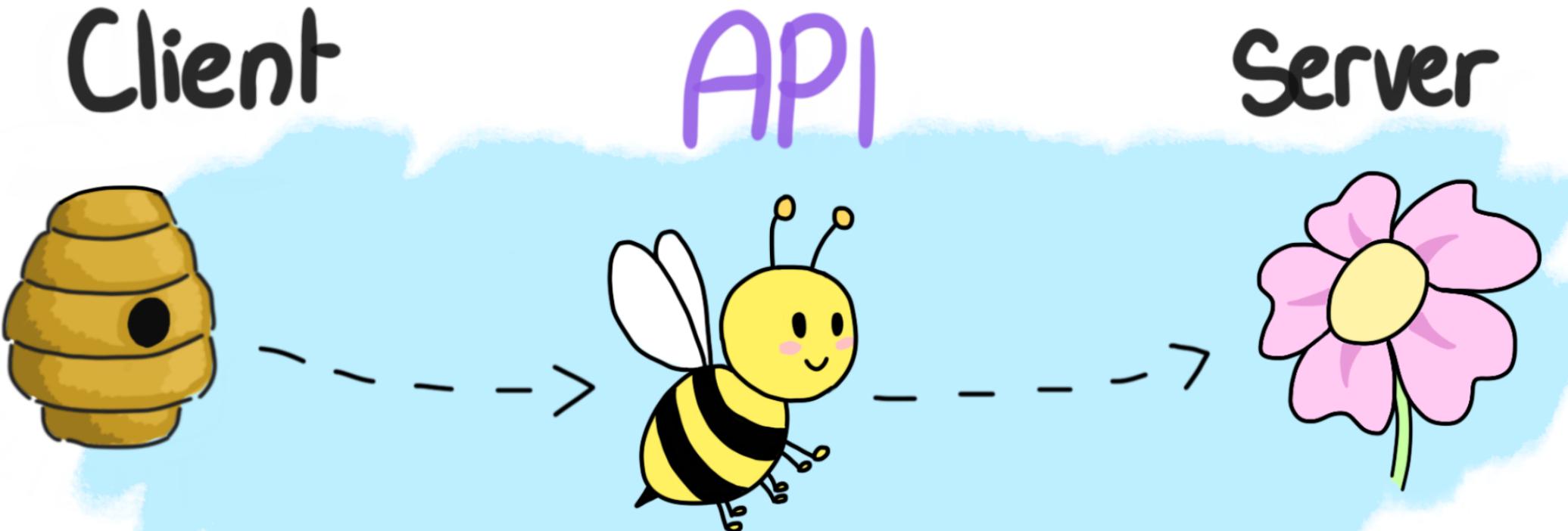
↓ APIs (application programming interface) are the intermediary between two programs, and allow data to be transferred. APIs are very versatile and can be used on the web, databases, and operating systems.



Request



An API call is initiated. This is the process of the client app submitting a request to a Server. APIs can be used to share data, embed content, and More.



Our worker bee acts like an API going to fetch the needed data for the client.

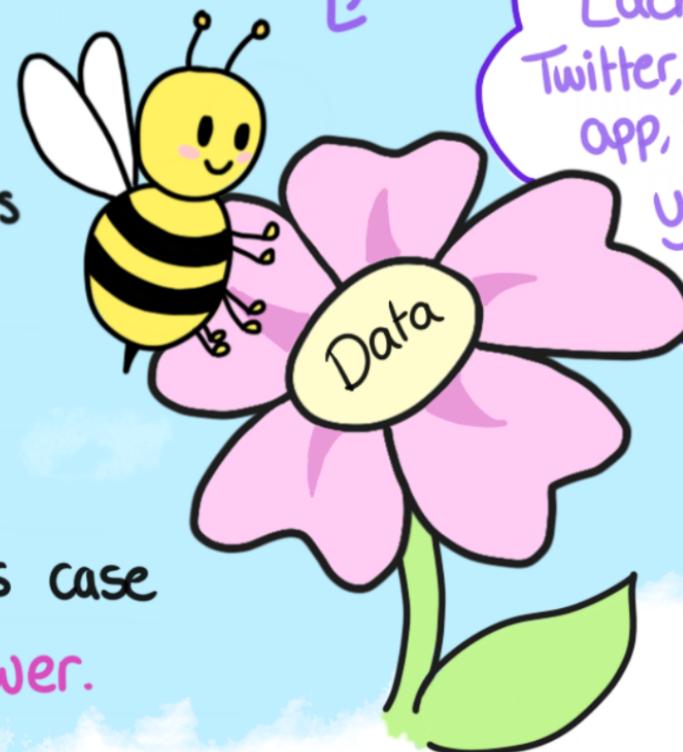
APIs use HTTP protocols to transfer data.

GET/nectar/pinkFlower

To find the right data, APIs have endpoints. Endpoints are essentially the URLs that navigate to the correct resource.

Our endpoint in this case is a pink flower.

Our bee (API)
Collecting nectar
(data)



FACT

Each time you open up Twitter, google maps, a weather app, and so many more, you're using APIs. APIs are everywhere!

@Rapid_API 

Response



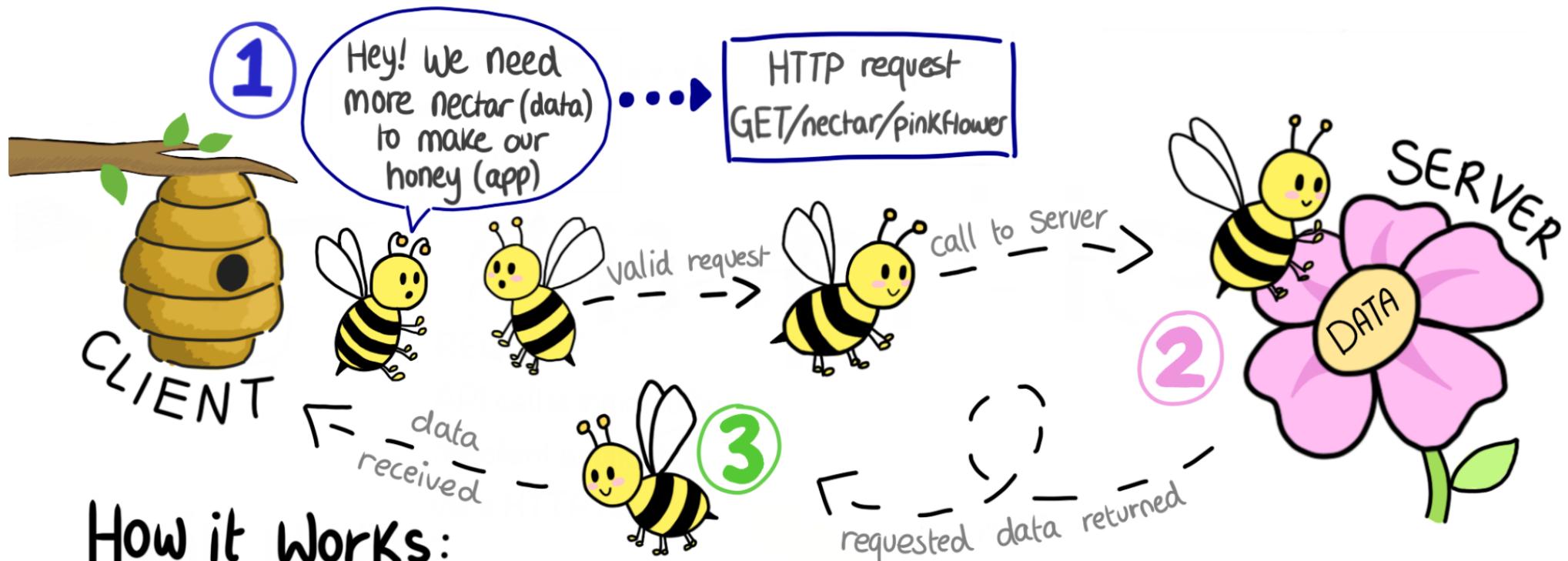
As long as the server (flower) can return the requested data (nectar) to the client successfully, then mission accomplished!

If the server can not return what the client asked for, the API will return the appropriate error message.



What is an API?

An application programming interface allows two programs to communicate. On the web, APIs sit between an application and a web server, and facilitate the transfer of data.



How it Works:

1 Request

API call is initiated by the Client application via a HTTP request.

2 Receive

Our Worker bee acts as an API, going to a flower (server) to collect nectar (data).

3 Response

The API transfers the requested data back to the requesting application, usually in JSON format.

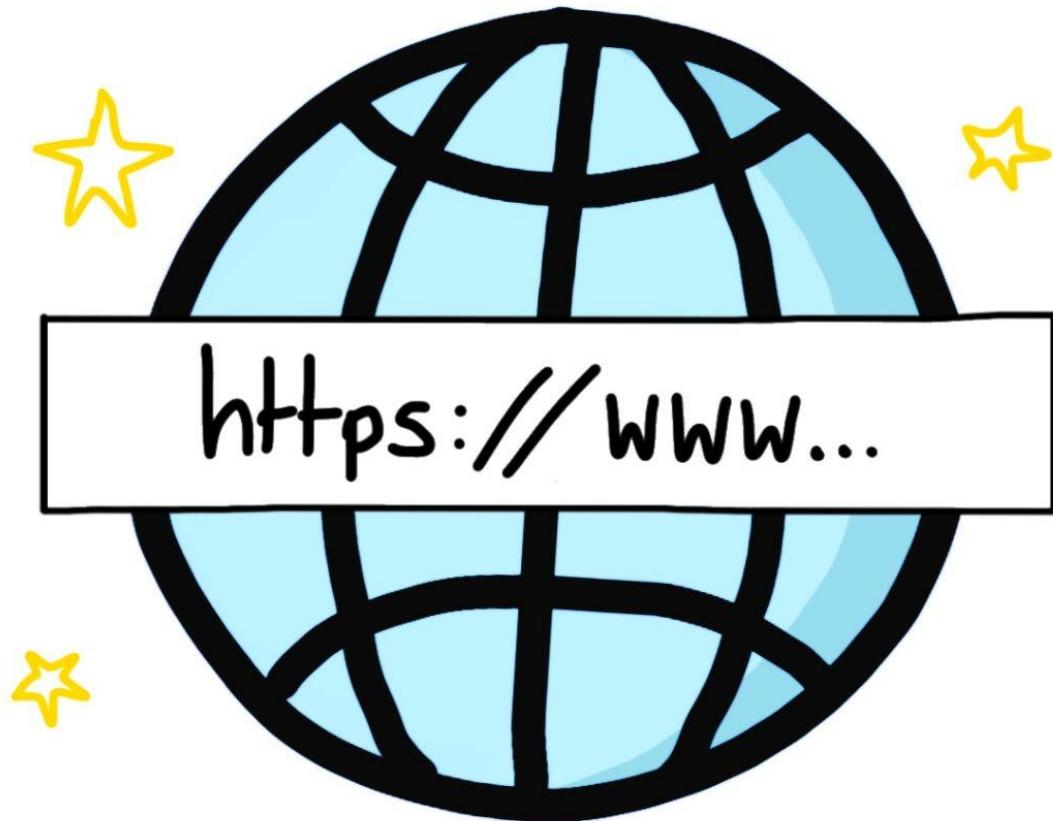
What is HTTP

What is

HTTP?



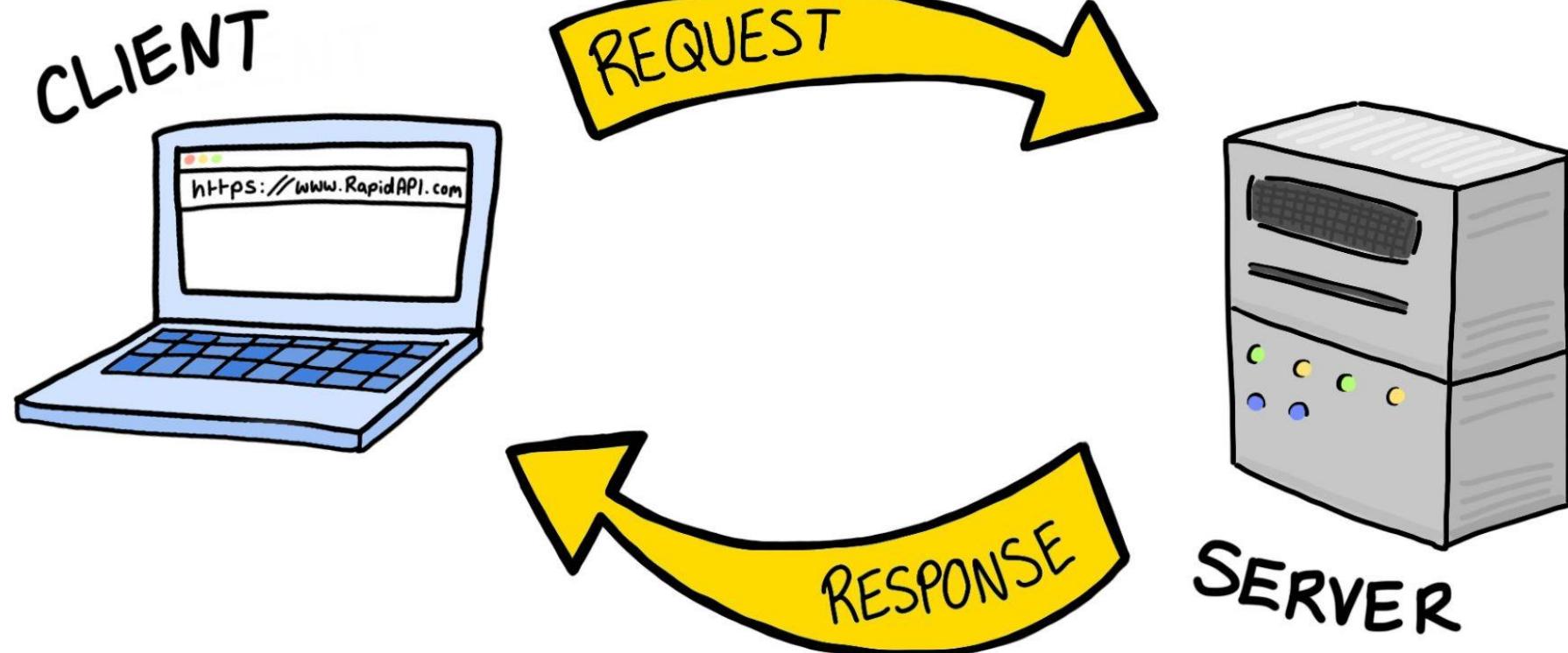
HTTP stands for Hypertext Transfer Protocol



HTTP is the foundation of the Internet and how it functions.



How HTTP works:



HTTP works in a request and response system.

The browser requests specific information, and the server will respond with it if it's available.

HTTP Status Codes

★ **1XX - Informational**

The Server returns a status code in its response to let the client know if the request was successful or not.

★ **2XX - Success**

Example: 200 (OK)

Successful with no problems.

★ **3XX - Redirection**

★ **4XX - Client Error**

Example: 404

File not found - resource was not located by the server.

★ **5XX - Server Error**

When you load a web page in your browser,
HTTP Requests are made to fetch and display it.



Stateless

HTTPS - The
Secure
version

Uses
Client-
server
model

Responses defined
with HTTP status

Codes

Features
of HTTP

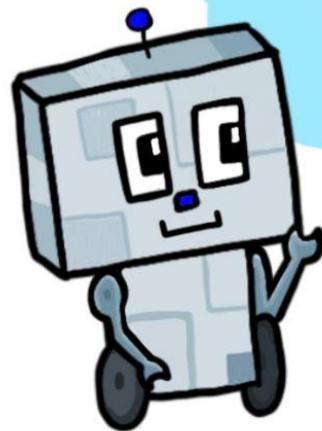
Custom HTTP
Headers can be
used in
requests and
responses

Application layer
protocol based on
TCP/IP

Widely used HTTP methods

Widely used

HTTP



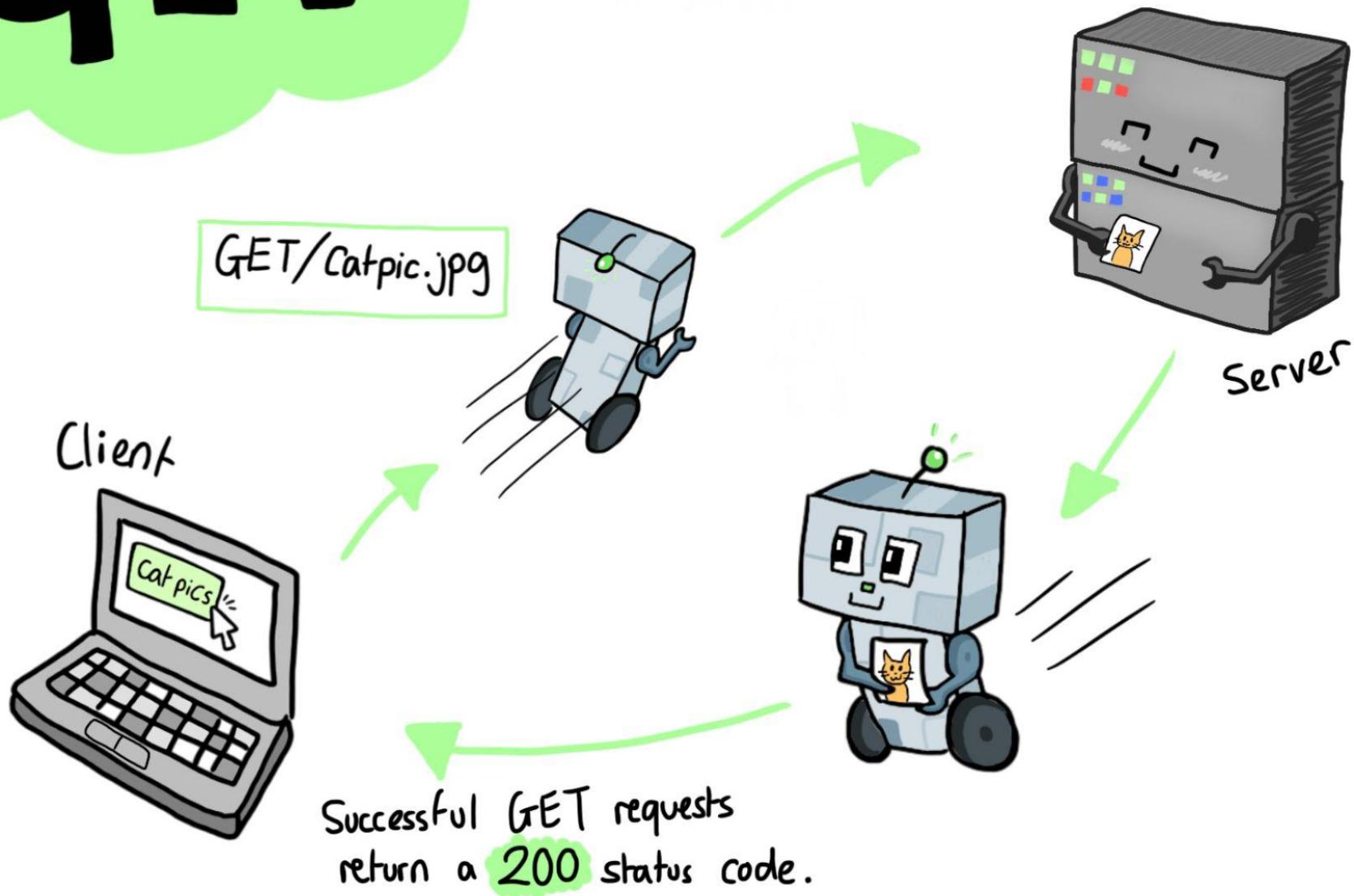
Methods



GET

GET requests retrieve a resource from a Server.

GET requests are cacheable and idempotent (do not affect the status of the server).



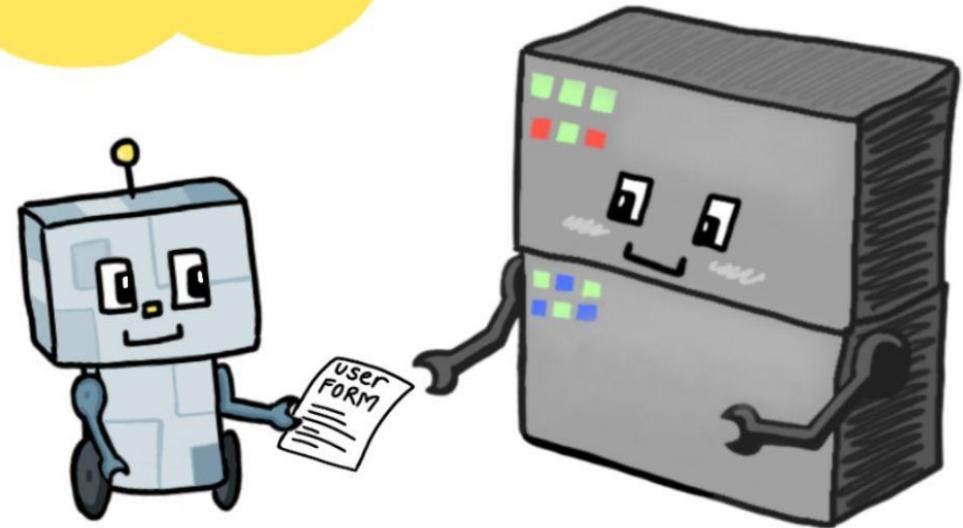
POST requests submit information to the Server.

They Change the state of the Server, so they are not idempotent.
They are also not cacheable.



POST / user-form.php

POST

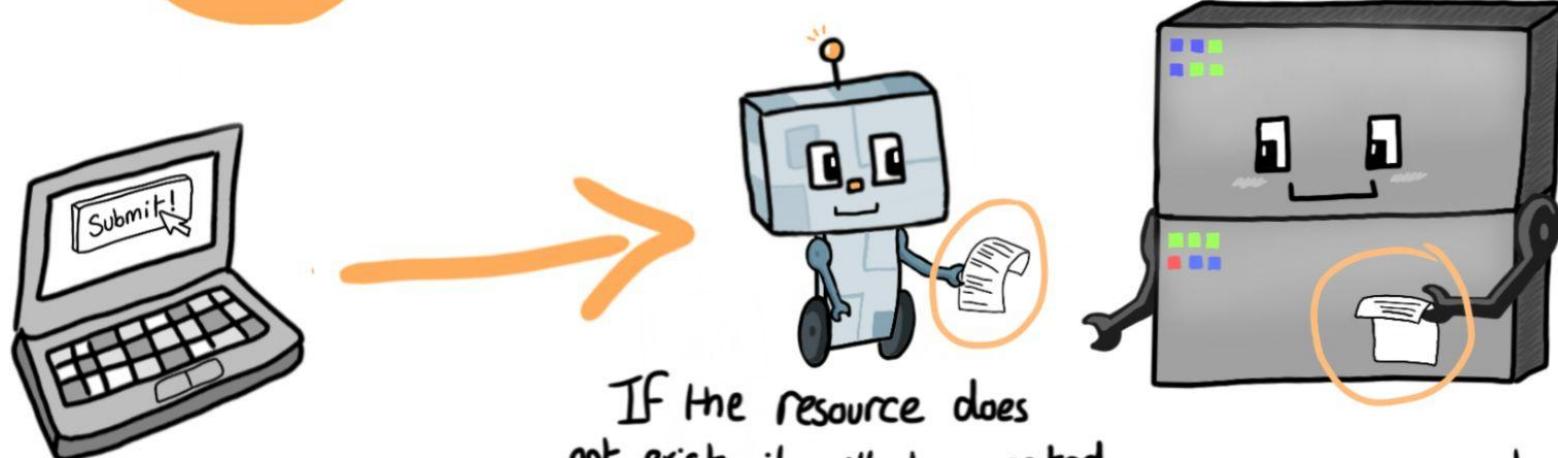


Successful POST requests will return a 201 status code (created).

PUT

PUT is also used to create/update resources,
but unlike POST, it is idempotent

Calling multiple of the
same PUT request does not
affect the Server.



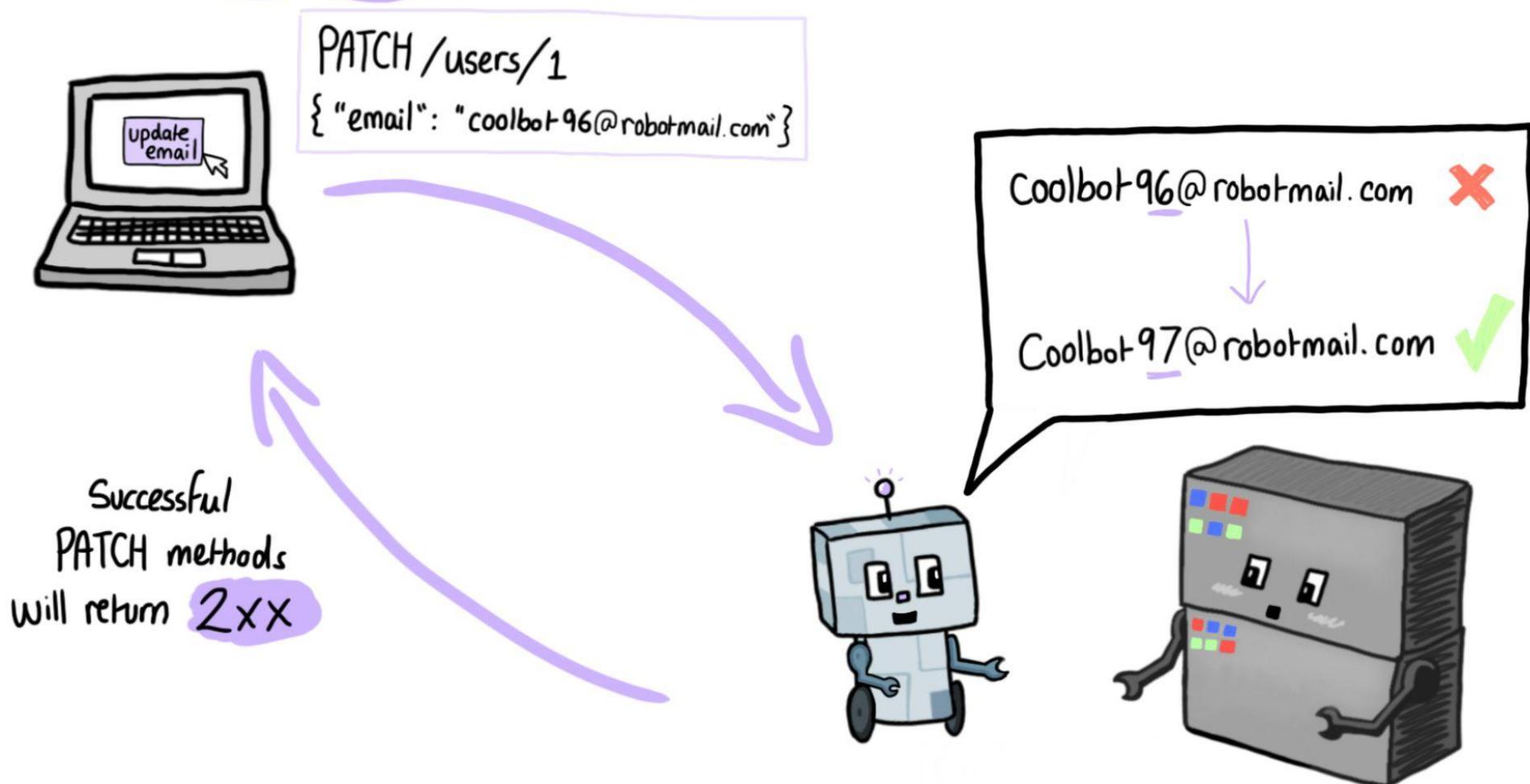
If the resource does
not exist, it will be created.

On Success,
will return a
200 (OK), 201 (created),
or 204 (no content) status.

If the resource already
exists on the Server, it will be
replaced with the new payload.

PATCH

Like PUT, PATCH also updates a resource, but PATCH only modifies a specified part of it instead of updating the entire resource.



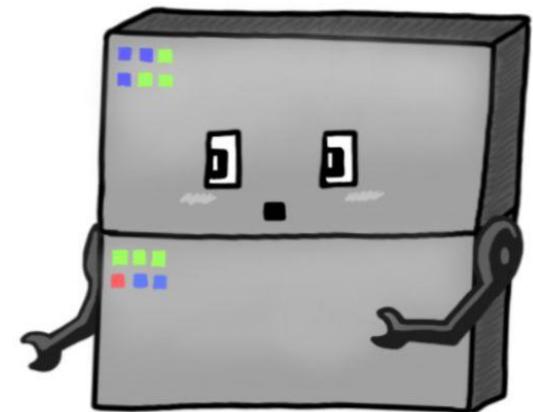
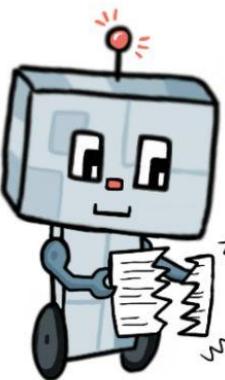
DELETE

The delete method
is idempotent.

As suggested, it
deletes a specified
resource on the Server.



DELETE /file.html

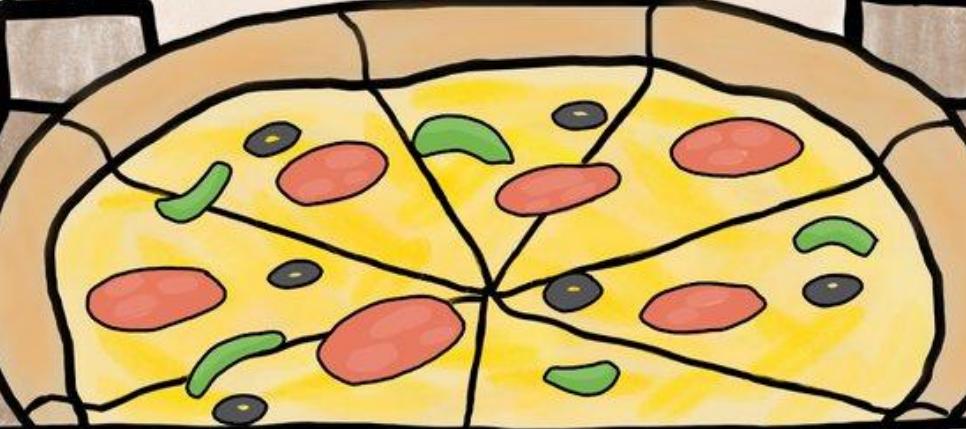


Successful
DELETE requests return
a 202 (accepted), 200 (OK),
or 204 (no content) status.

DELETE requests the
server to remove the resource
identified by the request URL.

The anatomy of HTTP Request

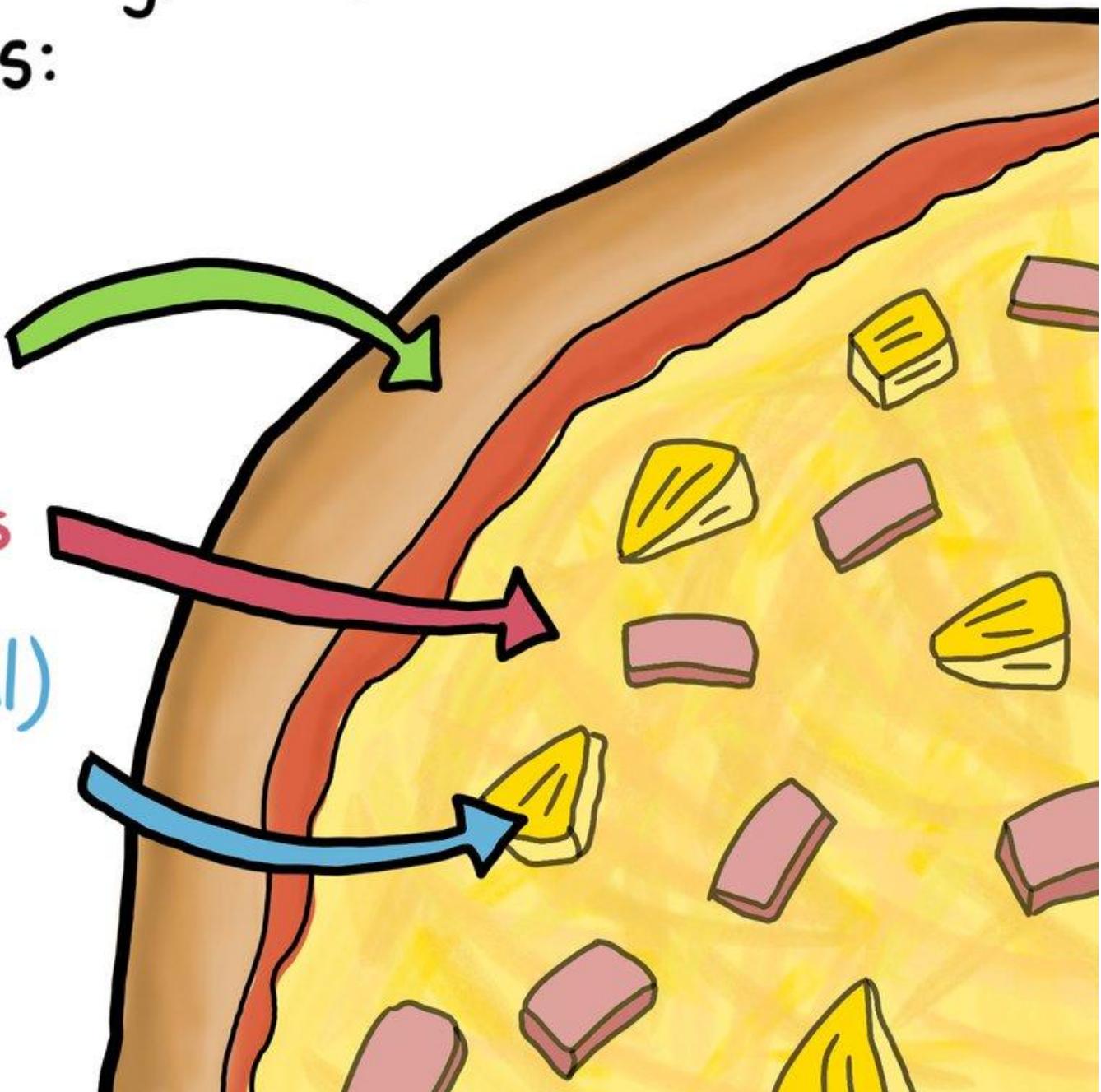
The Anatomy of **HTTP** Requests



HTTPizza

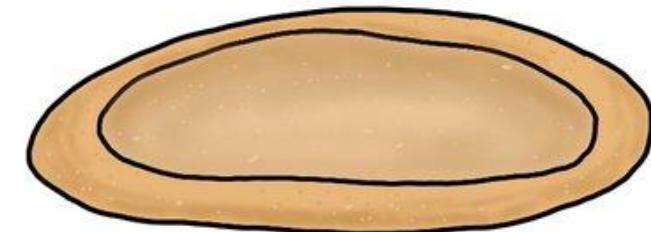
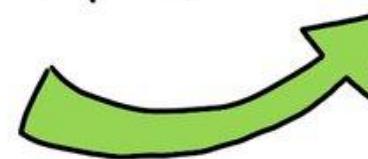
HTTP Requests typically Contain 3 things:

- ① Request line
- ② Header Fields
- ③ Body (optional)



① The Request line

The Request line is made up of three parts shown below. You can think of it as the foundation of the request, like our pizza's base.



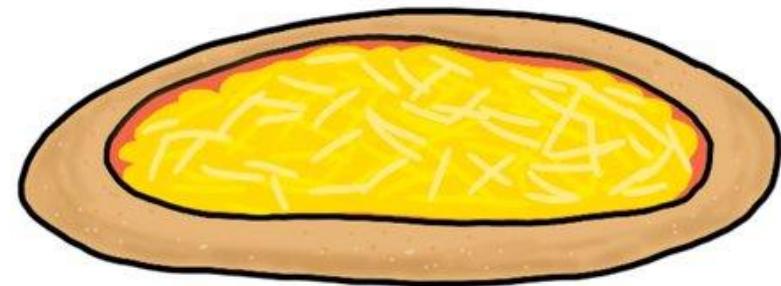
HTTP Method that Commands the Action to the Server e.g. GET, POST, DELETE



Resource URL, locates the resource

② Header fields

HTTP Headers provide extra information about the request to the server. There are many different headers, below is an example of some.



Indicates the data format



Content-Type: application/json

Length of
data in bytes



Content-Length: 30

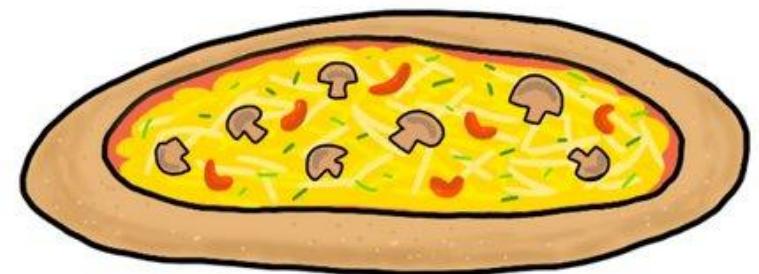
Accept-Language: en, de



Indicates the languages accepted by the client

③ Body (data)

The body is only needed if the HTTP Method is POST, PUT, or PATCH. It Contains the data being sent to the Server.



For example, the body could be the details of our pizza order.

```
{ "Customer": "Joe Robbins",  
  "base": "standard",  
  "cheese": "mozzarella",  
  "toppings": "mushroom" }
```

HTTP Request Structure in Full:

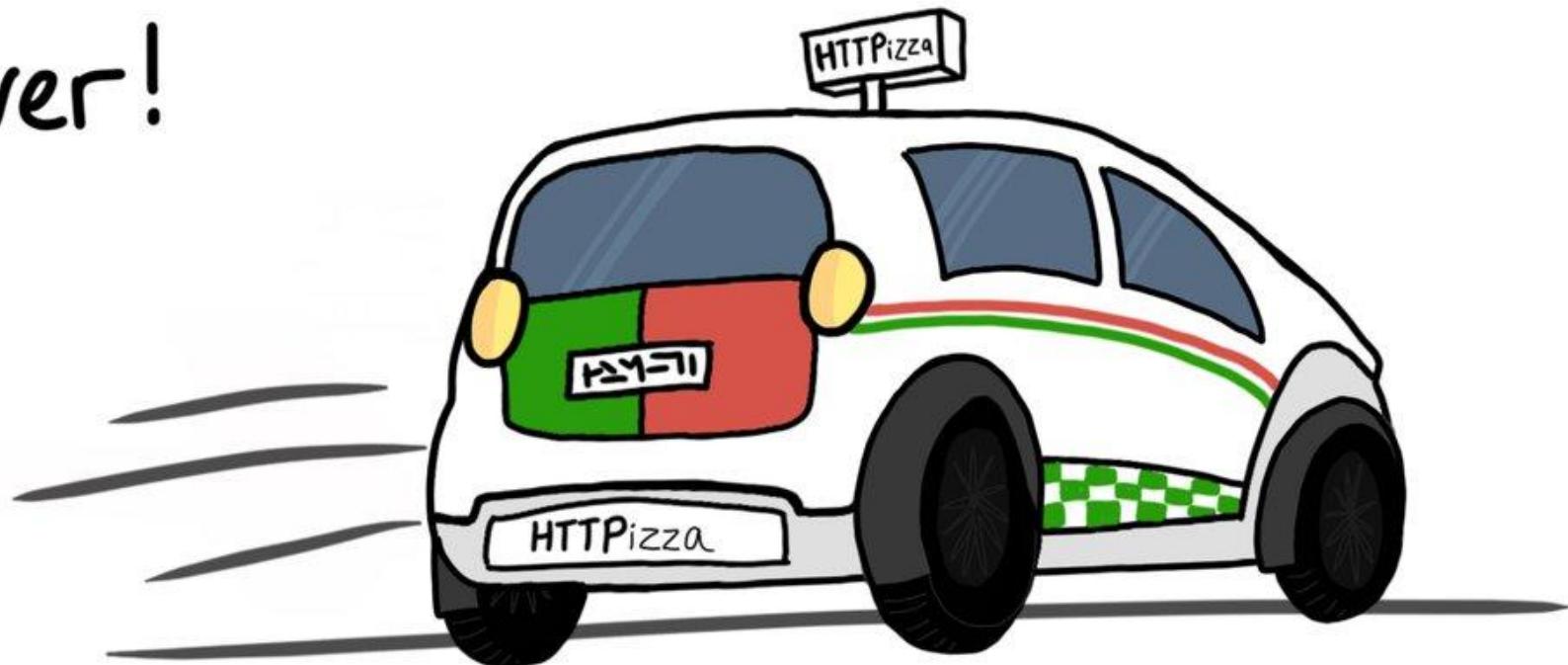
Request Line [POST/orders/{orderId} HTTP/1.1]

Headers [Host: httpizza.com
Content-Type: application/json
Accept-Language: en]

An Empty Line always follows HTTP Headers

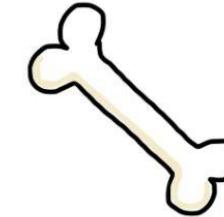
JSON body [{ "toppings": "mushroom" }]

Now the request is ready
to be sent to the
Server!

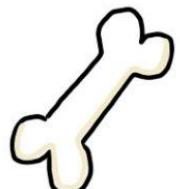


How API endpoints work

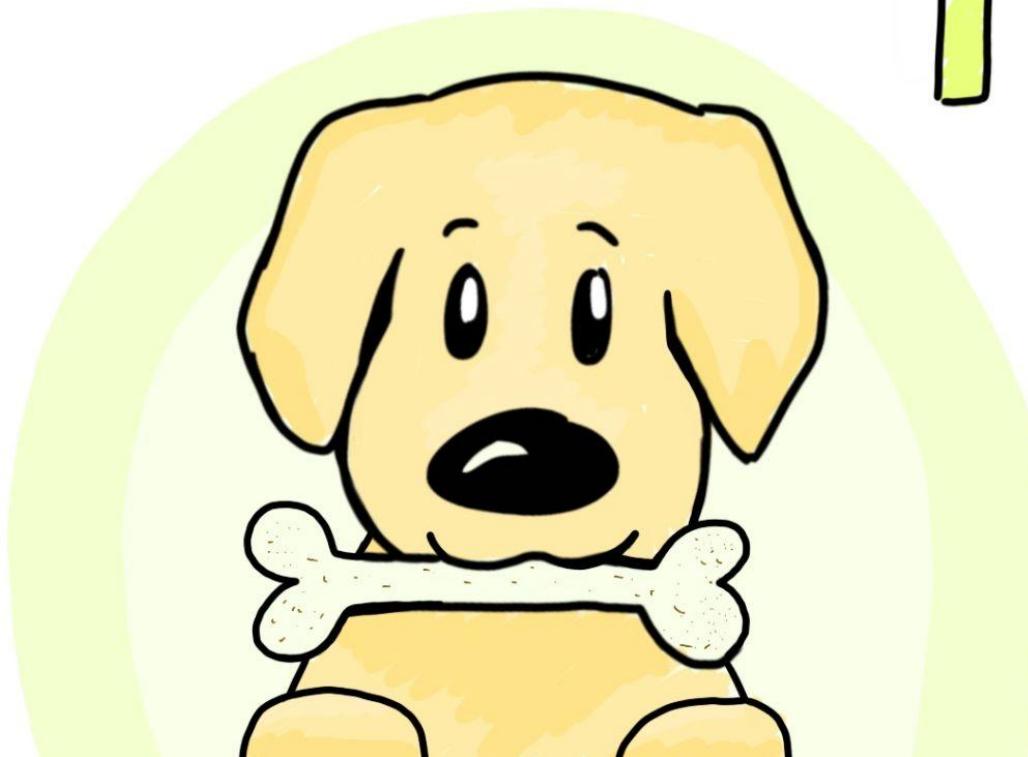
How API



Endpoints



Work



Firstly, what is an endpoint?

Endpoints are the communication touchpoints between the API and the server.

They are URLs that users can access to interact with specific resources and data.

They typically look like this

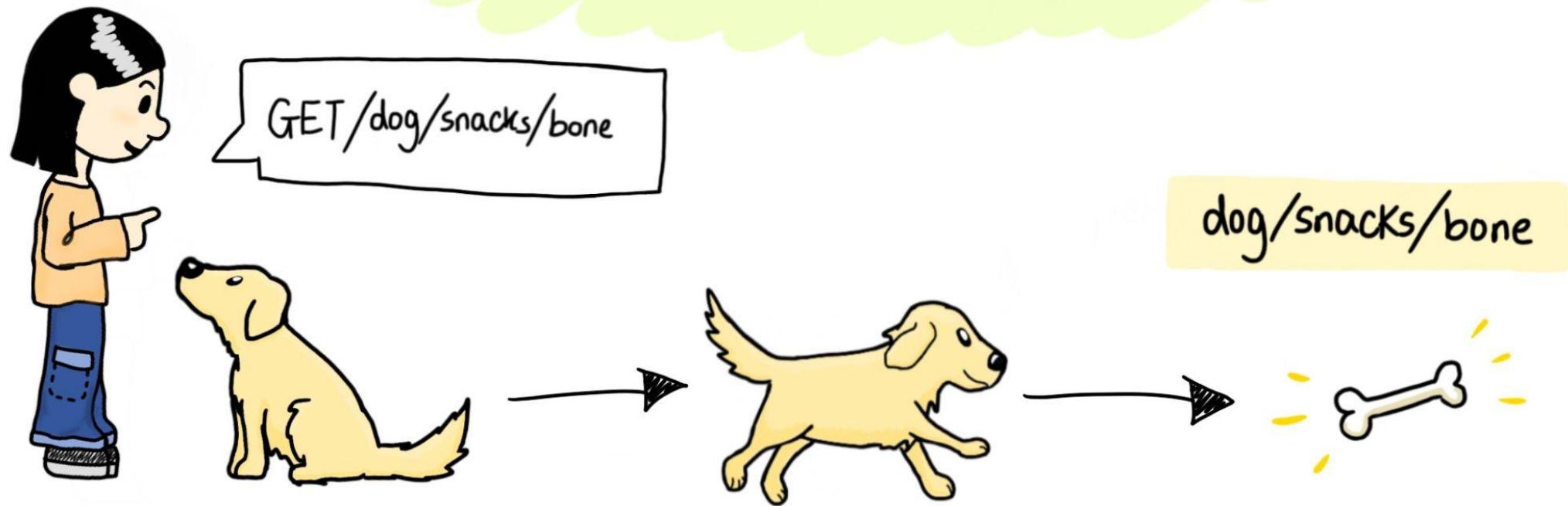


pets/dog/snacks/bone



The role of an endpoint:

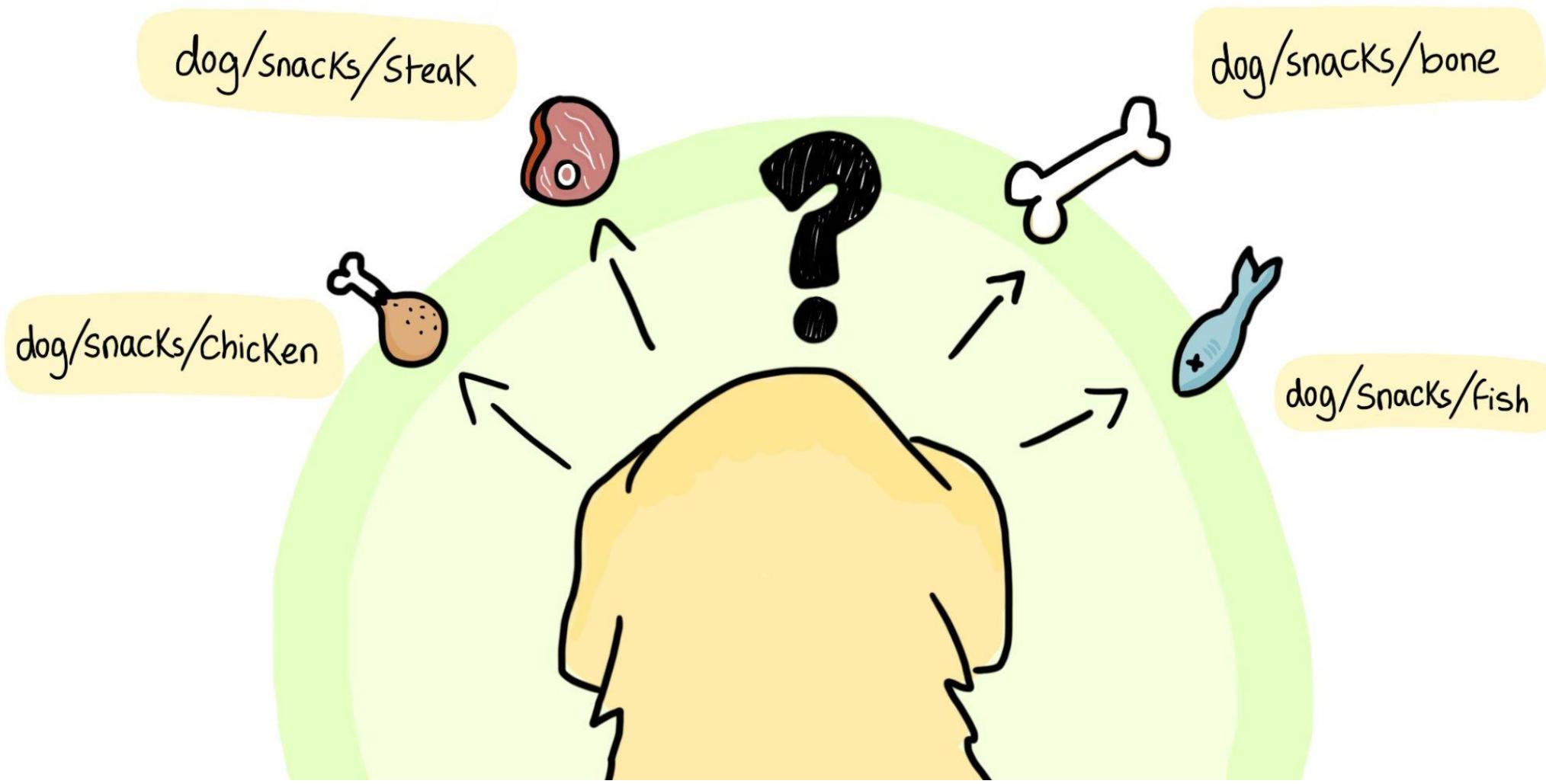
In this example, a resource is being fetched using the GET method.



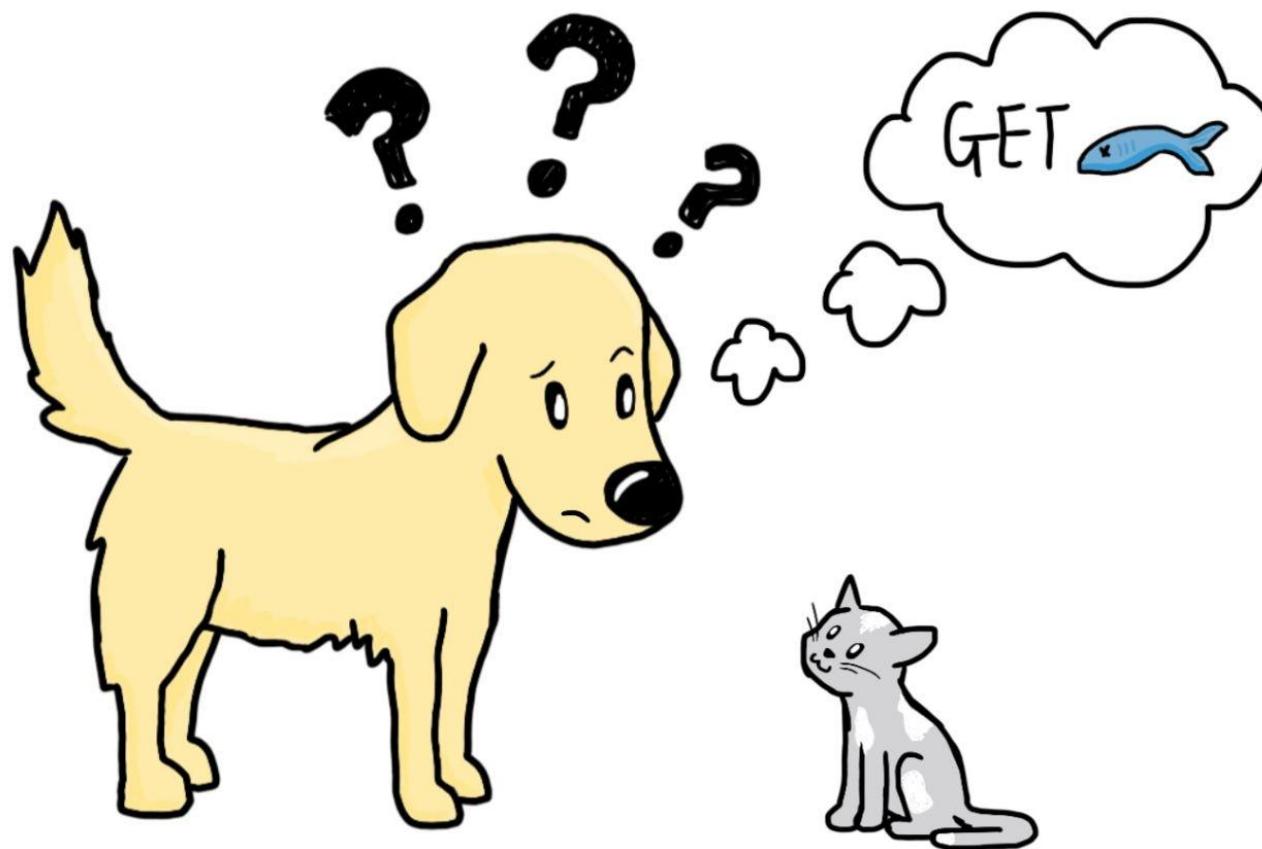
The endpoint dictates the location of a resource and is where the request is sent.

Each endpoint locates a unique piece of data.

APIs will have many endpoints representing different data, so it's crucial that they are clearly named after the entity they represent.

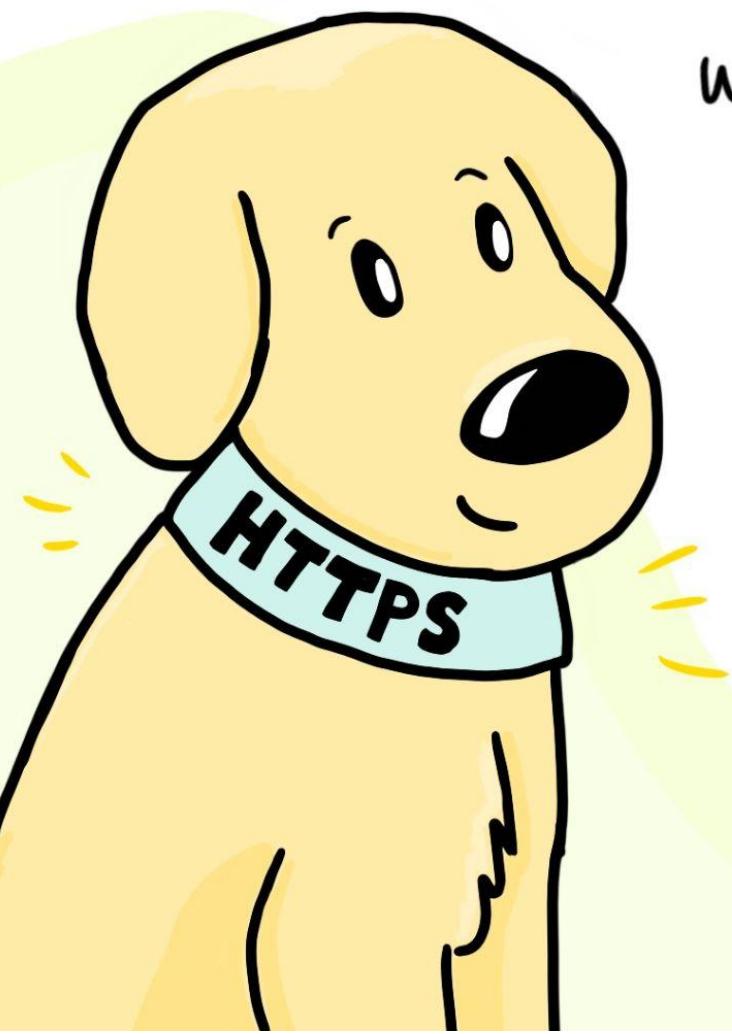


If endpoints are faulty or inaccurate, the API will not be able to locate the correct resource.



How to Secure endpoints?

There are various simple ways to secure endpoints, such as:



- Use HTTPS
- Use one-way password hashing
- Use Input validation
- Utilize rate limiting

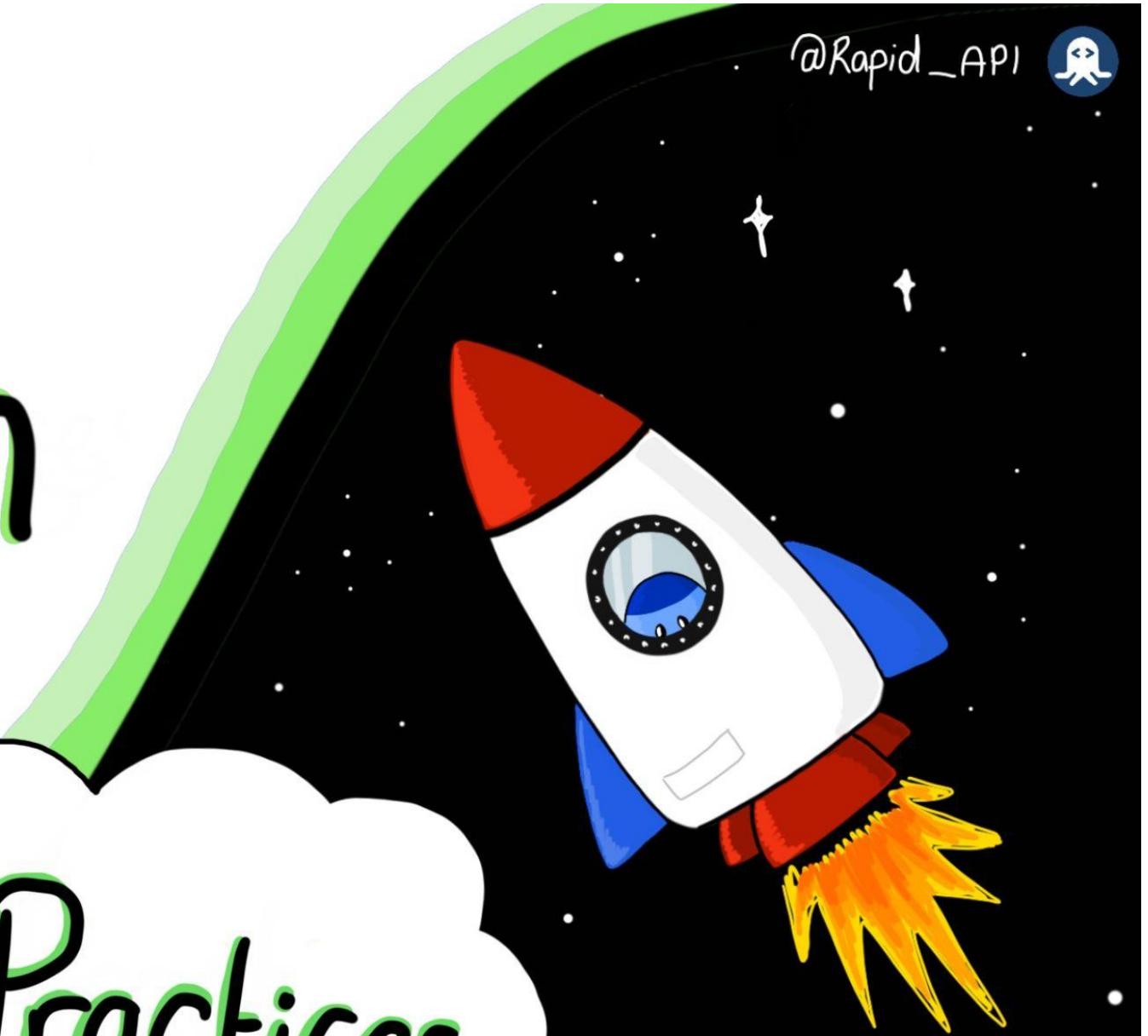
API design best practice

API design

@Rapid_API



Best Practices



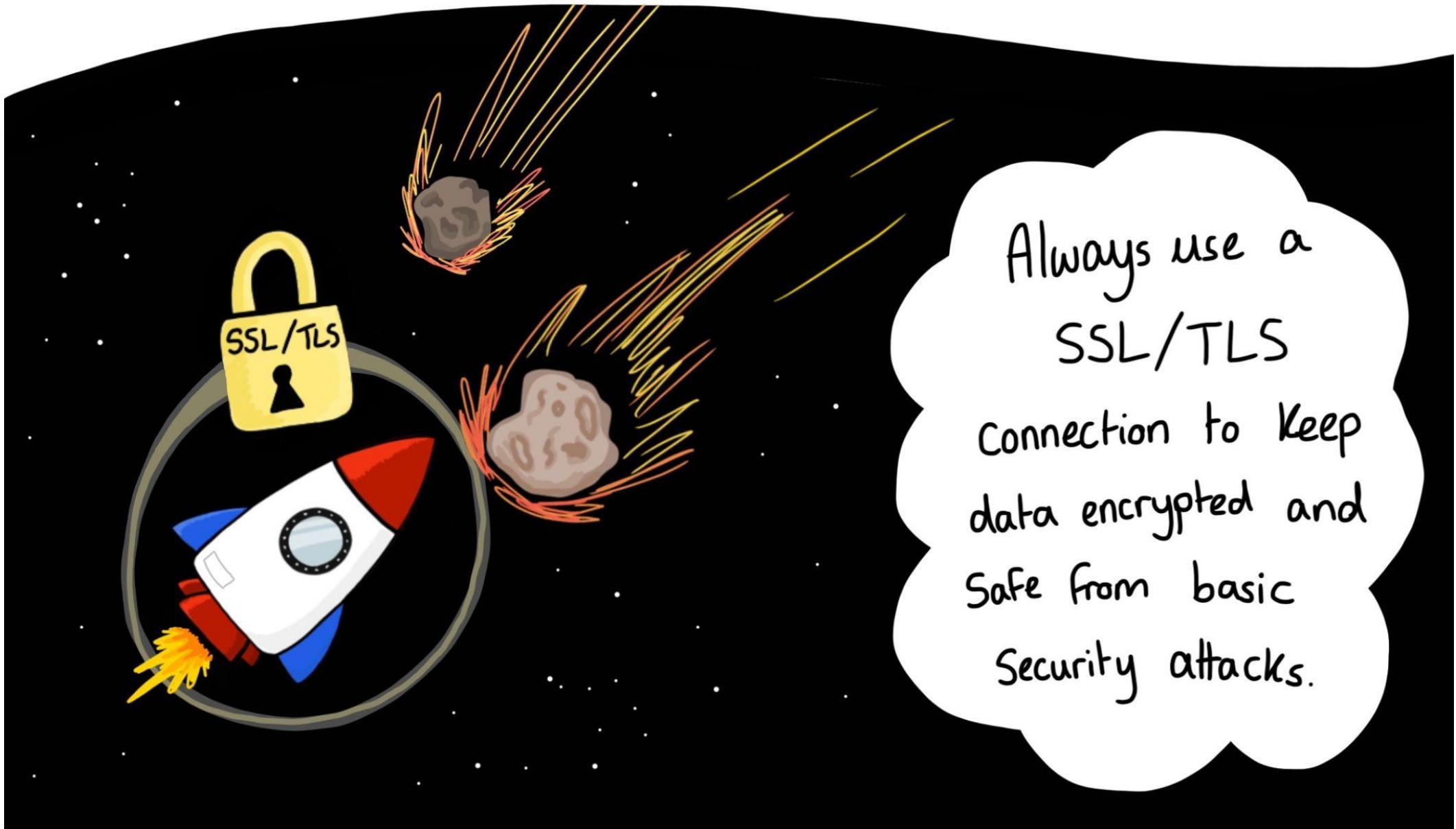
Use nouns for endpoint paths

Endpoint paths should always be named after the entity they represent.

The HTTP method is the verb indicating the action. ↴

GET /rocks/lunar

Maintain good Security Practices

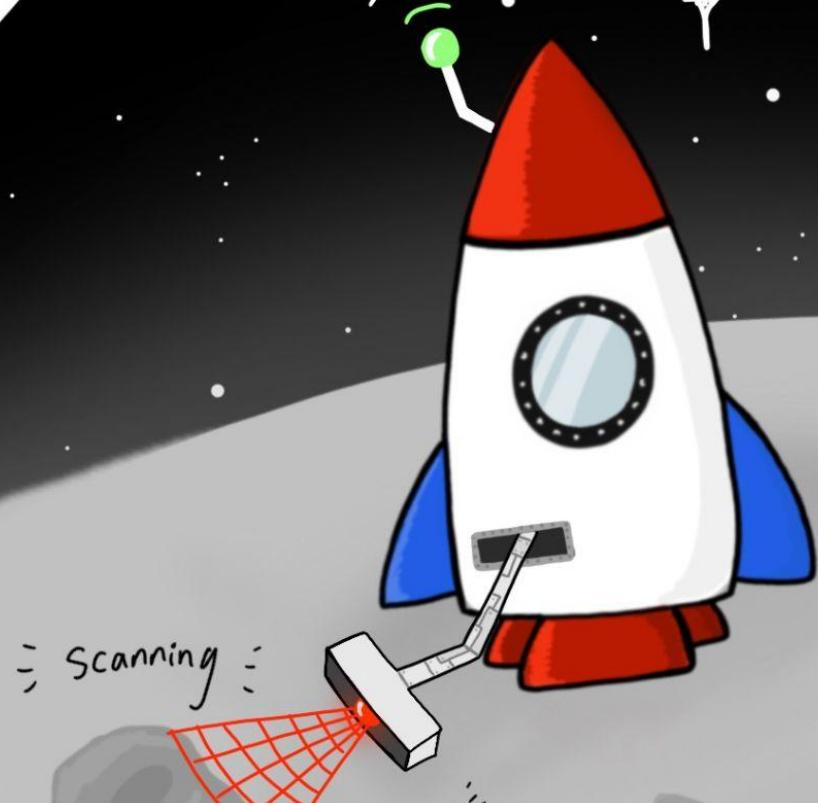


Use JSON

JSON is the standard for transferring data.

JSON
is widely supported and
all APIs should accept
JSON payloads.

JSON DATA



Pagination

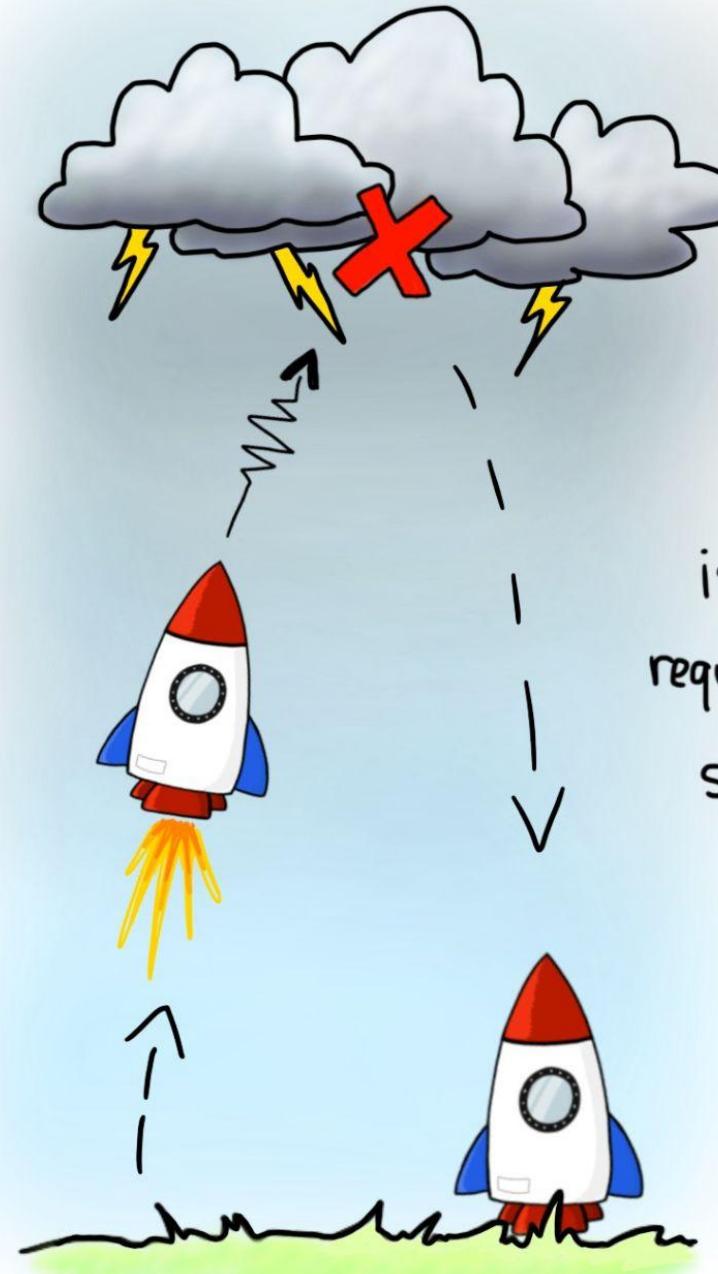


If endpoints return a large amount of data, it can slow a system down.

Pagination and filtering make data return in 'pages', which reduces the usage of server resources.

Implement timeouts

Timeouts cause a request to fail after a specified amount of time.



This means the client isn't left waiting on a request if there is an issue, such as a network connection issue.

Versioning

Making changes to your API
and updating it could potentially
break it.

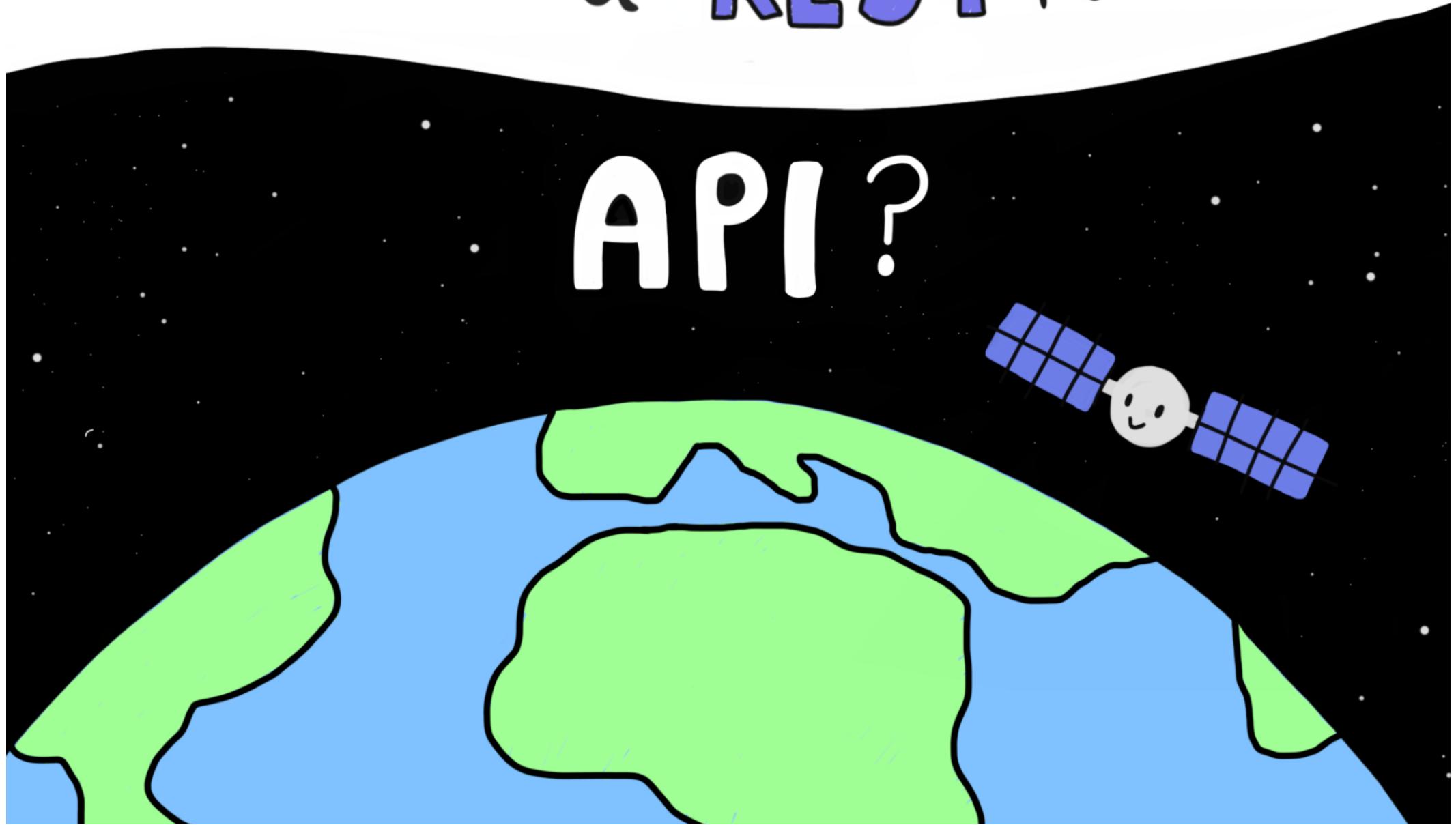
So it's crucial to
have previous versions
as a backup.



What is a RESTful API

What is a RESTful

API?



RESTful APIs follow

REST architecture

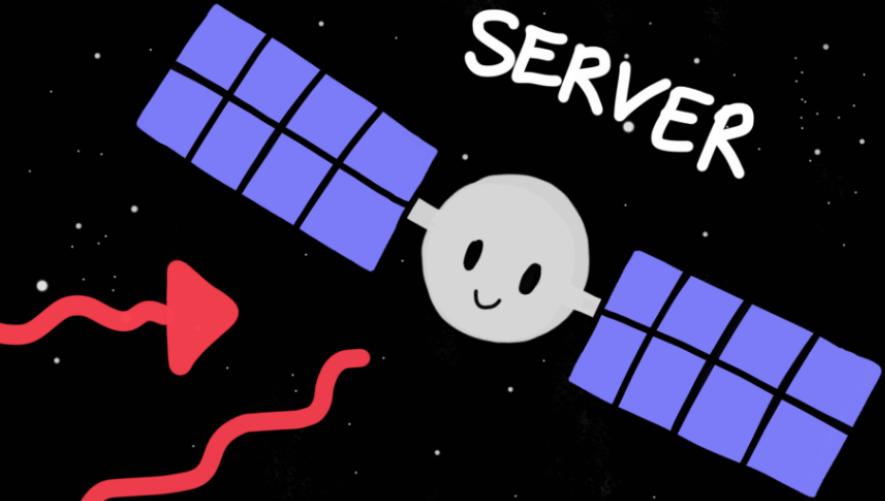
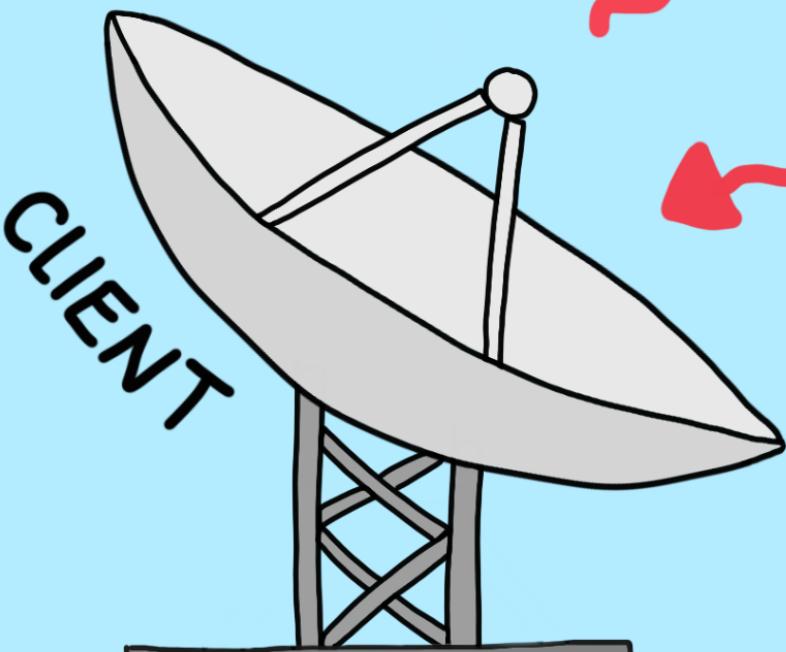
↳ Representational State Transfer

For an API to be RESTful, it must follow a number of principles, properties and constraints.

A deep dive into these principles 

1 Client-Server

The Client server principle Separates Client concerns and data Storage Concerns. All requests can only be made by the Client, and only the Server can respond.



By keeping these two independent, each can be modified without affecting the other.

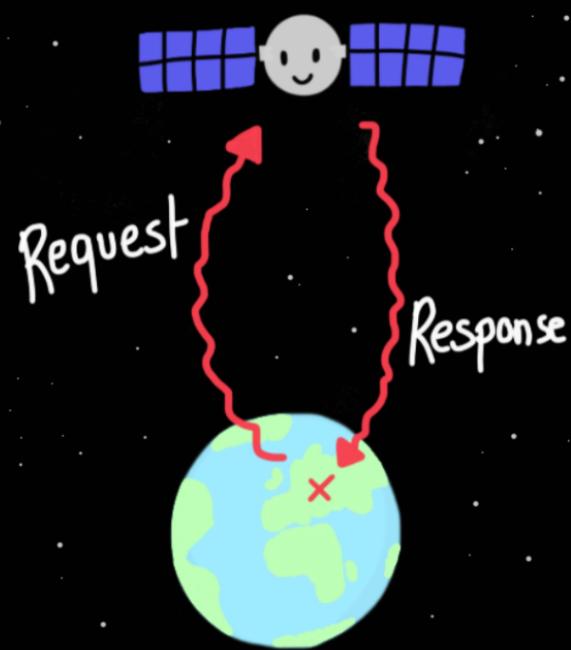
② Uniform Interface

This principle requires that all responses follow the Same format.

Applications and servers can use different languages, so a uniform interface as an intermediary makes communication easier and simplified.



REST APIs
use HTTP
as their common
Protocol.



Common HTTP Methods:

GET – Retrieves a resource.

POST – Creates a new resource.

PUT – Updates an existing resource.

DELETE – Deletes a resource.

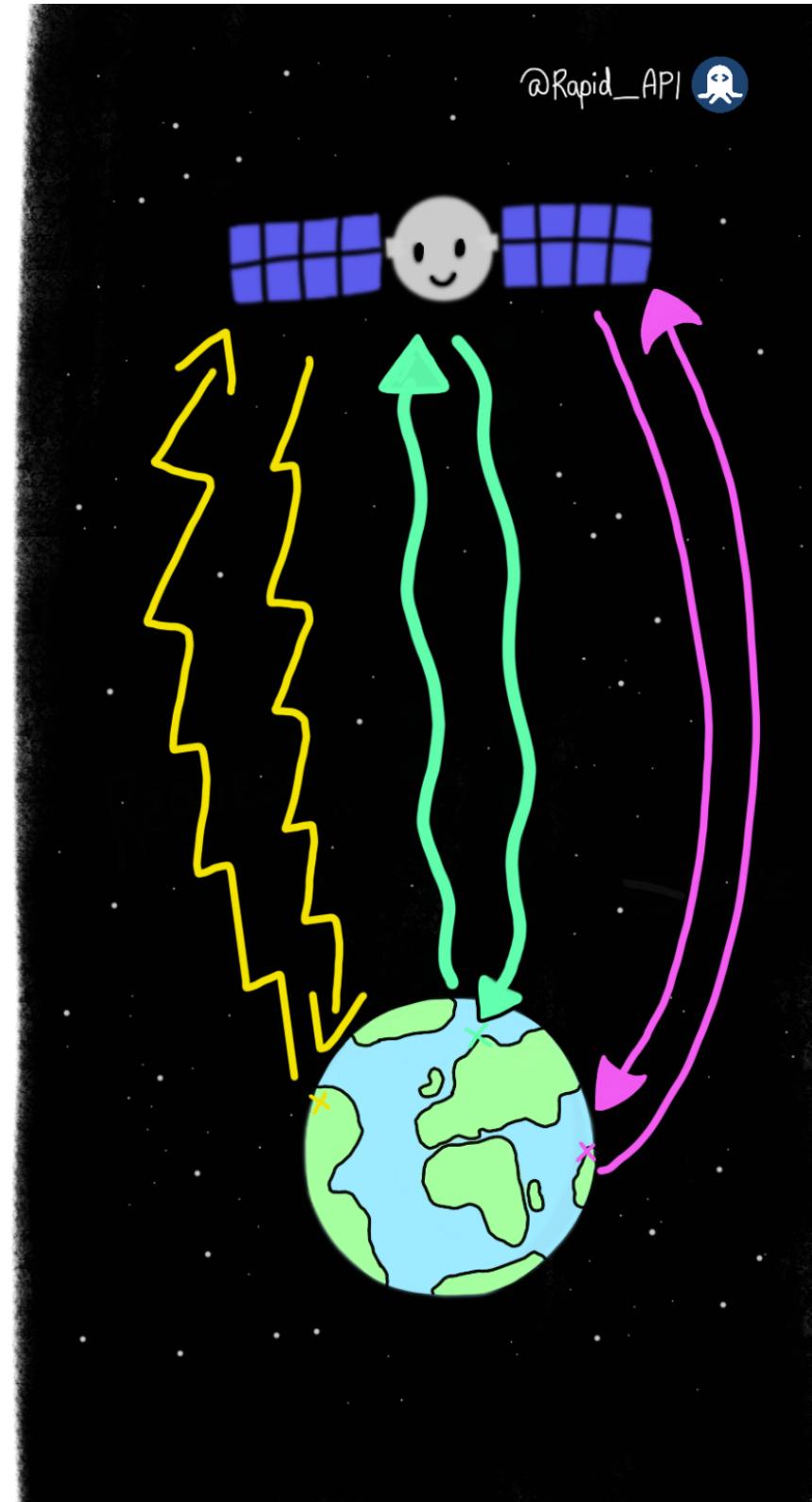


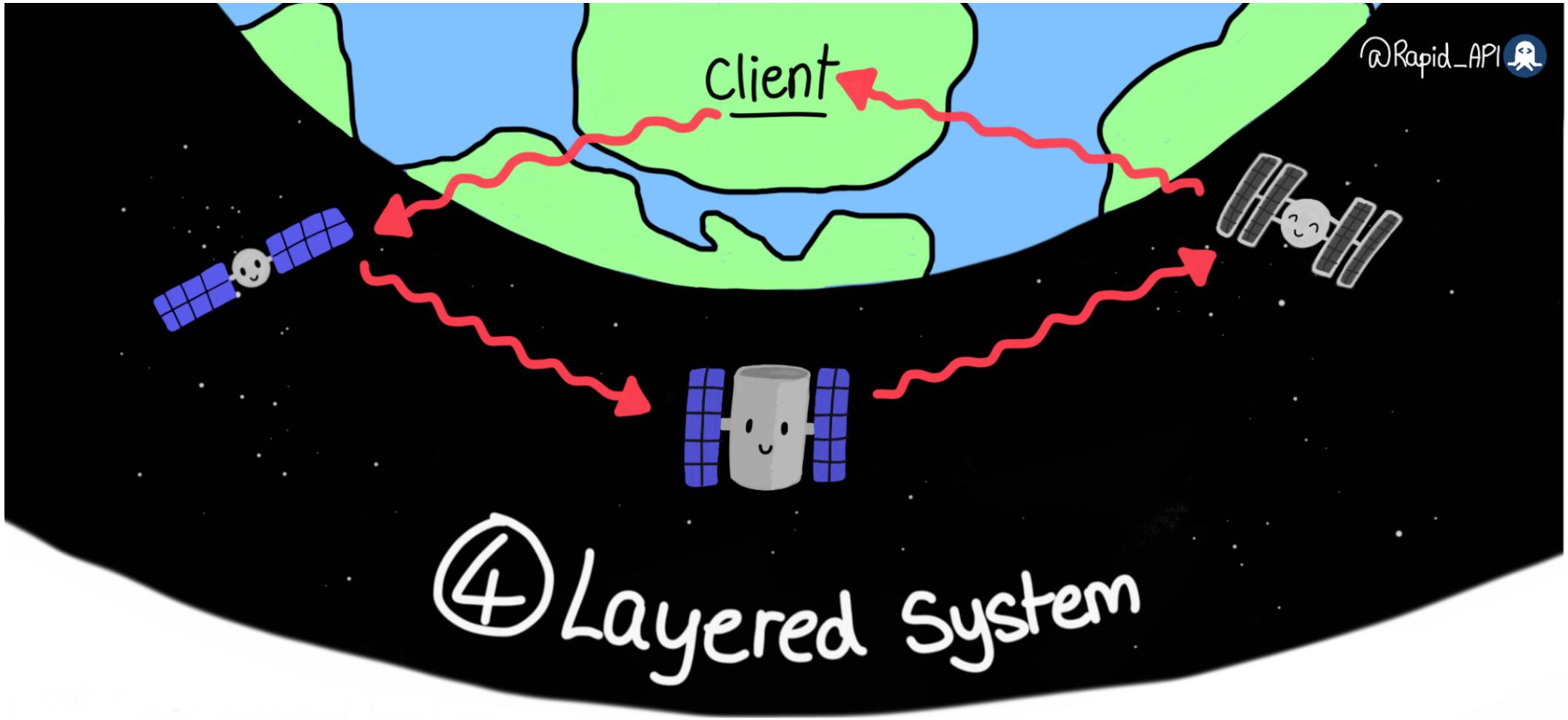
③ Stateless

Stateless means each Server request is dealt with independently, regardless of previous requests.

Stateless transfers allow interactions to be scalable because less server memory is required, and there's no need to retrieve old data.

As software grows, using large amounts of memory isn't a concern.





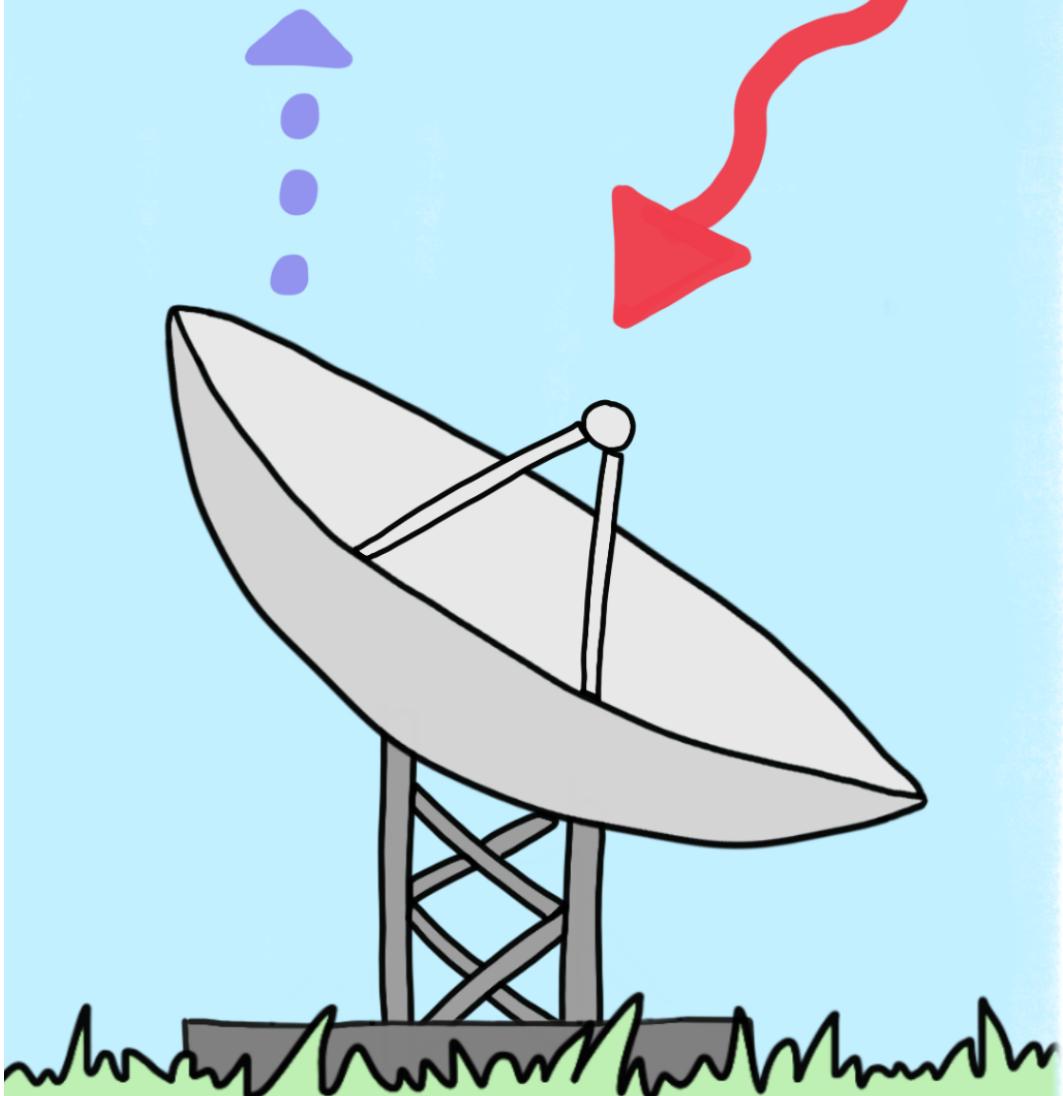
④ Layered System

Other servers (layers) between the client and server carry out other essential functions. The layered system principle requires data to be transferred in the same format.

This means servers can be modified or updated without affecting the API requests and responses.



Client Saves received data to local Storage.



5 Cacheable

Caching allows locally saved data to be loaded quickly when a user returns to a website.

REST APIs can indicate if a resource can be cached.

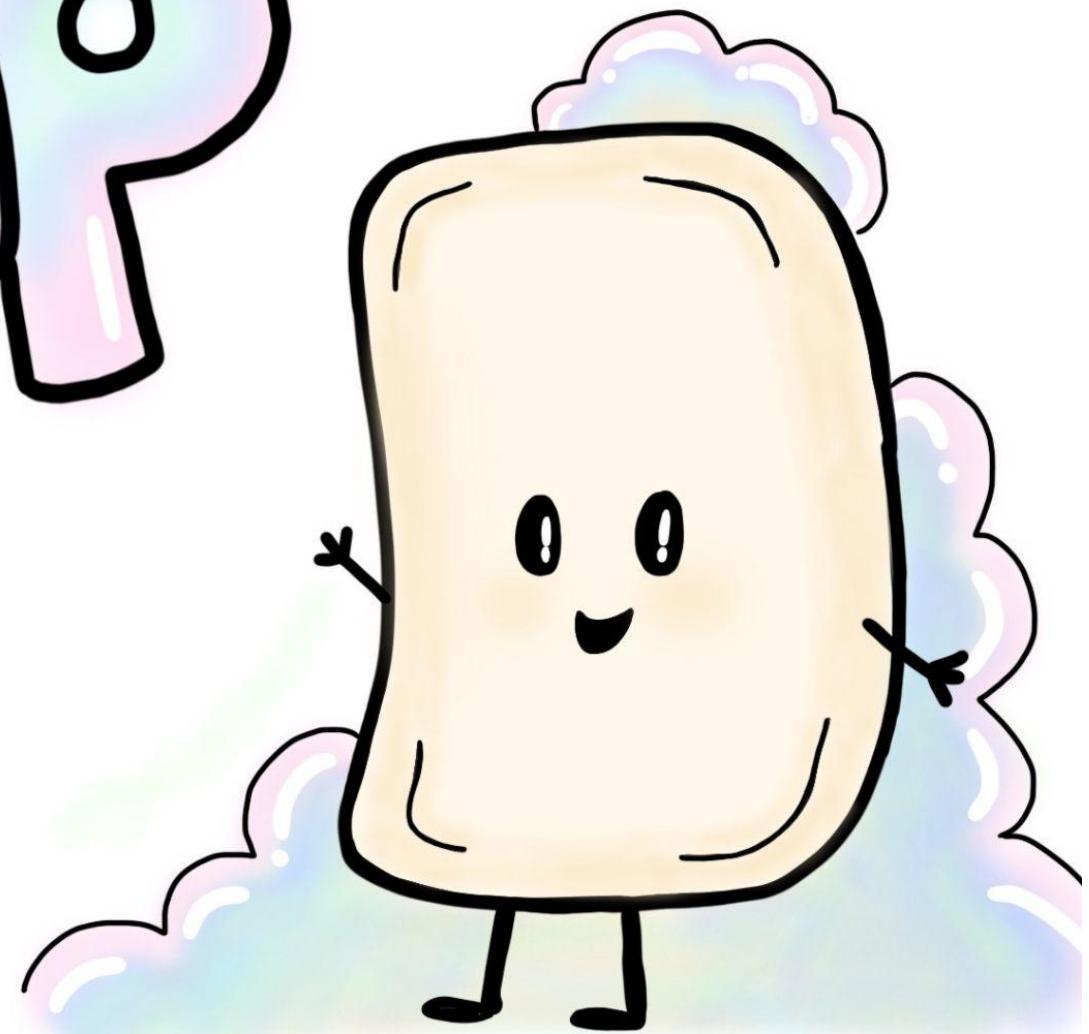
Caching reduces page load time and saves bandwidth.

An introduction to SOAP API

An Introduction to

@Rapid_API 

SOAP API

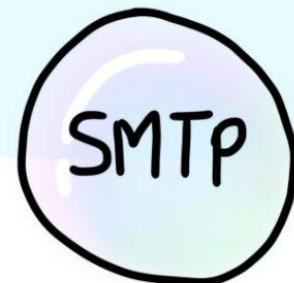


SOAP (Simple Object Access Protocol)
is a web communication protocol designed
by Microsoft in 1998.

SOAP relies only on
the **XML** data format.



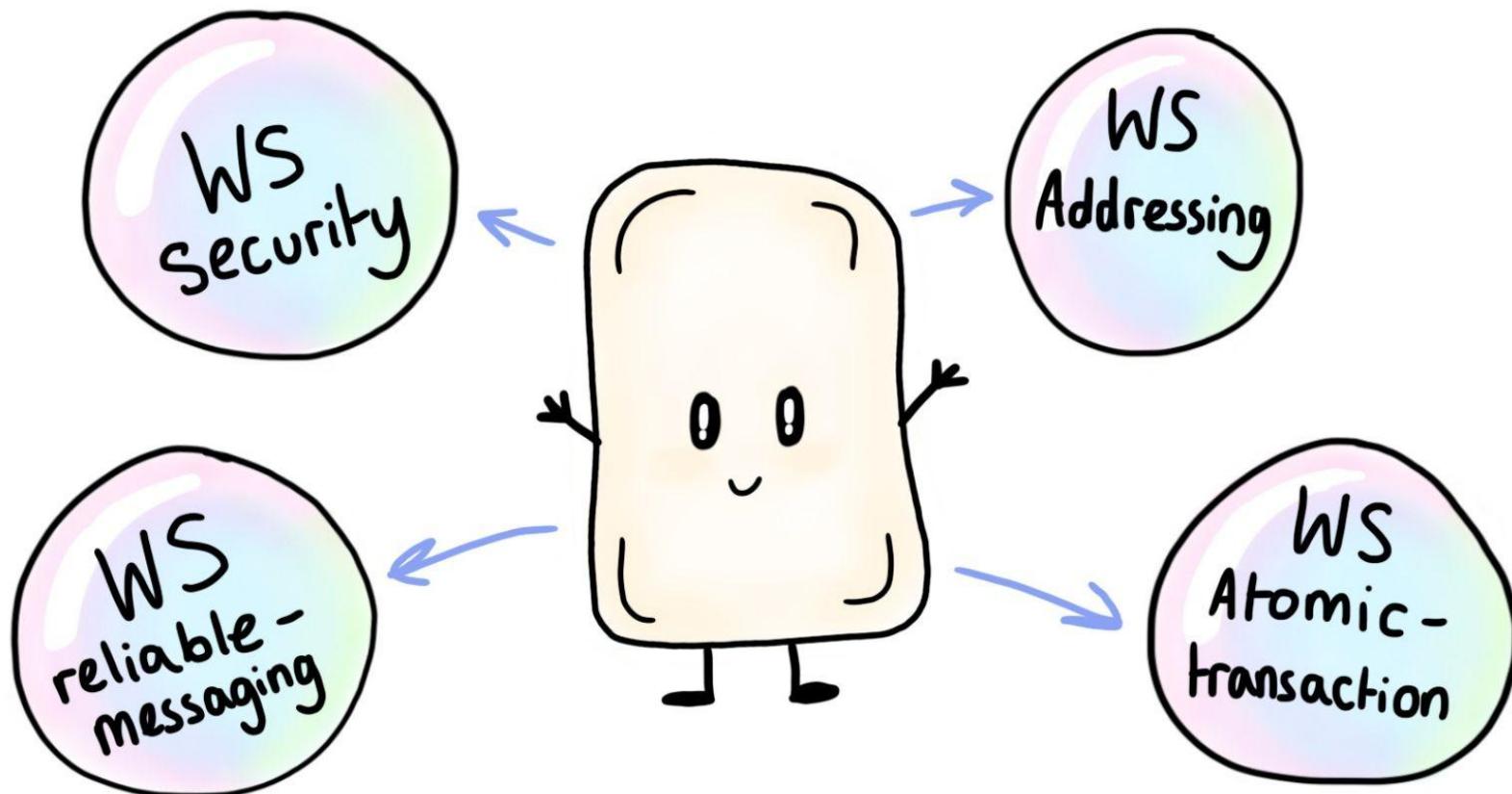
SOAP can use a
range of different protocols.



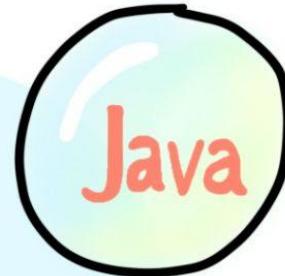
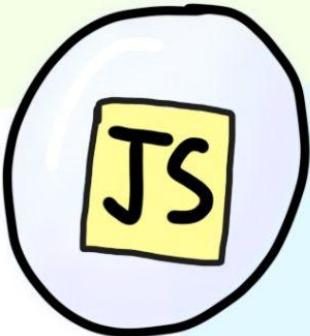
Key features of SOAP

1 Extensible

SOAP was designed to support expansion and added features.



2 Independent



SOAP is language and platform independent thanks to the XML data format.

3

Built-in error handling

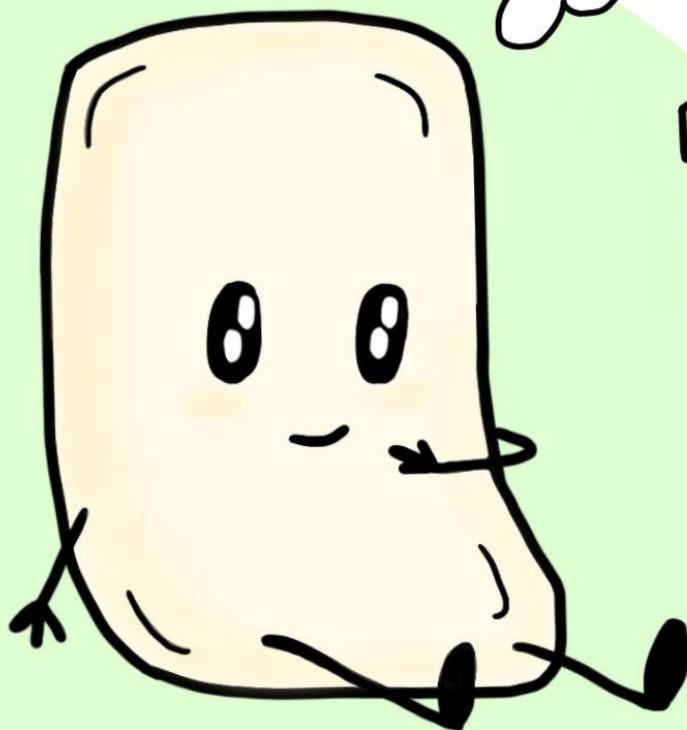
If a request has a problem, it will return error information in its response to help you understand the problem.



Uses of SOAP

API

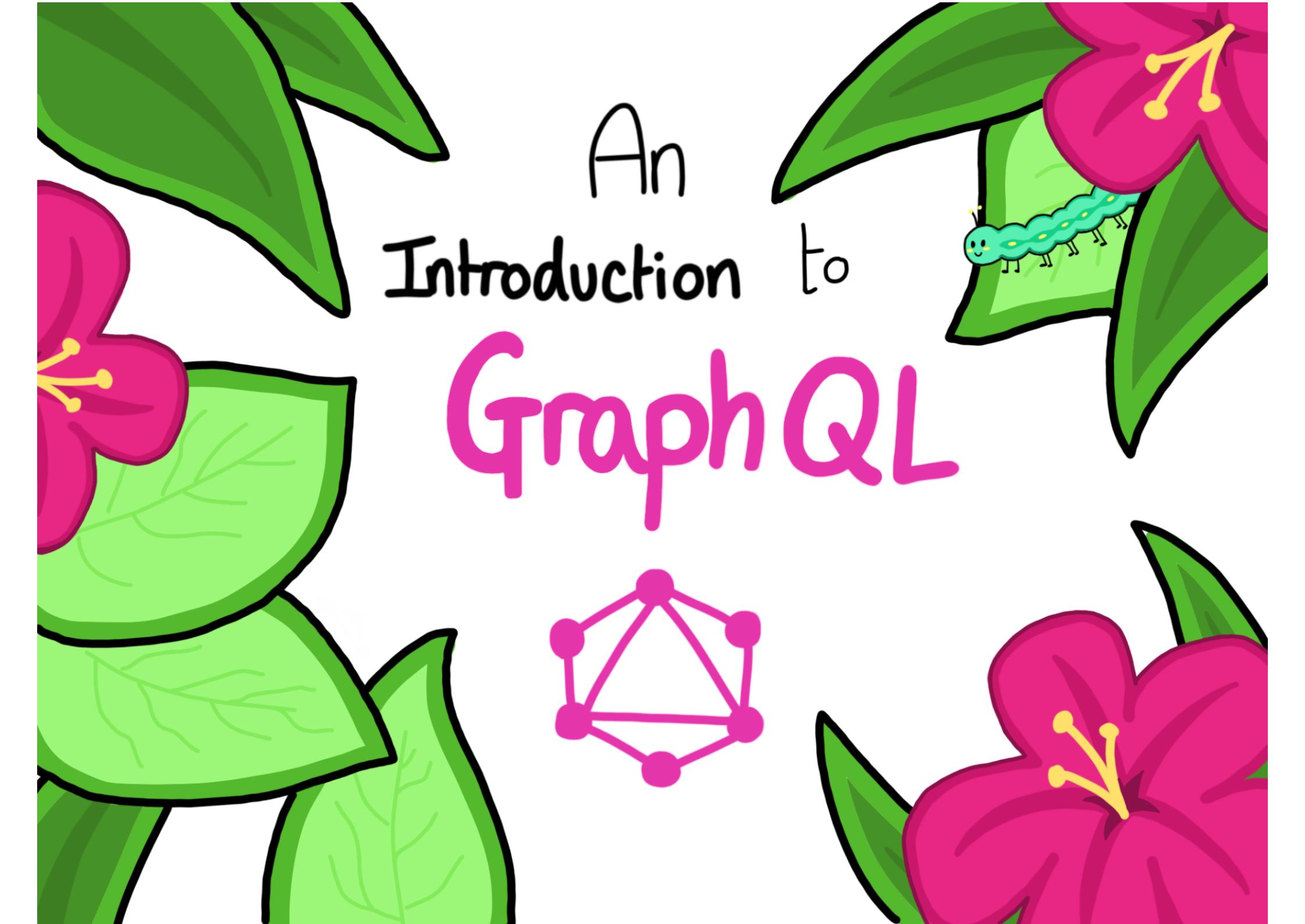
@Rapid_API



Due to its security extensibility, SOAP is used mainly in enterprise-level web services such as financial services, identity management, and complex transactions.

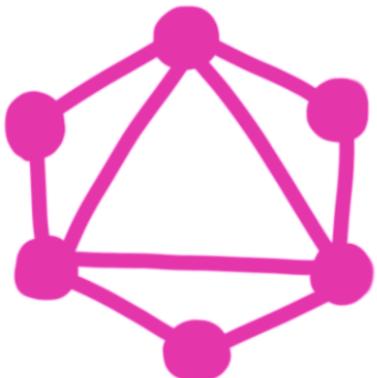


An introduction to GraphQL

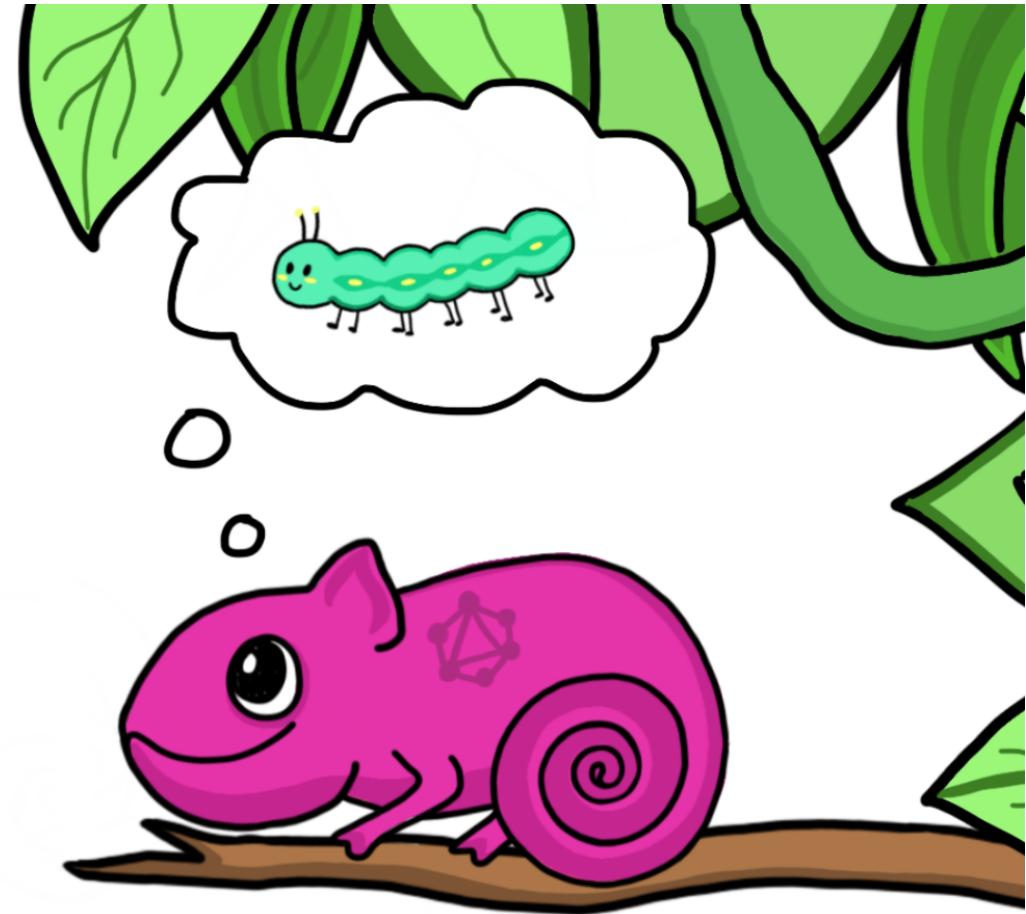


An
Introduction to

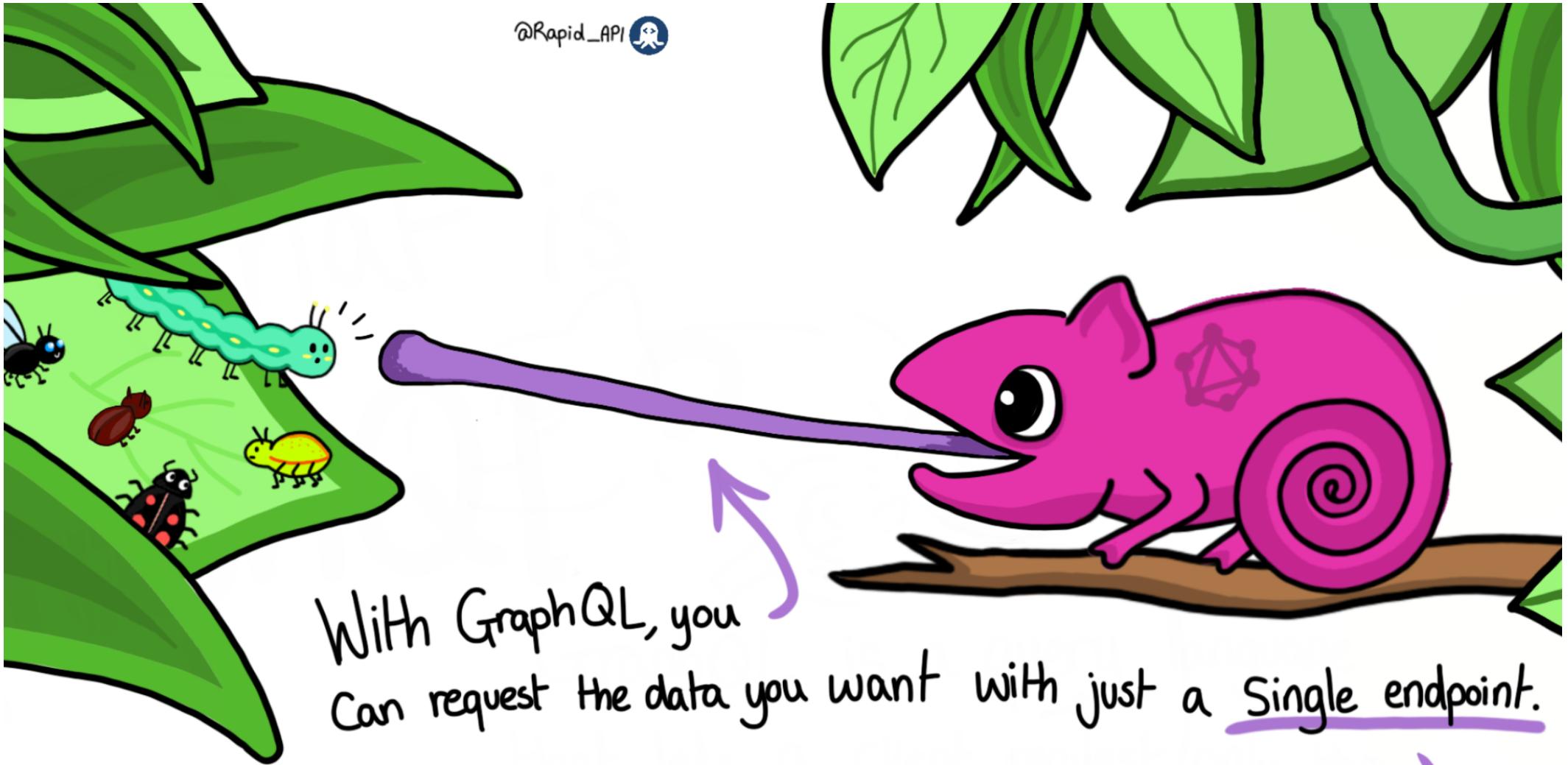
GraphQL



What is GraphQL?



GraphQL is a query language
that lets Clients request only the
exact data they require from a server.



With GraphQL, you
can request the data you want with just a single endpoint.

GraphQL does this
using a strongly typed schema.

The Schema definition language (SDL)

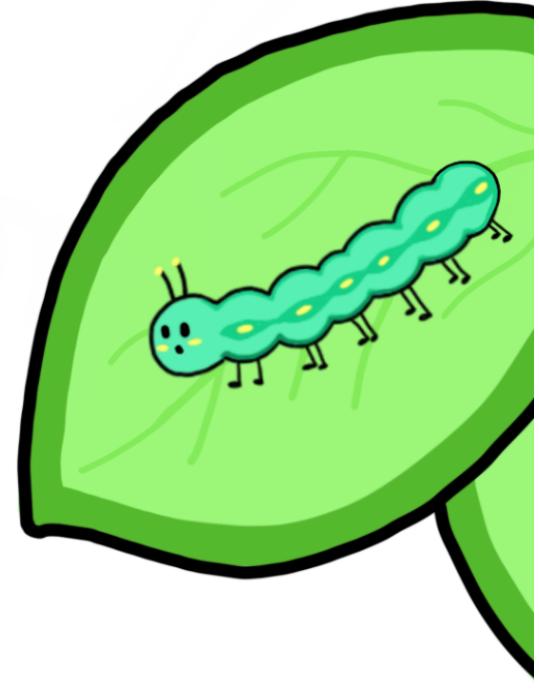
The GraphQL Schema defines the shape of your data and consists of a hierarchy of types and fields.

```
type User {  
    name: String!  
    email: String!  
}
```

- > A typical 'User' schema setup.
- > The exclamation marks (!) mean the field is required.

Once a **Schema** is defined, relationships between the types can be established.

```
type Chameleon {  
    prey: [Caterpillar]  
}  
  
type Caterpillar {  
    predator: Chameleon  
}
```



There are two Operations:

Queries

Queries are used to fetch data.

```
{  
  bugs {  
    Species  
  }  
}
```

This fetches all the Species from a bug list API.

Mutations

Mutations are used to Create, update, and delete data.

```
{  
  createBug (  
    Species: "stag beetle"  
  ){  
    Species  
  }  
}
```

This adds a new bug species to the bug list.

The benefits of GraphQL

Fetches data
with a Single API
call.



Tailored
requests.

Reduces bandwidth.

No over-fetching or
under-fetching of data.

Suitable for smaller
networks and microservices.

API growth without
versioning.

An introduction to Webhooks



An Introduction To



Webhooks



What are Webhooks?

Webhooks allow third-party services to send data to an application in real-time.

Webhooks are another way apps communicate and exchange data, just like APIs, but a little different.

There is no request and response system, just an event and an automatic response.

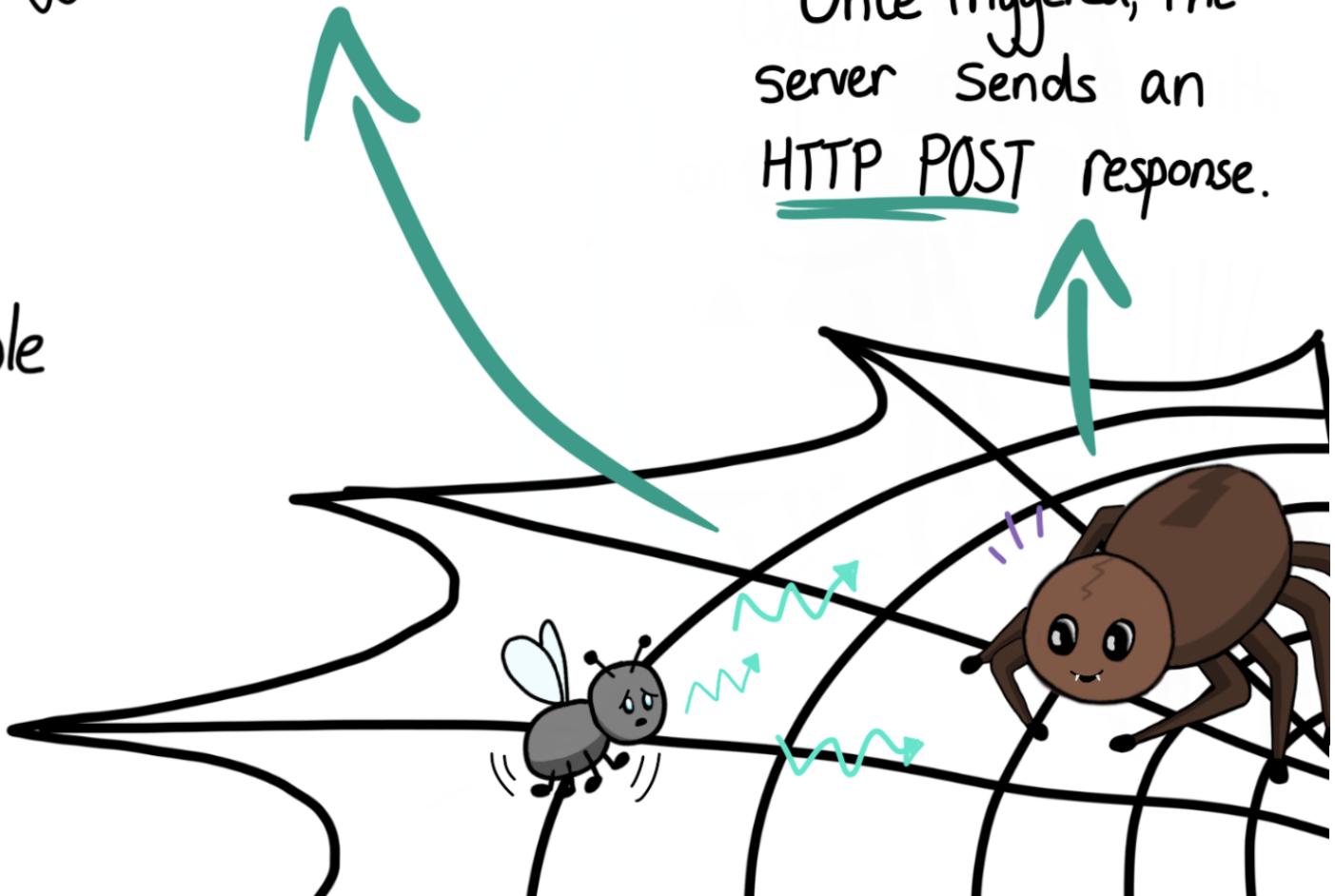


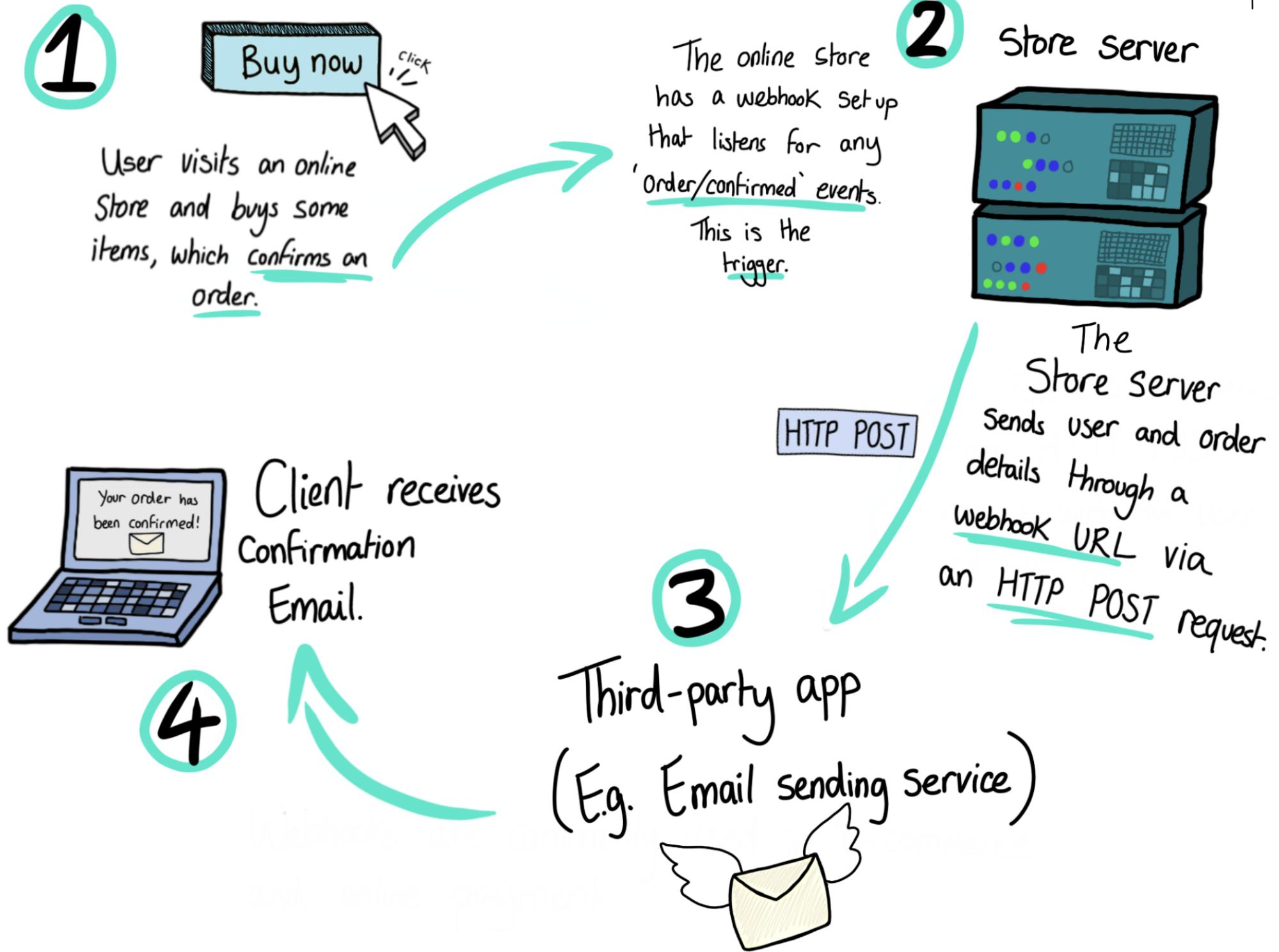
How do Webhooks Work?

Take a look at
a real-world example
of how they work.

A set event or action
triggers Webhooks.

Once triggered, the
server sends an
HTTP POST response.







Webhooks simplify and streamline the data transfer process. They are particularly convenient for real time notifications and updates.

Why use Webhooks?



Ecommerce

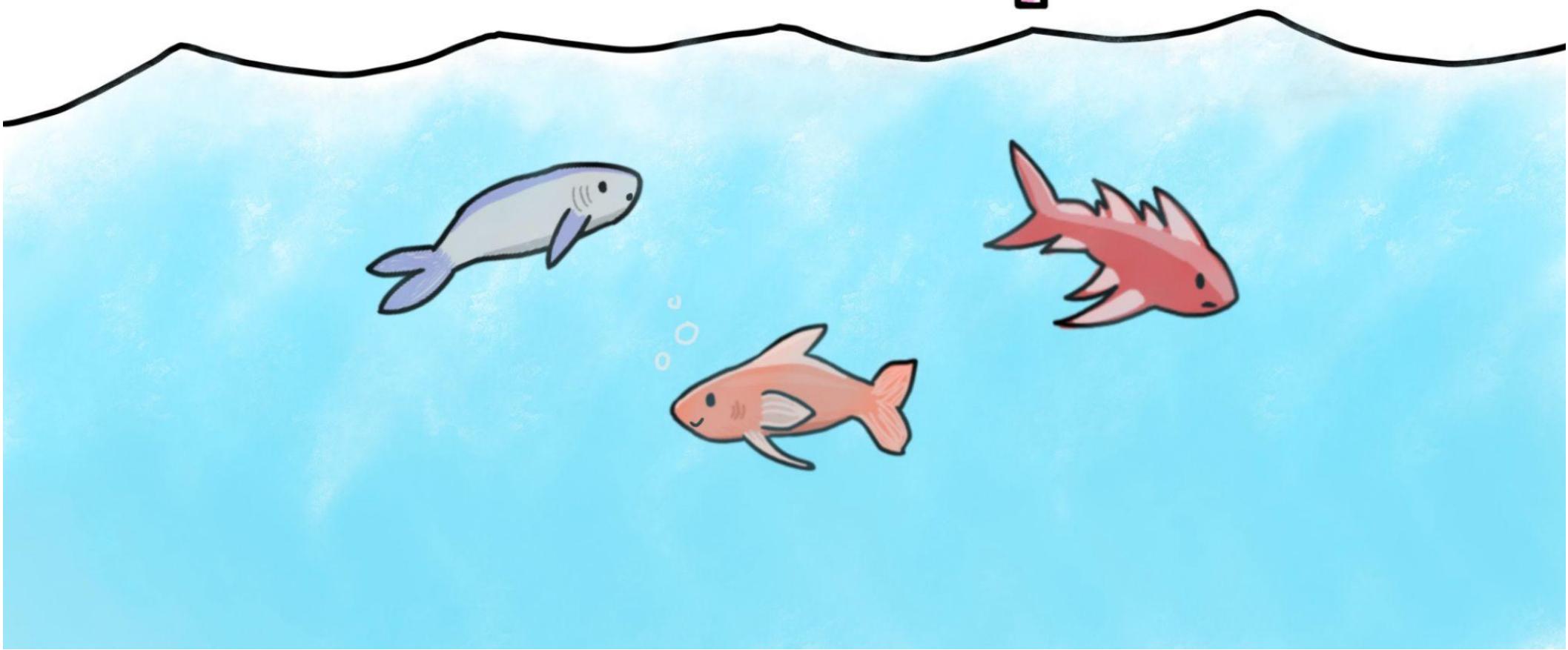
Payment
Software



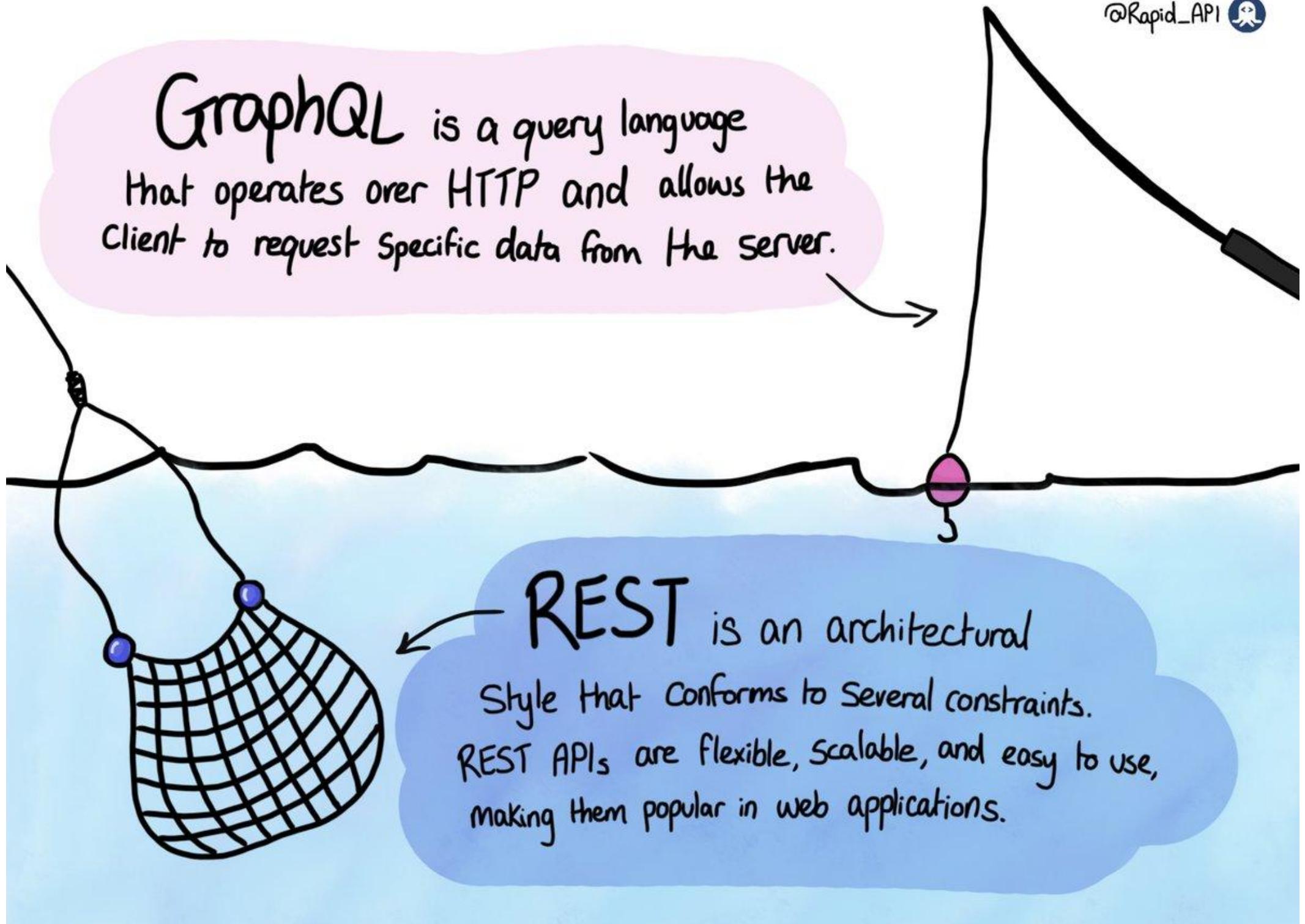
Social media

REST vs GraphQL

What's the difference between
REST and **GraphQL**?



GraphQL is a query language
that operates over HTTP and allows the
Client to request Specific data from the server.

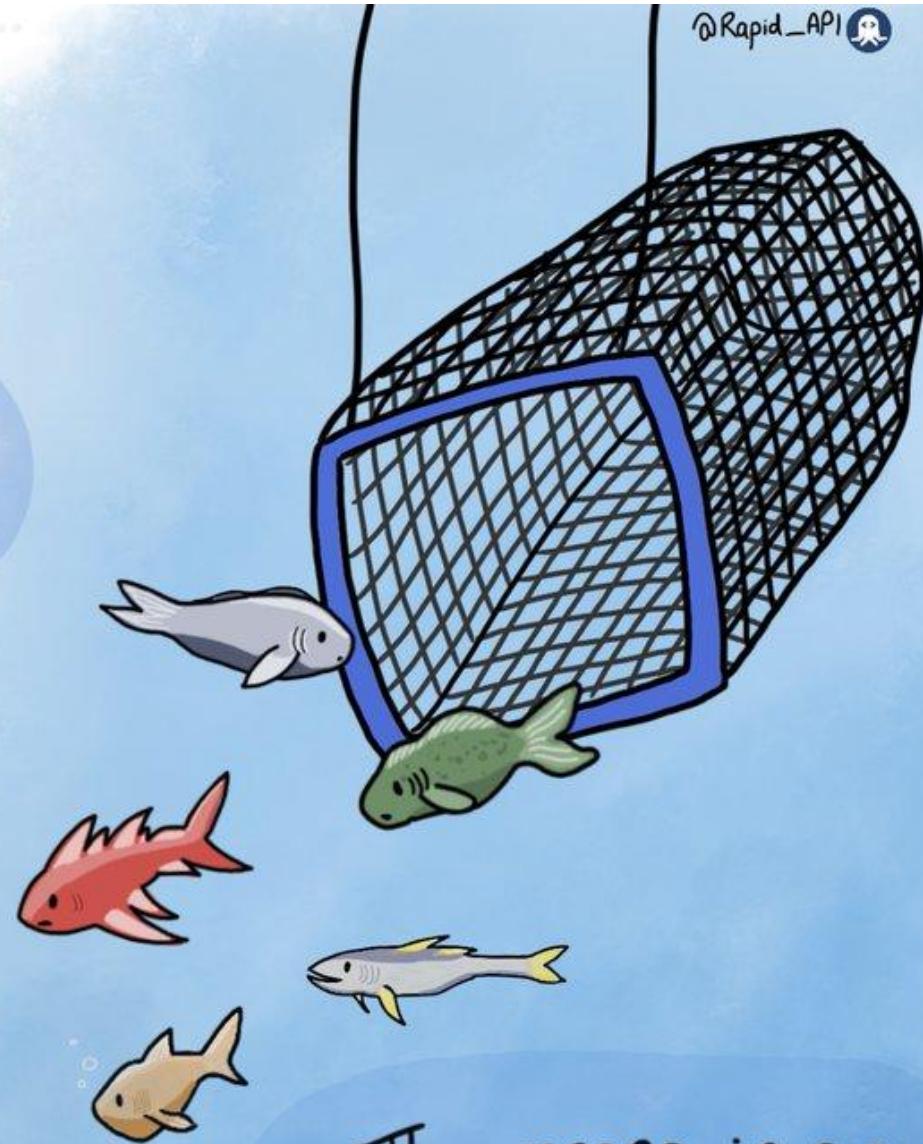


REST is an architectural
Style that Conforms to Several constraints.
REST APIs are flexible, Scalable, and easy to use,
making them popular in web applications.

REST APIs

Fetch data in a way that returns the whole data set.

If you want specific data from two objects, you'll need to make two REST API requests.



This means in some cases you might be over-fetching data, which means returning some data you won't need.

GraphQL allows

the Client to specify the exact data returned thanks to Schema definition language (SDL).

SDL is simply the Syntax of writing schemas in GraphQL.

When querying in GraphQL, we can access specific data using just one endpoint.

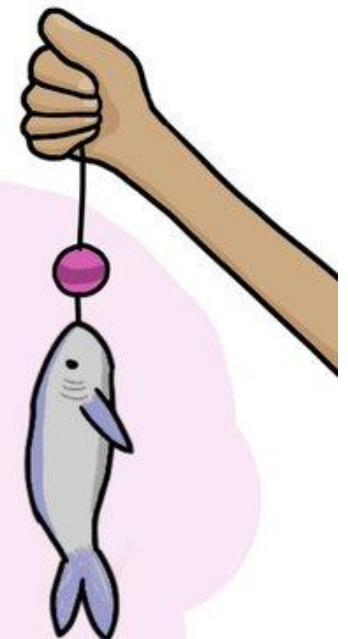


REST APIs are:

- Simple to use and set up
- Client and server independent
- Flexible, and scalable

GraphQL is:

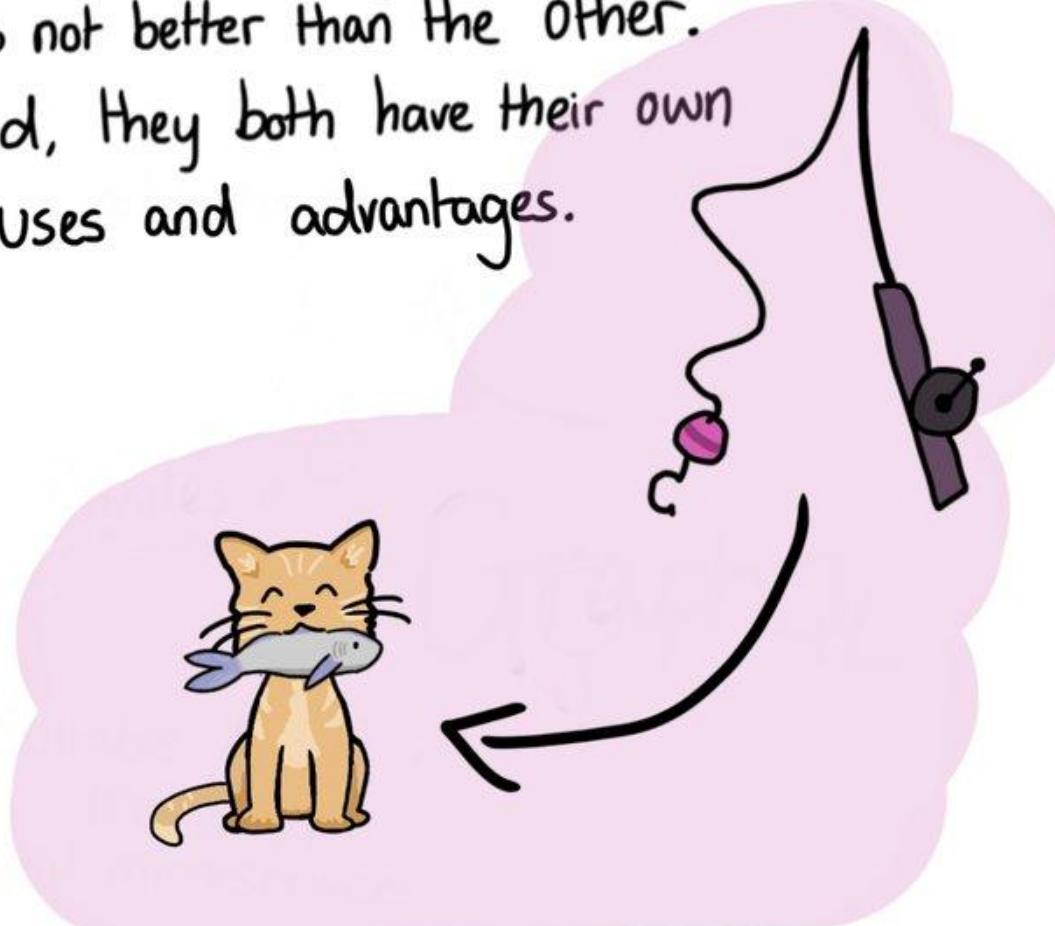
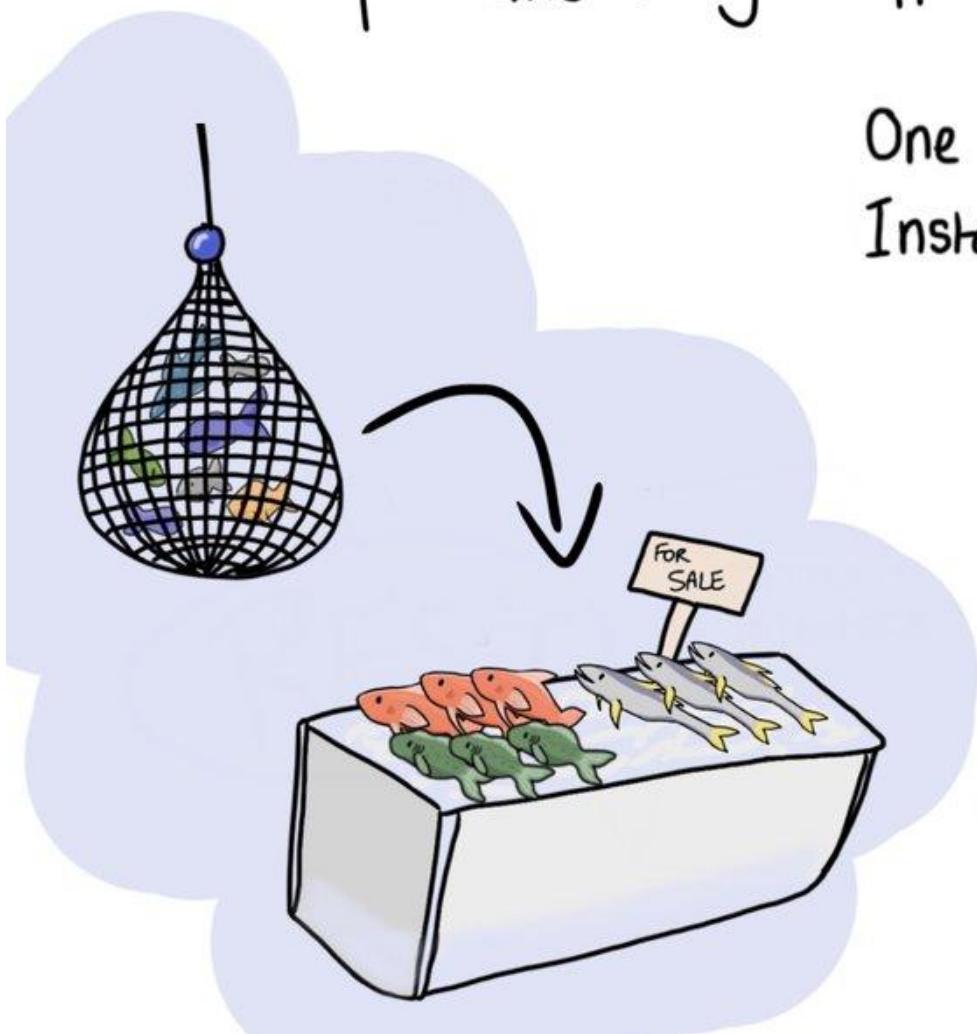
- Centered around one API endpoint
- Tailored to your data requirements
- Requires less bandwidth



Which Should you Use?

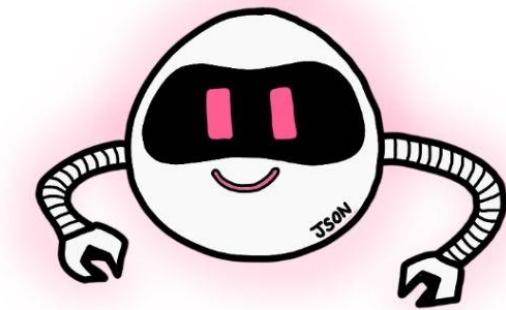
It depends entirely on the requirements of your application.

One is not better than the other.
Instead, they both have their own uses and advantages.

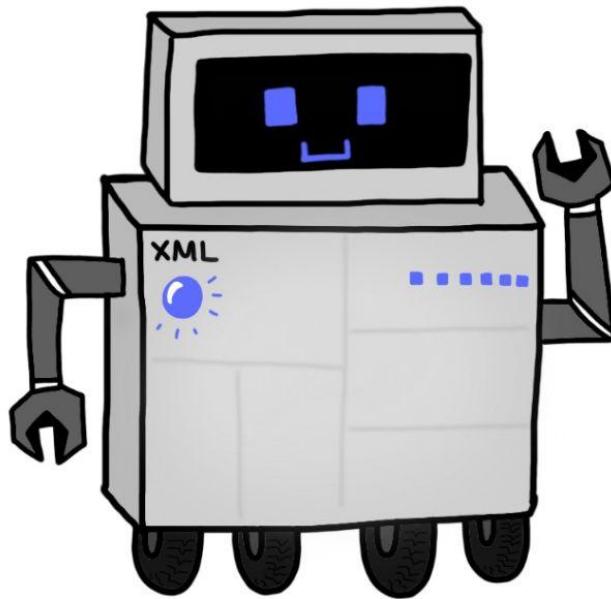


JSON vs XML

JSON



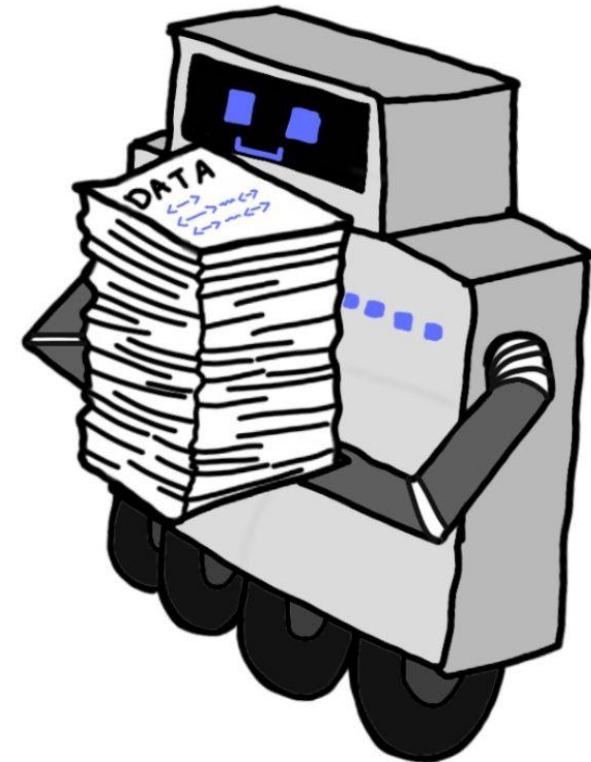
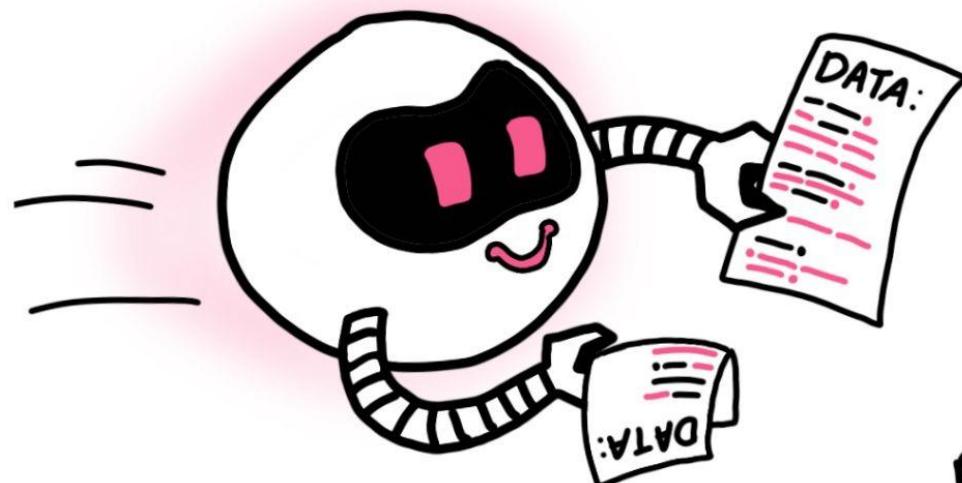
VS



XML

Both JSON and XML are data formats

used to send and receive data
from web servers.



Both play an important role in organizing data into a readable format in many different languages and APIs.

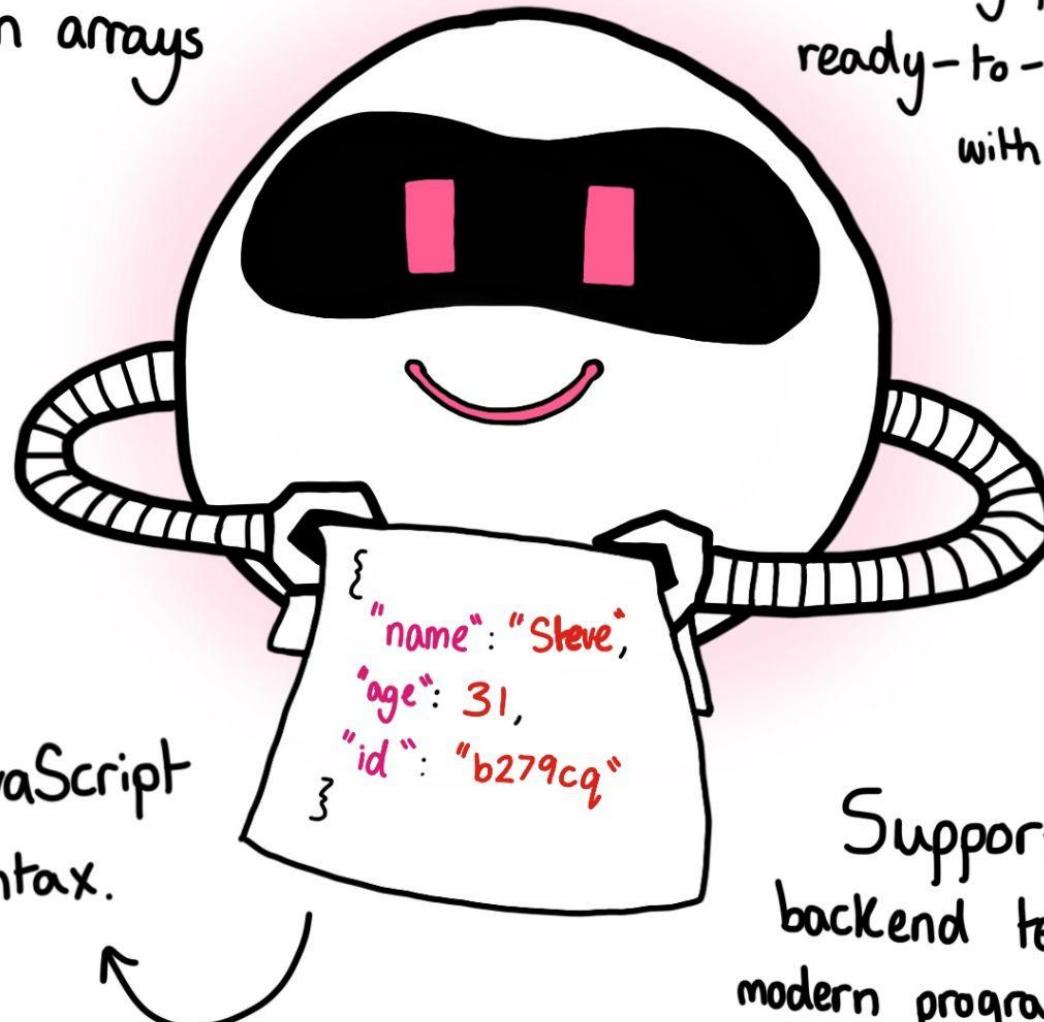
JSON

: "JavaScript Object Notation"

Stores data in arrays
for easier data
transfers.

Easily parsed into a
ready-to-use JavaScript object,
with no library needed.

Based on JavaScript
object literal syntax.



Easy to read
and write.

Supported by most
backend technologies and
modern programming languages.

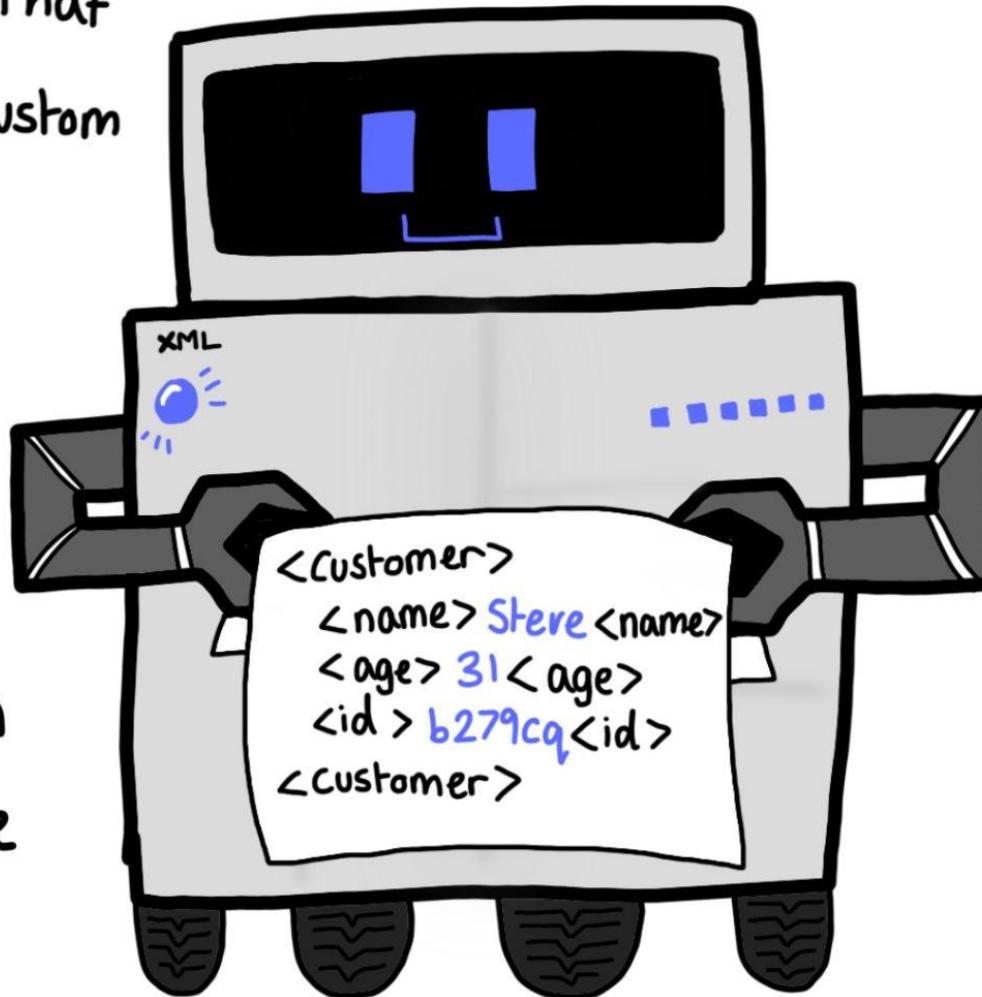
XML :

"Extensive Markup Language"

Markup language that allows creating custom user defined tags.

Manages data in a 'tree' structure hierarchy.

Complex data structure that must be parsed.



Similarities and differences

JSON and XML

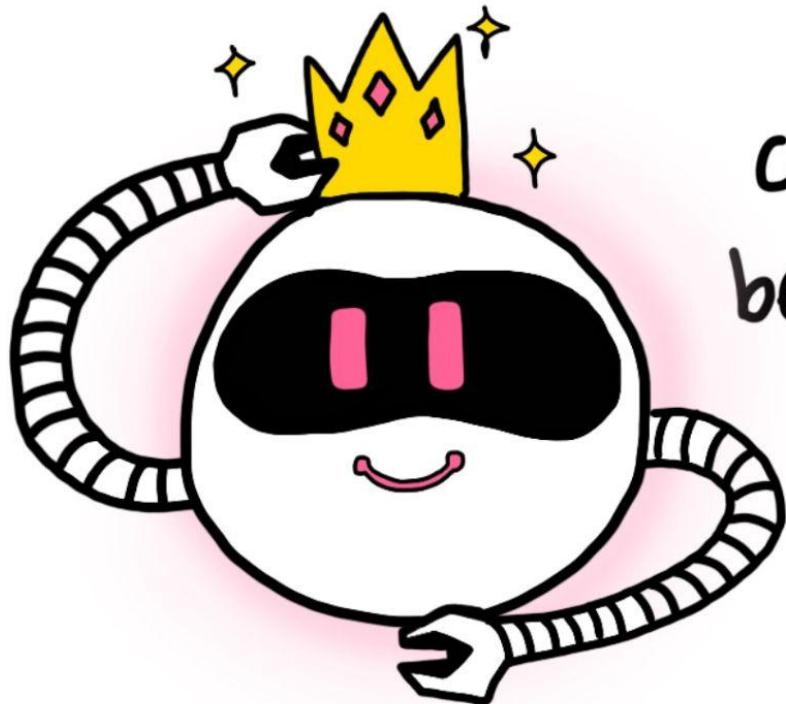
are similar because:

- Self-describing
(human readable)
- Parsed and used by many different programming languages
- Hierarchical

JSON and XML

are different because:

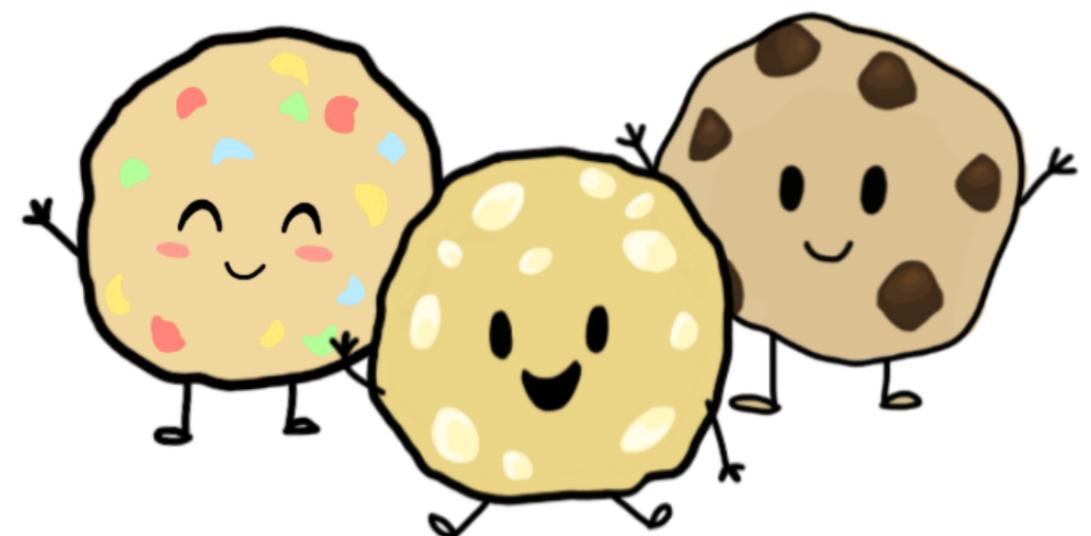
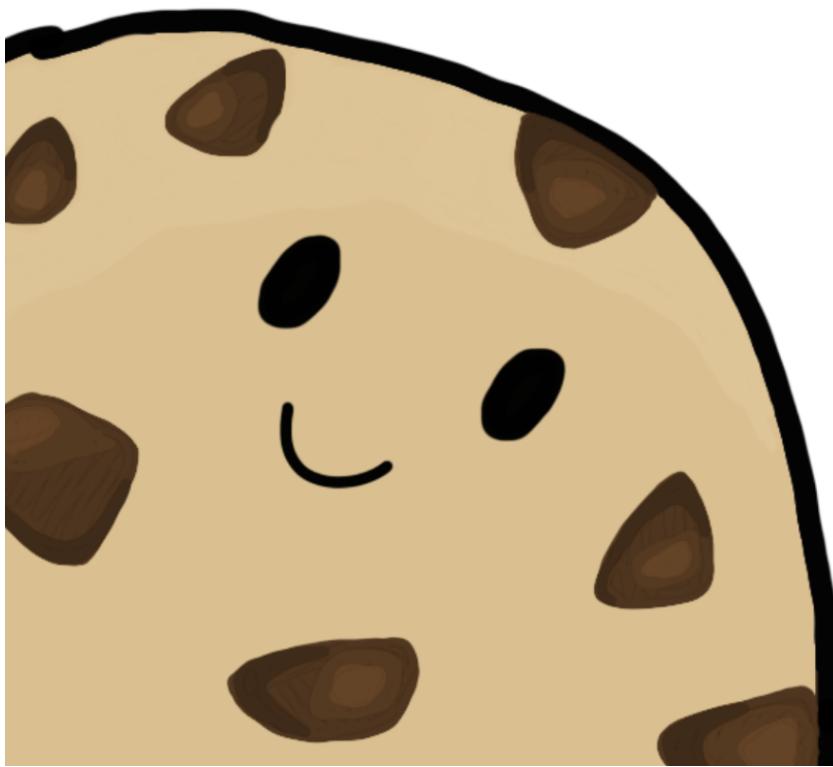
- JSON object has a type, XML is typeless
- XML has display capabilities unlike JSON
- JSON is less secure than XML
- XML is much more complex and slow to parse



Generally, **JSON** is considered better than **XML** because it is easier to parse into a ready-to-use **JavaScript** object.

What are HTTP Cookies

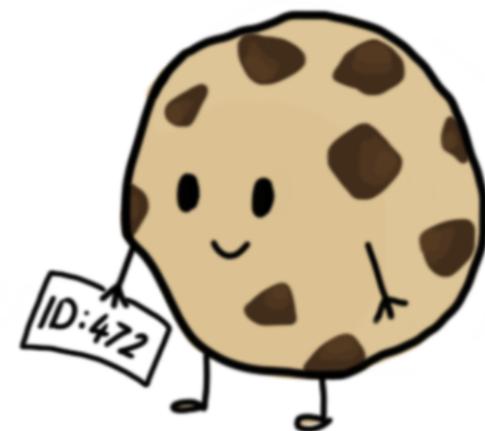
What are **HTTP Cookies?**





Cookies are small pieces of data sent by a server and stored in the client's browser.

The general purpose of cookies is to identify each user so that websites can adapt their content accordingly.

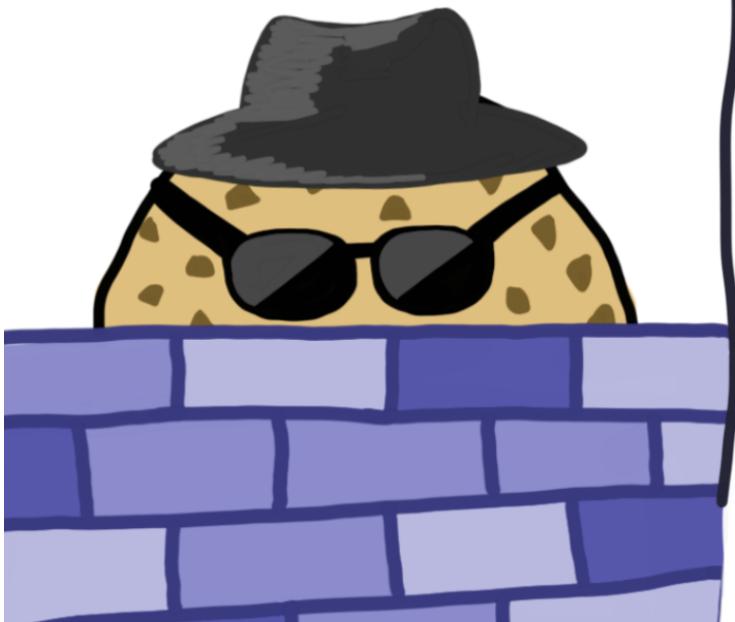


Although transferred via HTTP protocol which is stateless, cookies allow us to store meaningful state that benefits the functioning of the web.

Cookies have three Primary Purposes:

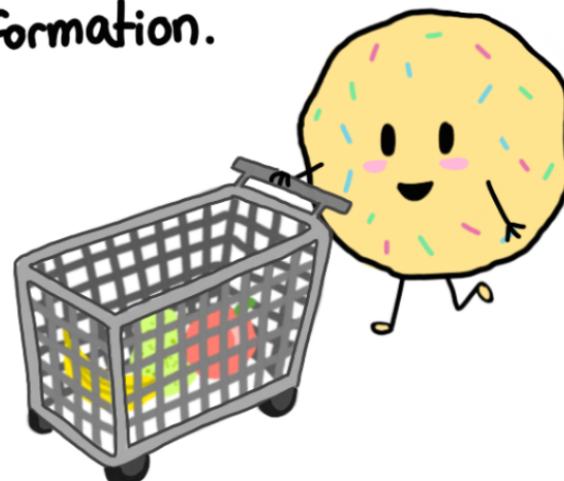
Tracking

Tracking builds statistics about the user, and this data can be used for ad personalization.



Session management

Cookies store data from sites so that when you return your changes are kept. For example - items you put into a shopping cart, or login information.

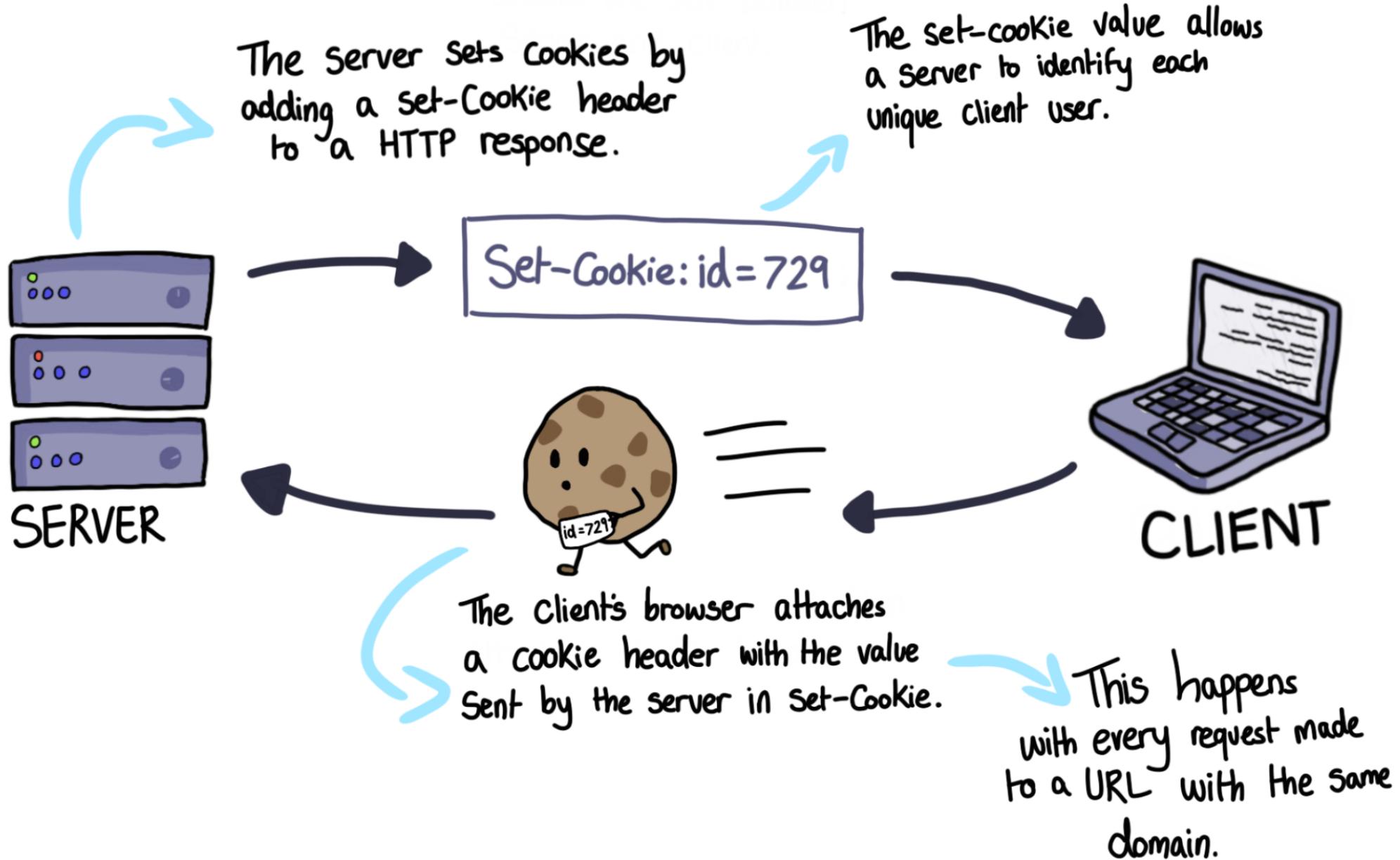


Personalization

Cookies can also retain data on user preferences, such as language, themes, location, and layout.



How are Cookies sent?



Cookie Lifetime

Cookies have different lifetimes depending on whether they are:

Permanent Cookies



This type uses the Expires attribute to set a specified date that the cookie will be deleted.

Set-Cookie:

`id=8125;`

`Expires=Sat, 2 Jul 2022, 08:00:00 GMT;`

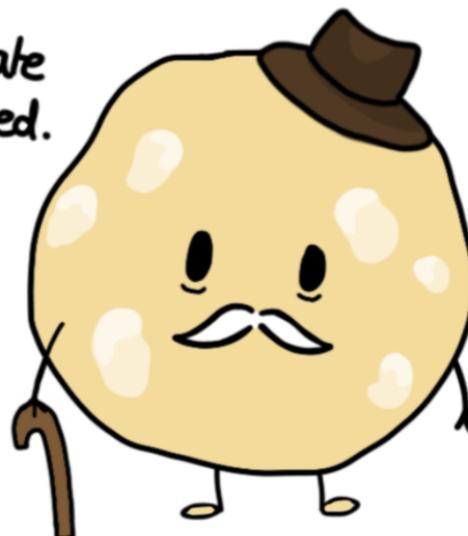
You may also use the Max-Age attribute to do this.

or

Session Cookies



Session Cookies are deleted when the current browser session ends. If the browser restores when restarting, some cookies can remain permanently.



Sync and Async programming

Synchronous and Asynchronous Programming

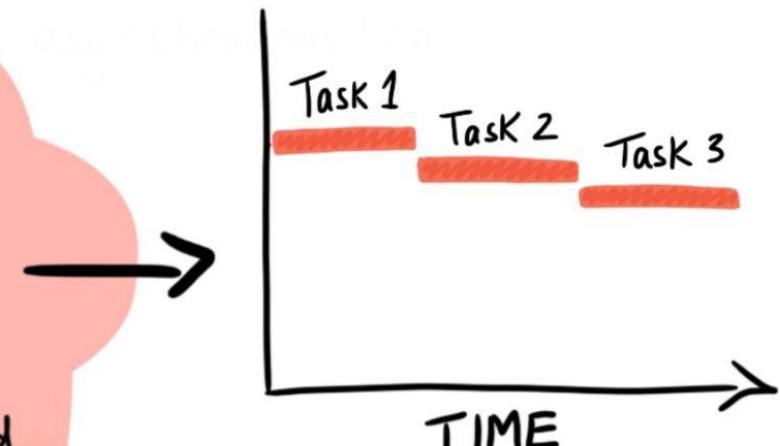


Synchronicity refers to the way code is executed.

Synchronous

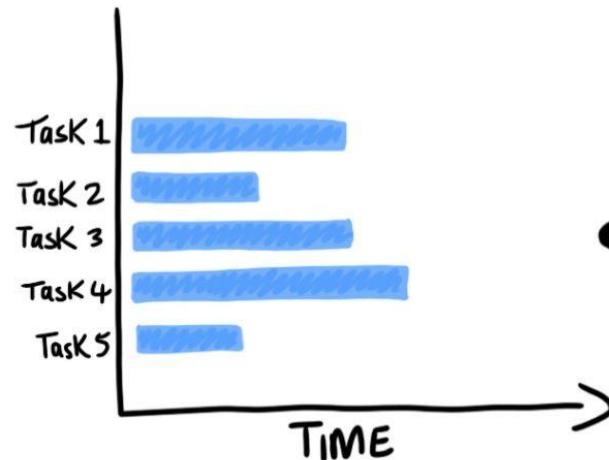
Synchronous calls are blocking.

This means all other code execution
is halted until the call is returned.



Asynchronous

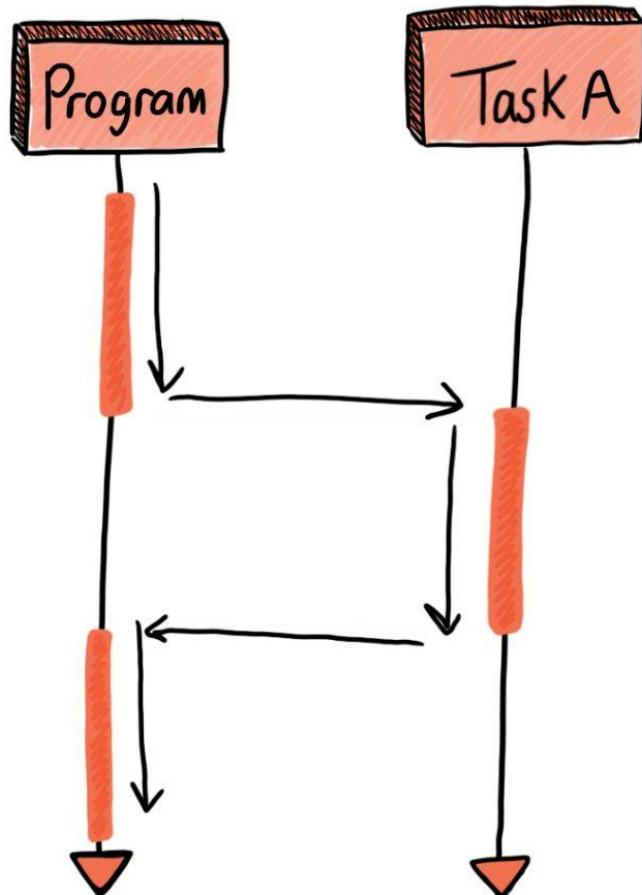
Asynchronous calls are carried out whilst
the rest of the code continues to execute,
no matter how long they take.





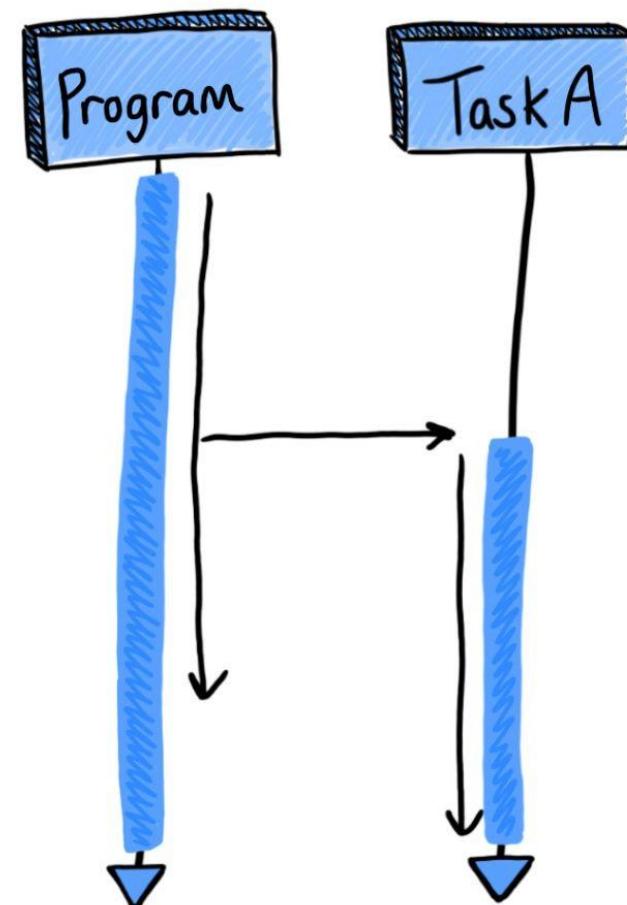
Synchronous

Tasks performed one at a time. When one ends, the next one begins.



Asynchronous

Independent tasks carried out in parallel.

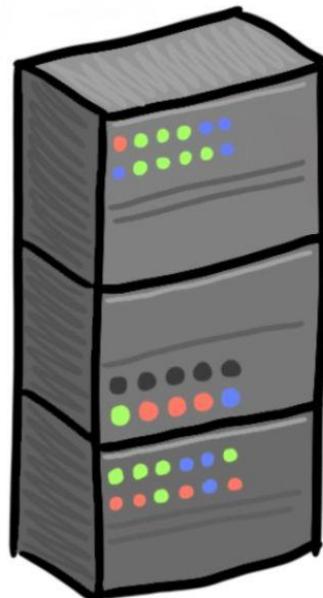


Asynchronous tasks mean the browser can maintain functionality rather than get held up waiting on a request.



Async programming can lead to greater performance and user experience.

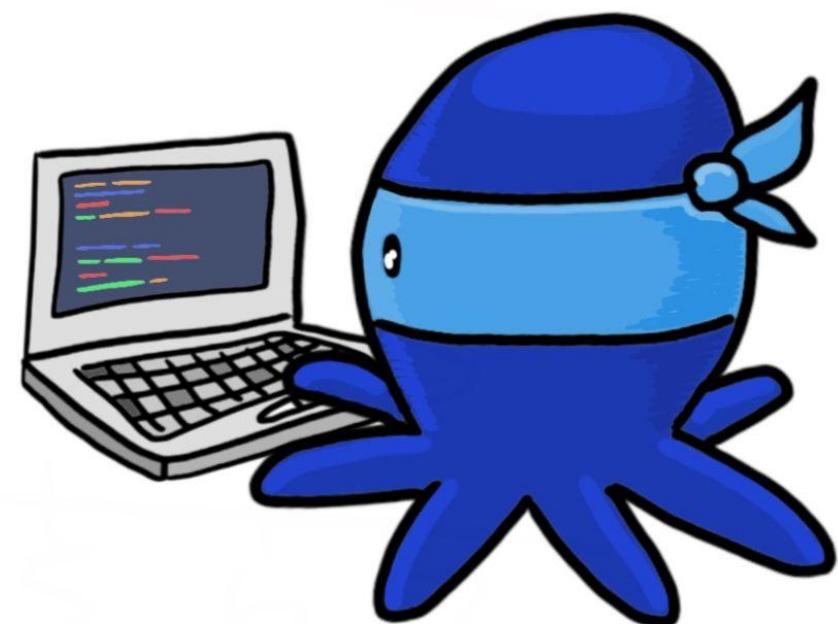
It isn't always
necessary to choose async over
synchronous programming.



Synchronous programming is great
for programs that react to
environmental input.

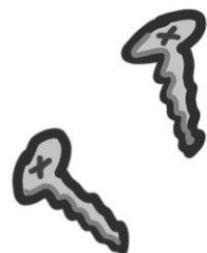
Tasks that may take
longer, such as a database query, are
better handled asynchronously.

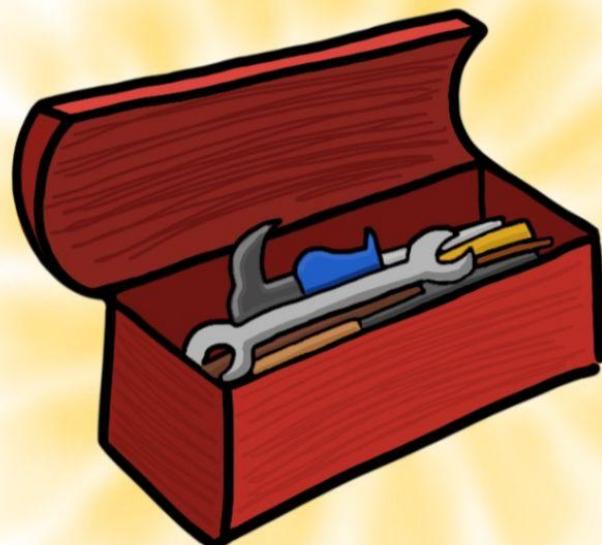
Ultimately it depends
on the requirements of
your program!



SDK vs API

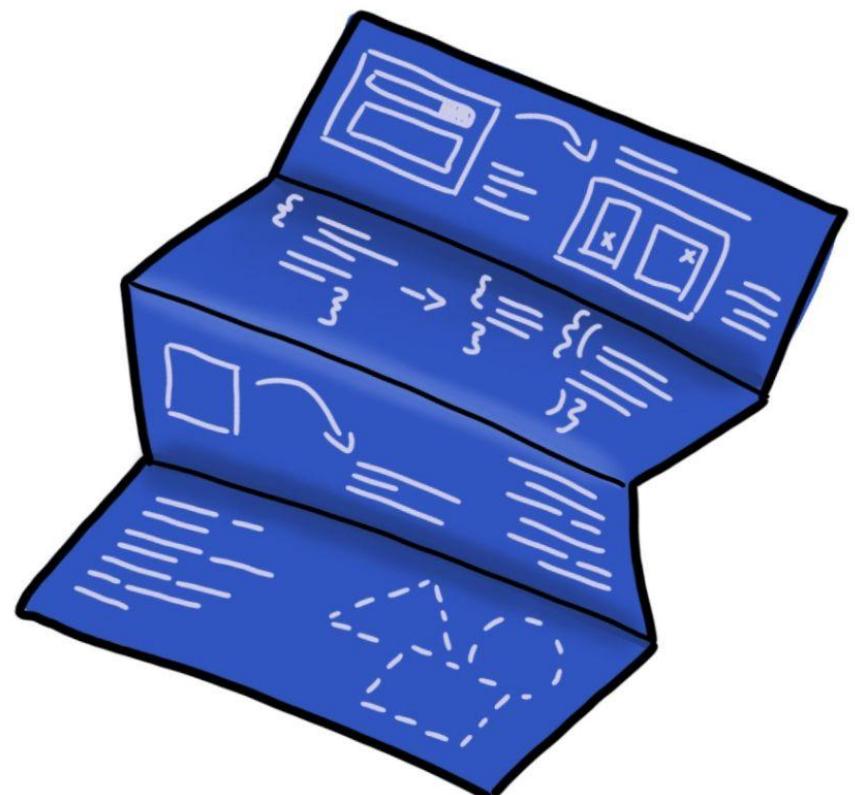
What's the difference
between
an **SDK** and
an **API**?





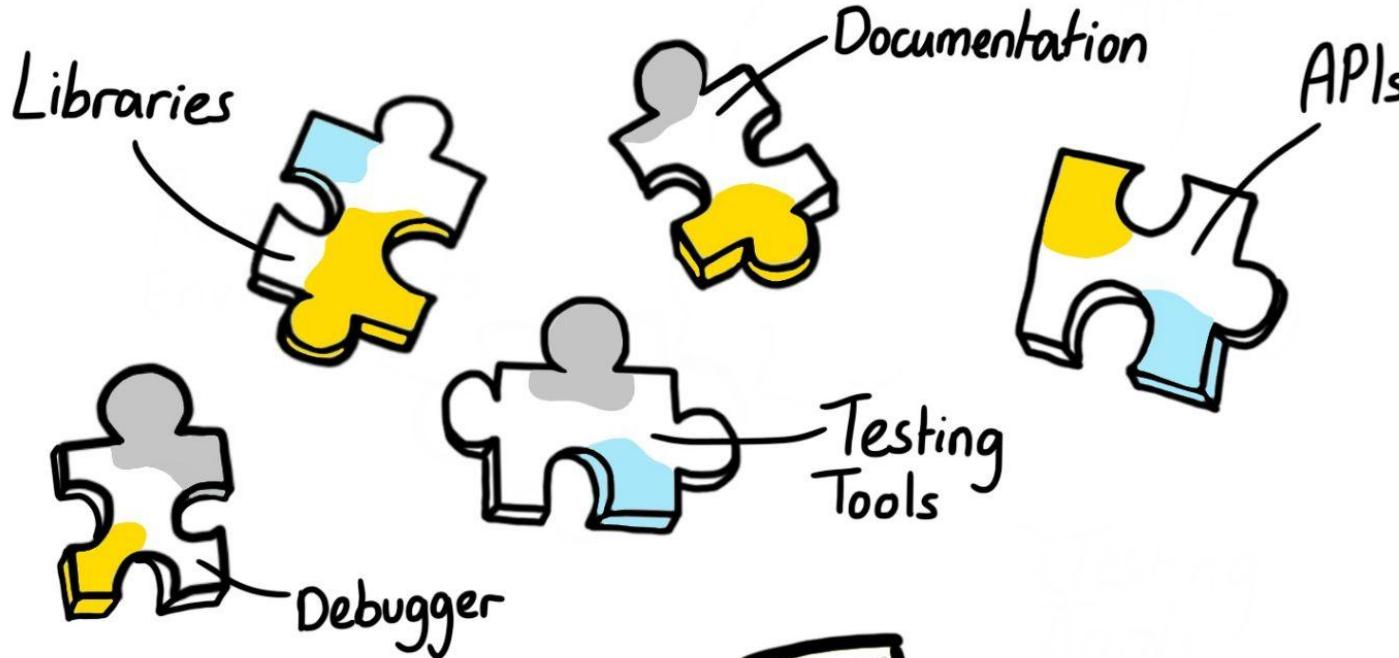
SDK Stands For
Software development Kit.

SDKs are a set of
ready-to-use tools that
allow developers to build
apps for specific platforms.

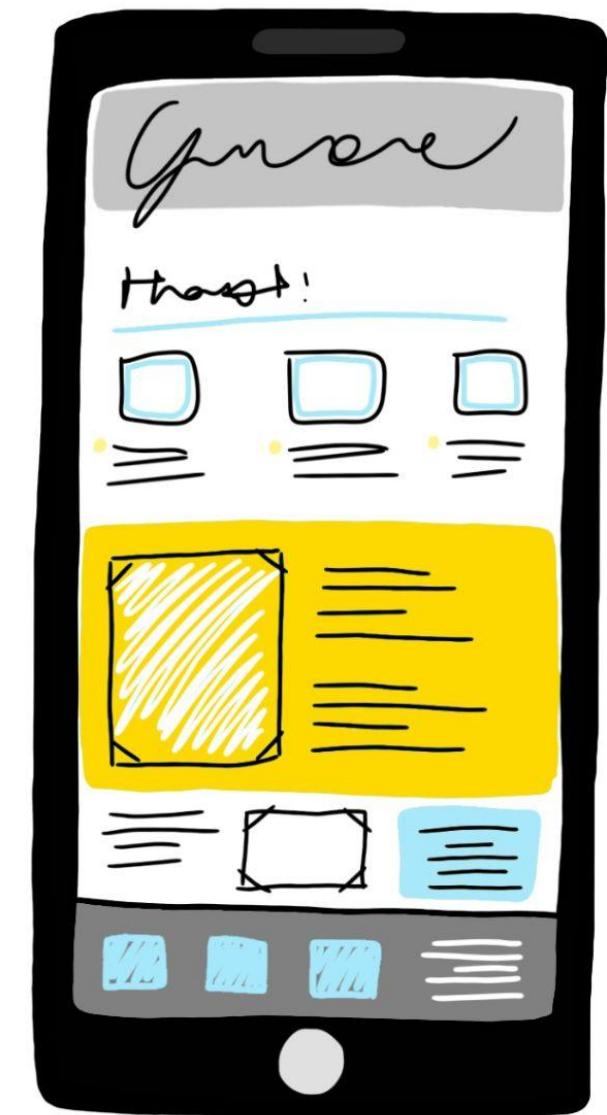


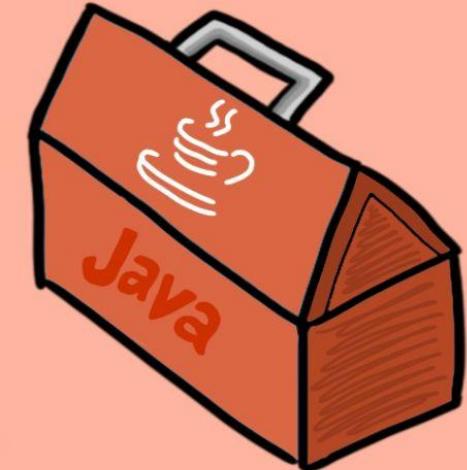
APIs are often another
'tool' included in an SDK.





SDKs can contain several components that facilitate the app building process. Most will always include a compiler, debugger, and APIs.





There are many SDKs available online. Some examples are Android, Java, and Facebook.

Microservices vs API



MicroServices

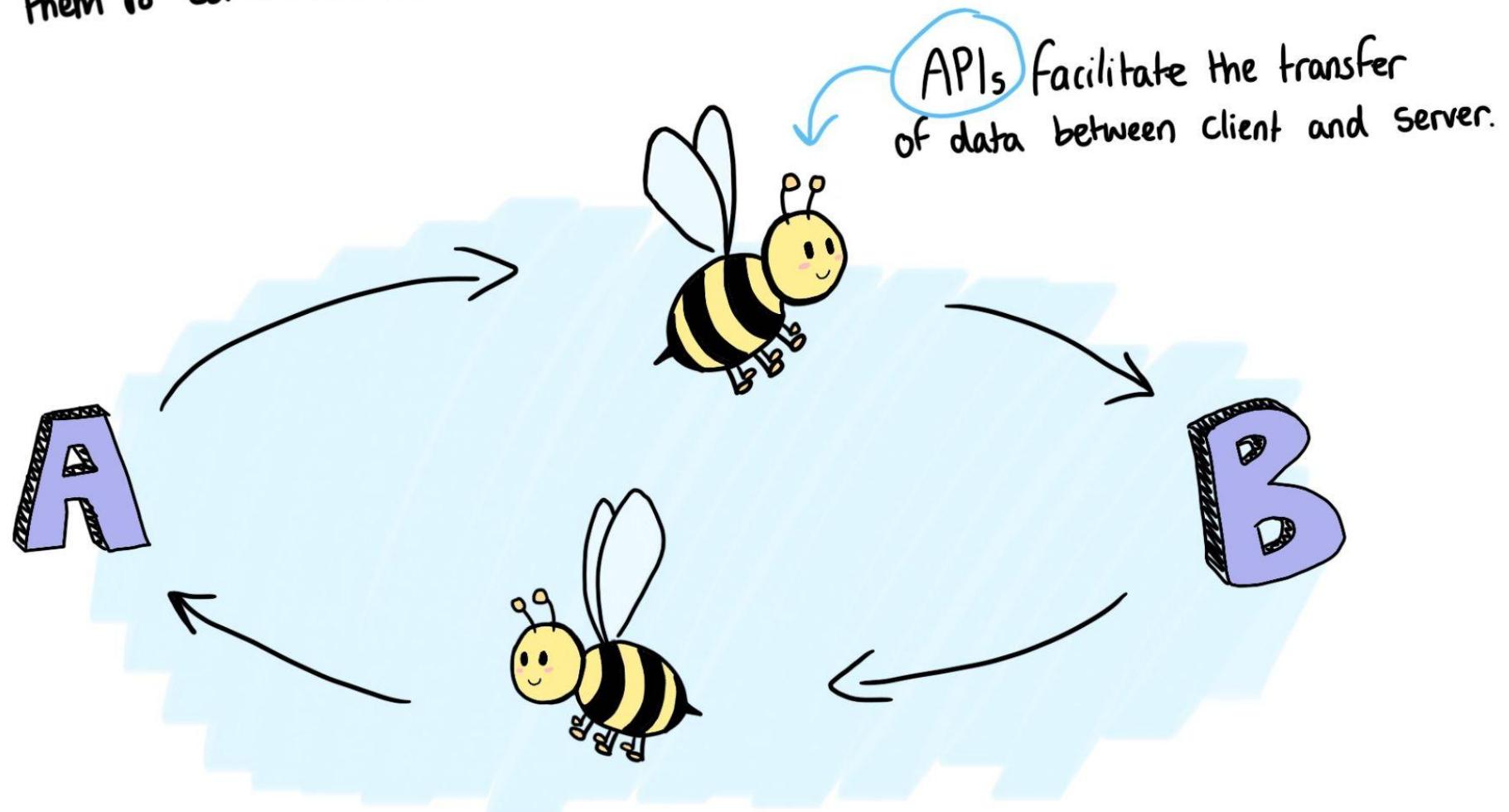
VS



APIs



Firstly, an API (Application Programming Interface) is the intermediary between two programs that enables them to communicate.



On the other hand, a microservice is an architectural style.

Microservices

focus on maintaining several independent services that work collectively to create an application.

Each Service is independently deployable, and has a purpose of its own.

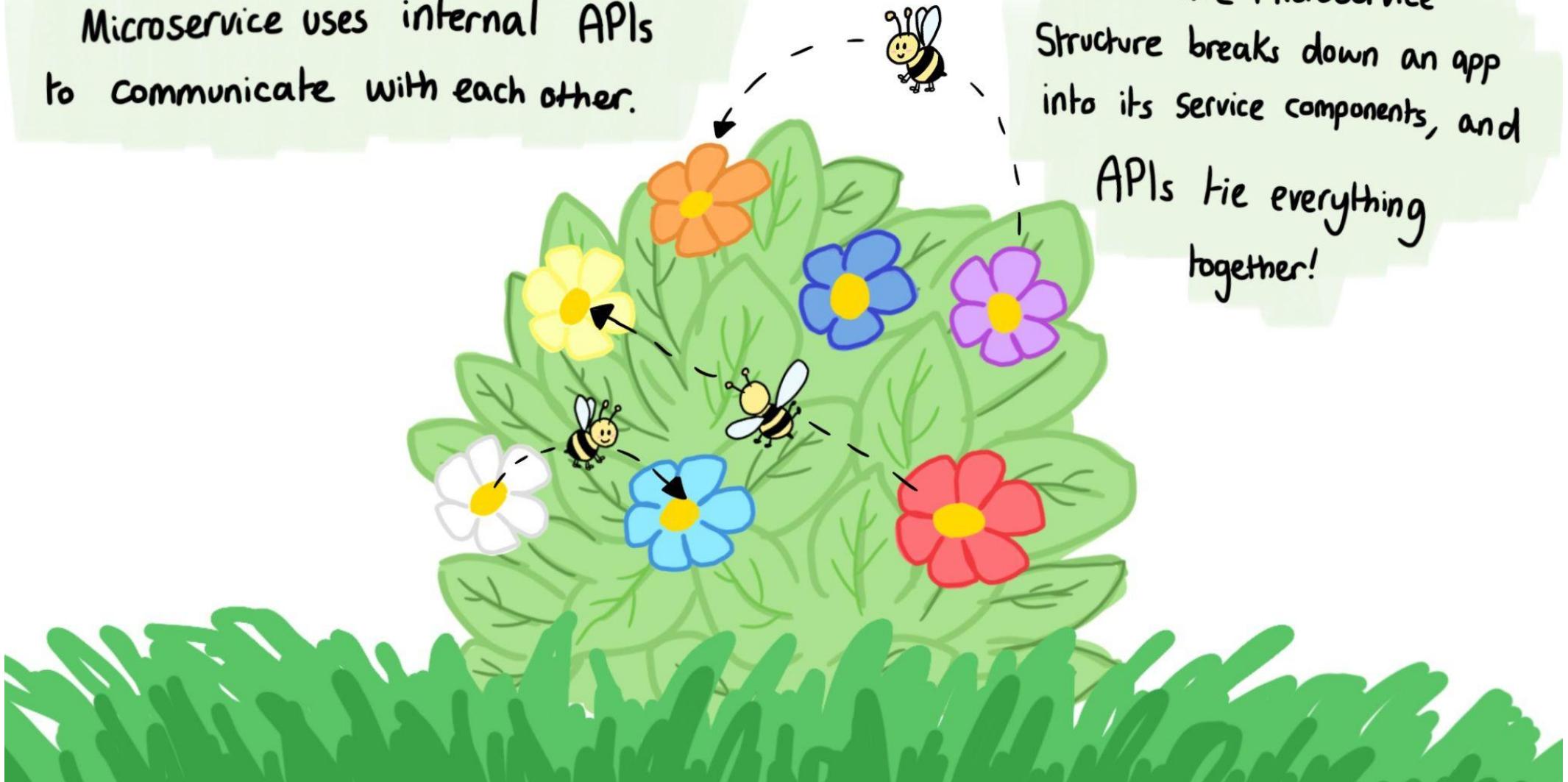


How do they work together?

Each individual service within a Microservice uses internal APIs to communicate with each other.

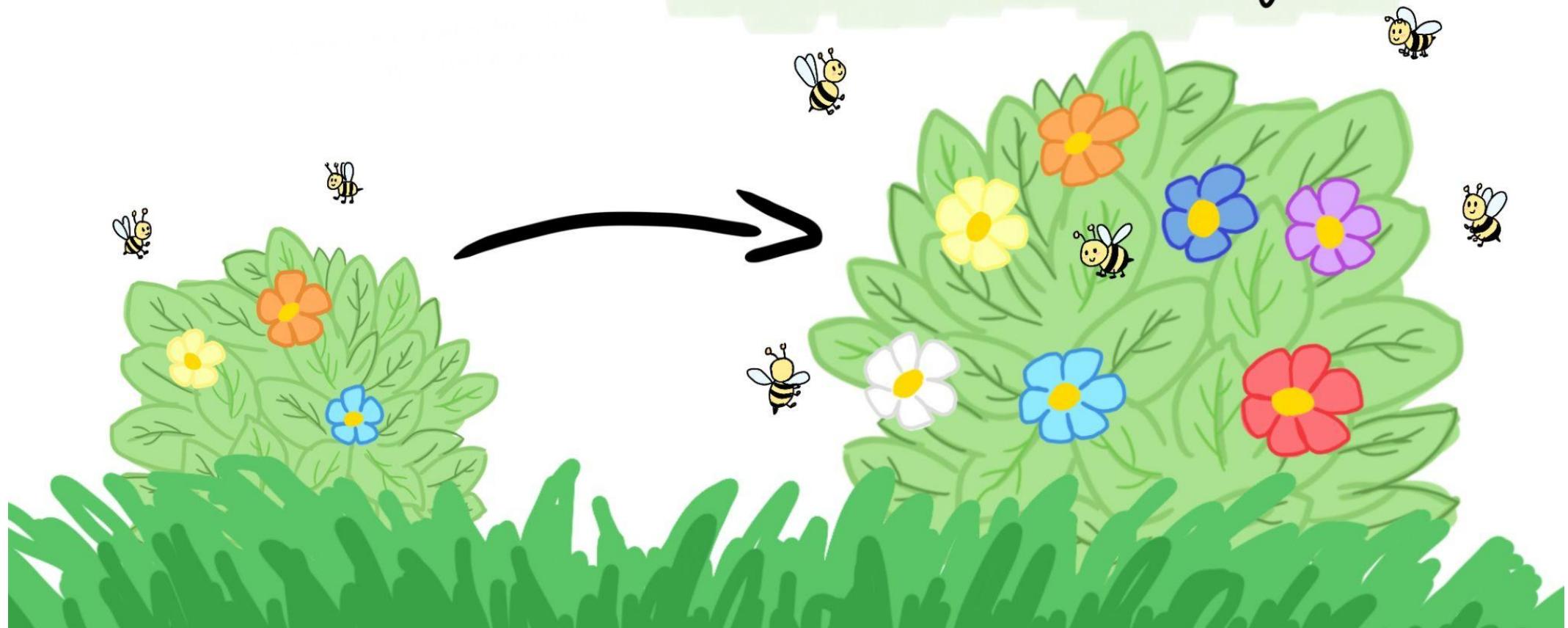
The Microservice Structure breaks down an app into its service components, and

APIs tie everything together!



Microservices create robust and flexible apps. If one service is compromised, it will only affect that Service, and not the entire app.

They are also scalable, so new modules can be added to the app as it grows.



How DNS works

@Rapid_API 🐈

How DNS Works

DNS stands for Domain Name System

DNS is the system that translates domain names into IP addresses.



rapidapi.com
2606:4700:3108::ac42:2918

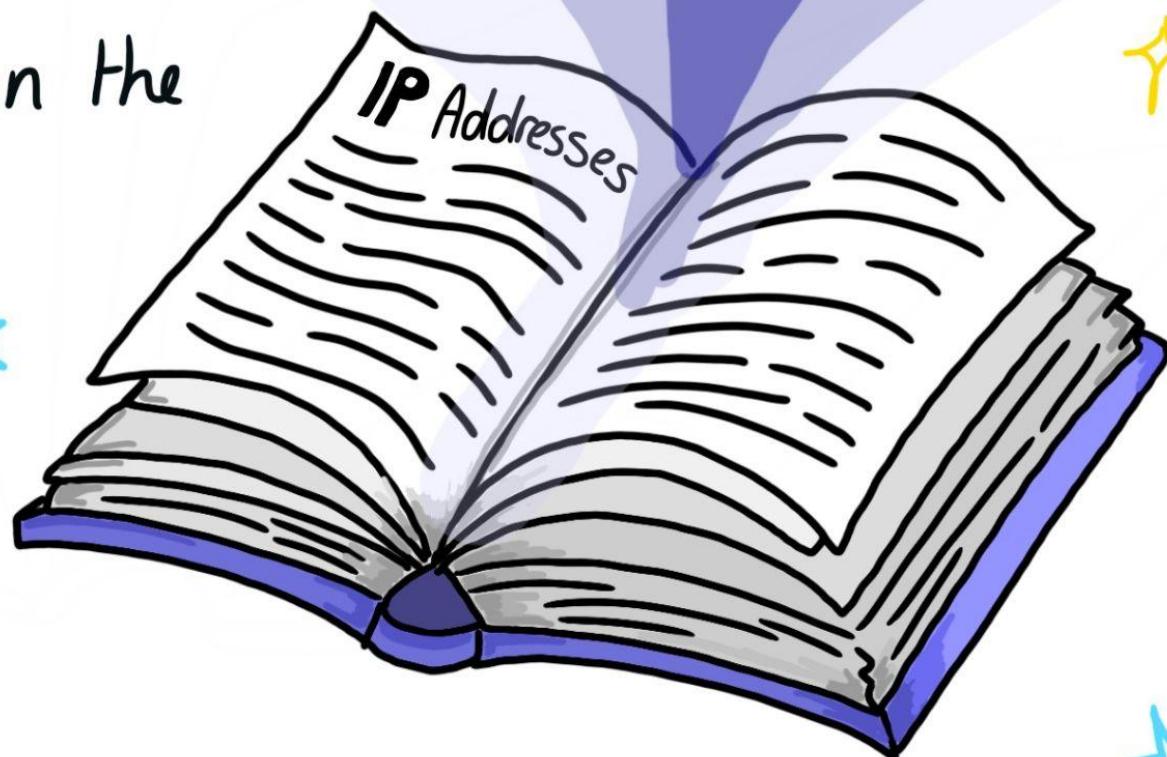
Computers and servers use IP addresses to identify websites and direct your browser to the correct one.

You can think
of DNS acting like

an address book for

every website on the

Internet.



There are **5** basic steps in the **DNS** System:



@Rapid_API



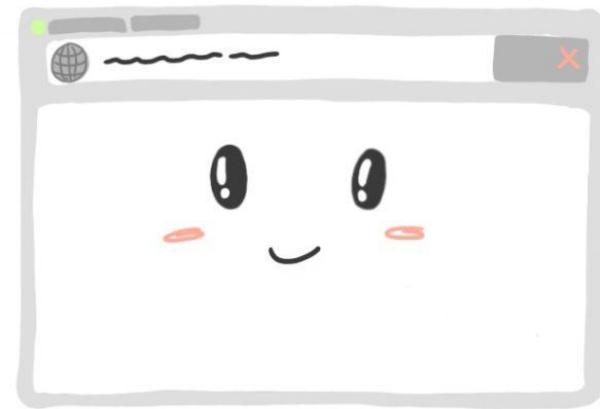
DNS Cache

User Searches for a site



I'll just check my Cache to see if I already know this site...

Your browser



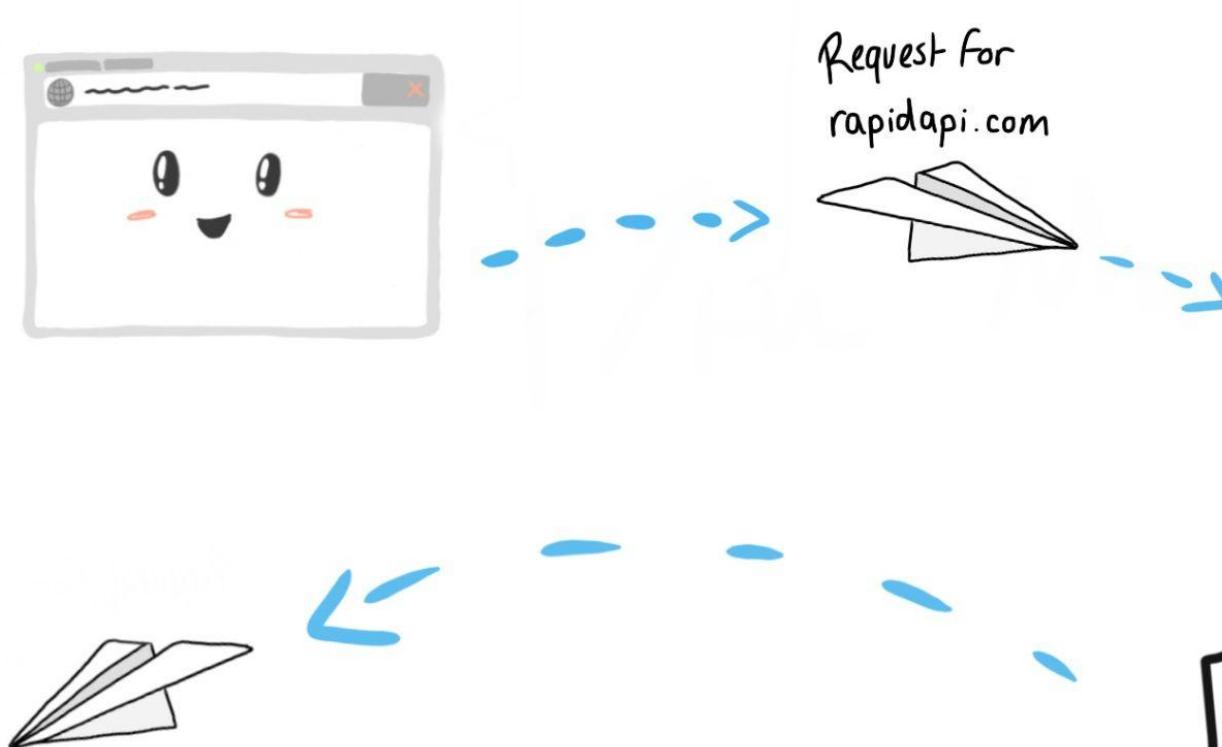
Nope! I don't know this one.
Let me send it to a Resolver.



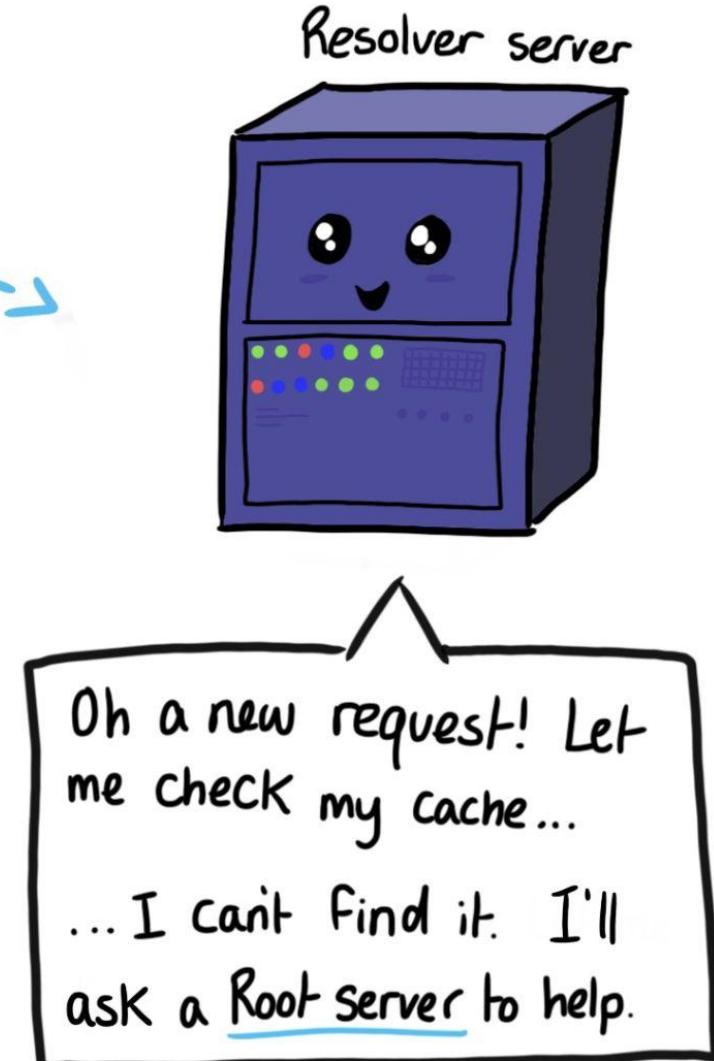
If the website is cached, it will be loaded and the DNS system stops here.



Resolver Server



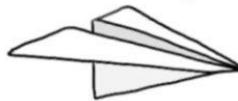
Resolver Servers are typically managed by your ISP (internet service provider).



Step 3

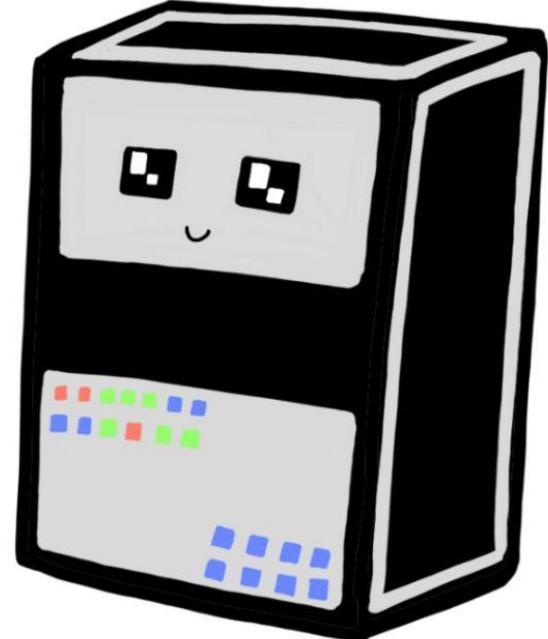
Root Server

rapidapi.com



RapidAPI.com is a cloud-based API platform. When you enter a URL like "rapidapi.com" into your browser, your computer sends a request to a Resolver Server. This server then queries the Root Server to find the correct TLD server for the domain.

Root Server



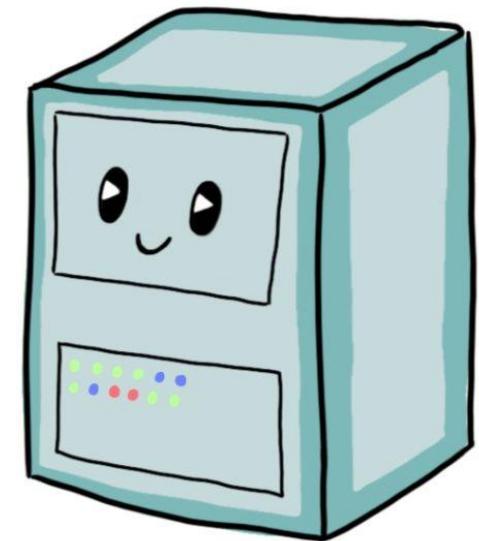
I don't know this site, but
I can redirect you to the
.COM TLD server!



TLD Server

There are TLD servers for domain endings (.com/.org./.net etc), as well as country codes, like .de for Germany, or .in for India.

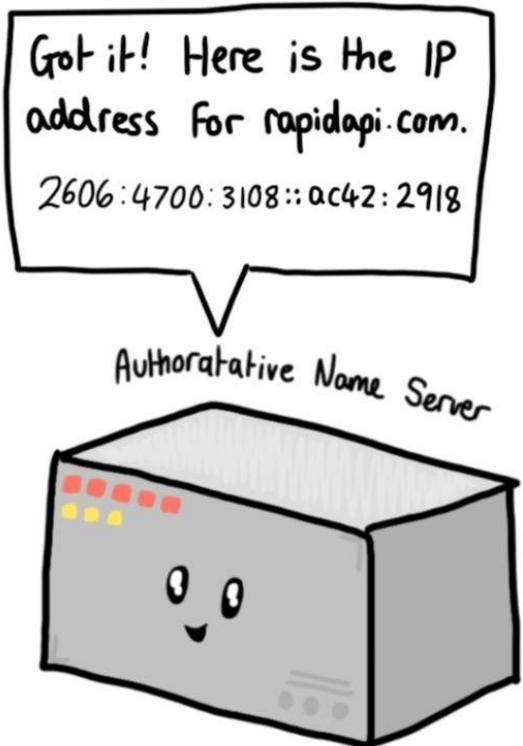
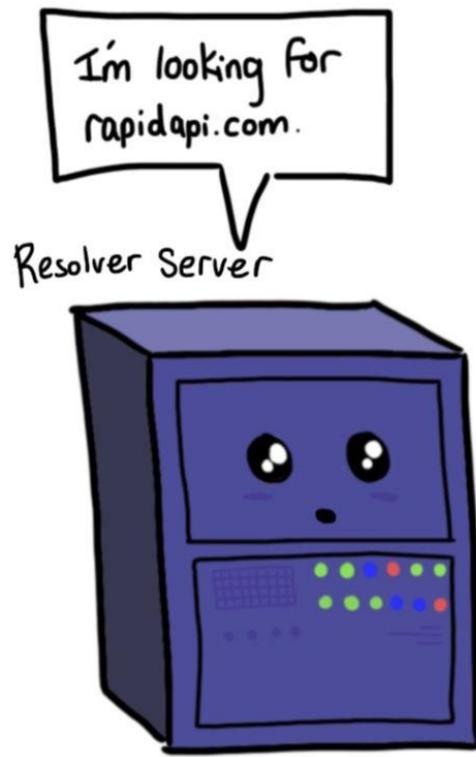
.Com TLD Server



I don't know the IP address for rapidapi.com, but I can redirect the Resolver to the correct Authoritative Name Server.

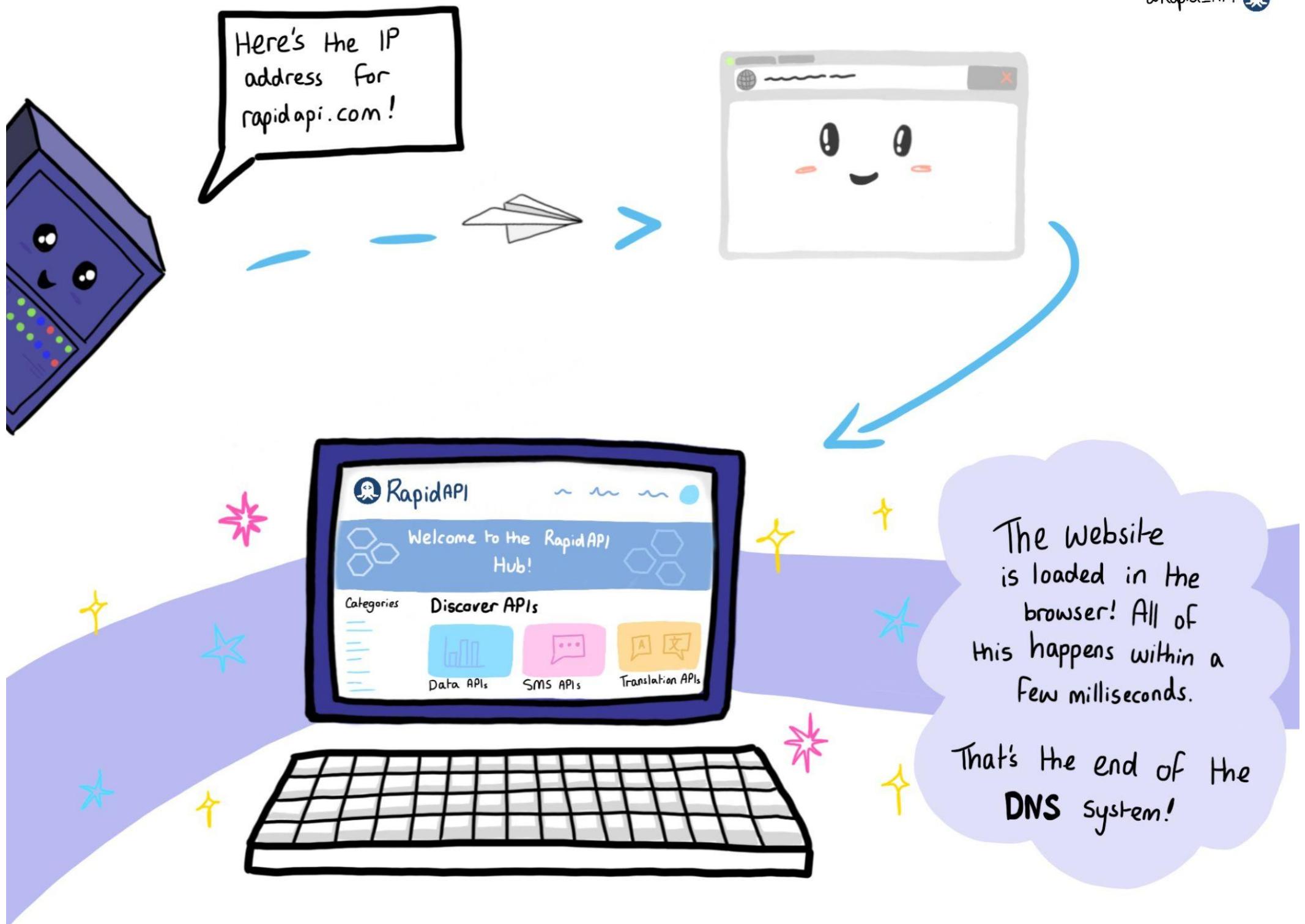


Authoritative Name Server



Authoritative Name Servers are responsible for knowing everything about the domain.

The Resolver can now send the IP address back to the Client.



DevRel Stack

We hope you find our book interesting and valuable. We have an entire ***infrastructure to learn API development***. Check out our DevRel Stack.

| | |
|-----------------------------------|---|
| RapidAPI Learn | Find challenges (with solutions!) and interactive learn API labs |
| RapidAPI Guides | Short & long-form API Development guides (interactive examples) |
| RapidAPI Courses | Free video courses by RapidAPI and RapidAPI Developer Experts |
| RapidAPI Threads | Twitter threads on RapidAPI and API Development (own our content) |
| RapidAPI Comics | Sketch notes and comics on API Development with RapidAPI tools |
| RapidAPI Examples | Open-source starter kits for building APIs & Applications with RapidAPI |