



Rapid API

★ COMICS ★

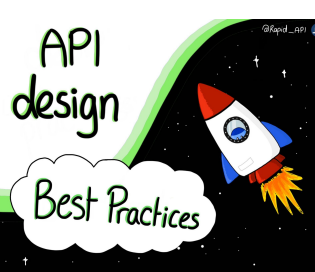
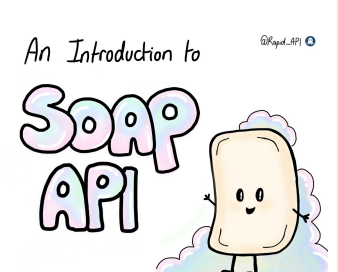
eBook

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



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

Table of Comics

 <p>What is an API?</p> <p>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.</p>	 <p>What is HTTP?</p>	 <p>Widely used HTTP Methods</p>	 <p>The Anatomy of HTTP Requests</p>	 <p>How API Endpoints Work</p>
 <p>API design Best Practices</p>	 <p>What is a RESTful API?</p>	 <p>An Introduction to SOAP API</p>	 <p>An Introduction to GraphQL</p>	 <p>An Introduction to Webhooks</p>
 <p>What's the difference between REST and GraphQL?</p>	 <p>JSON VS XML</p>	 <p>What are HTTP Cookies?</p>	 <p>Synchronous and Asynchronous Programming</p>	 <p>What's the difference between an SDK and an API?</p>
 <p>Microservices vs APIs</p>	 <p>How DNS Works</p>			

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.

HEY!
We need some more nectar (data) to make our honey (app).

Client

API

Server



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

APIs use HTTP protocols to transfer data.

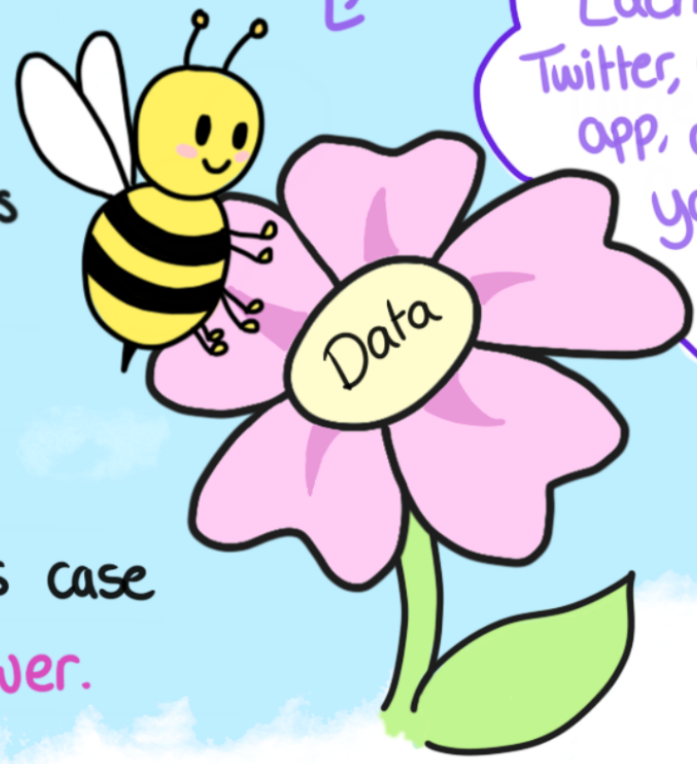
GET/nectar/pinkflower



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!

Response



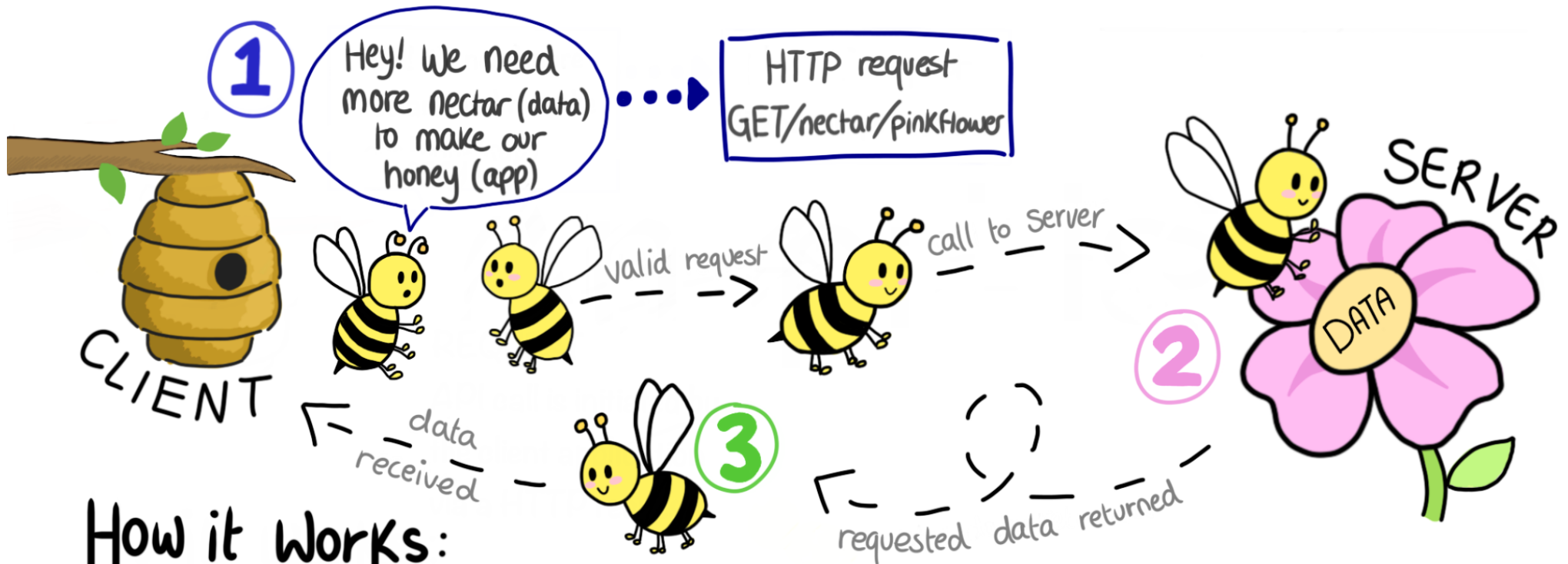
As long as the server (Flower) can return the requested data (nectar) to the client successfully, then **mission accomplished!**

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

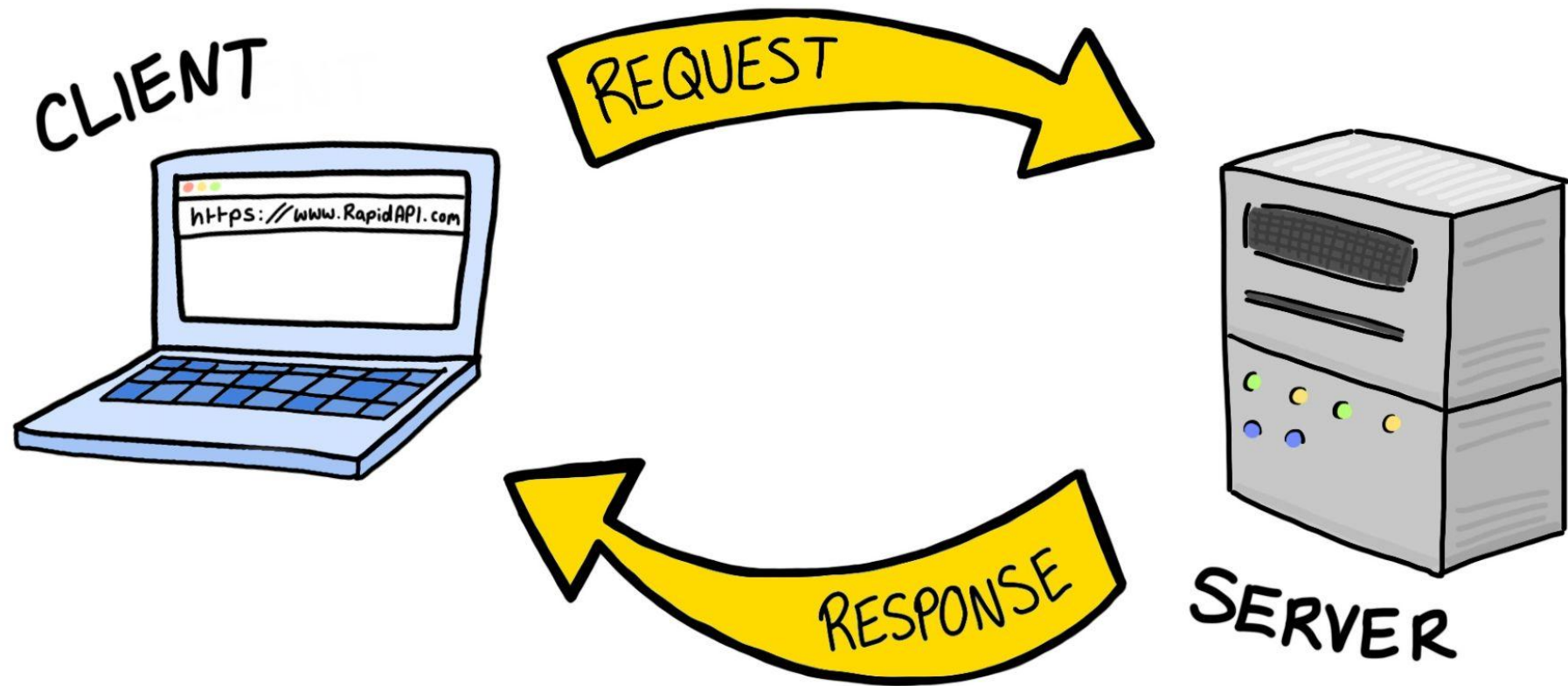
HTTP?

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

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

★ 1XX - Informational

★ 2XX - Success

★ 3XX - Redirection

★ 4XX - Client Error

★ 5XX - Server Error

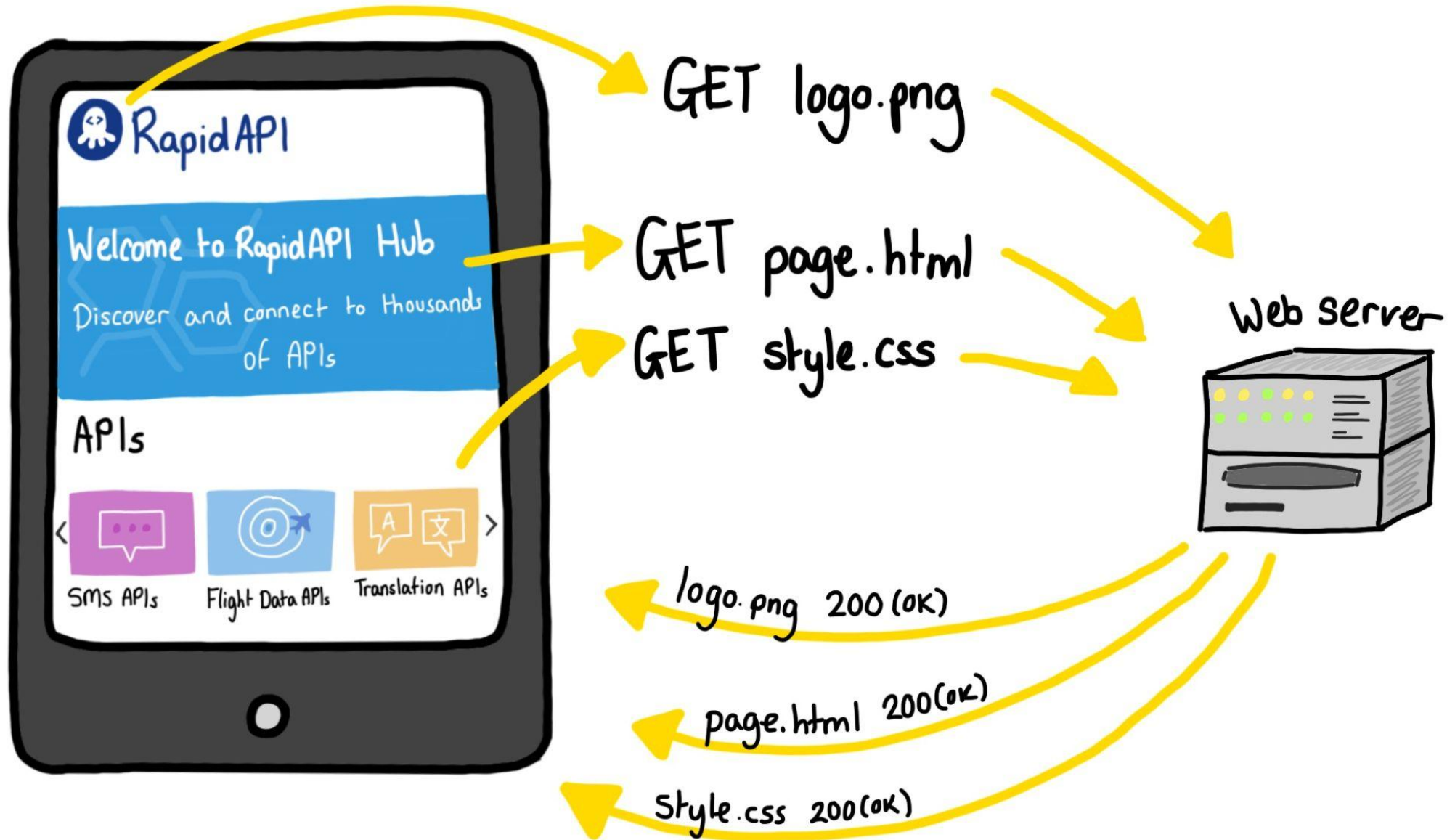
Example: 200 (OK)

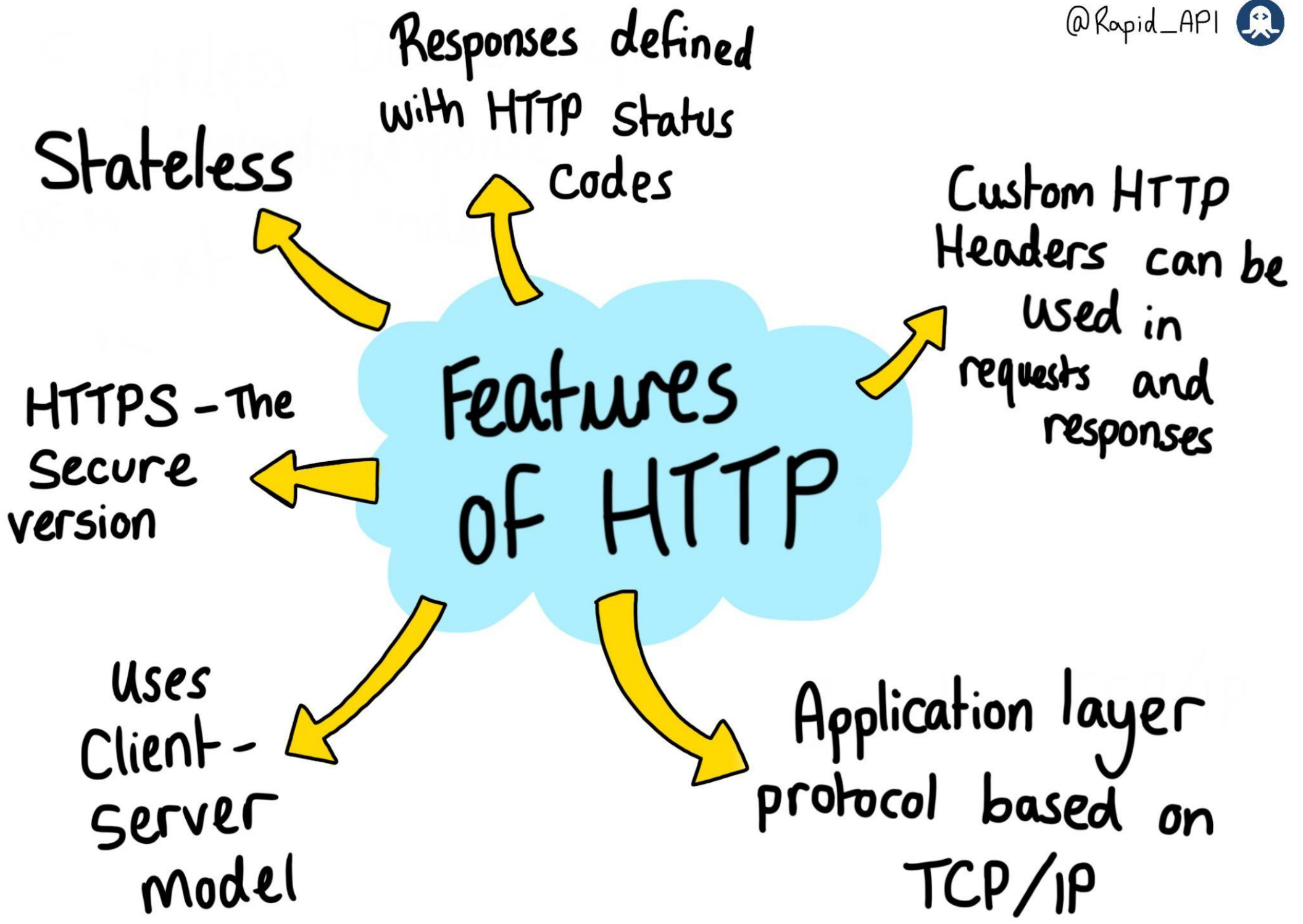
Successful with no problems.

Example: 404

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

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





Responses defined with HTTP Status Codes

Stateless

Custom HTTP Headers can be used in requests and responses

Features OF HTTP

Application layer protocol based on TCP/IP

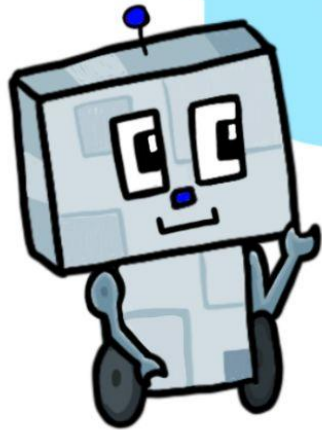
Uses Client-server model

HTTPS - The Secure version

Widely used HTTP methods

Widely used

HTTP

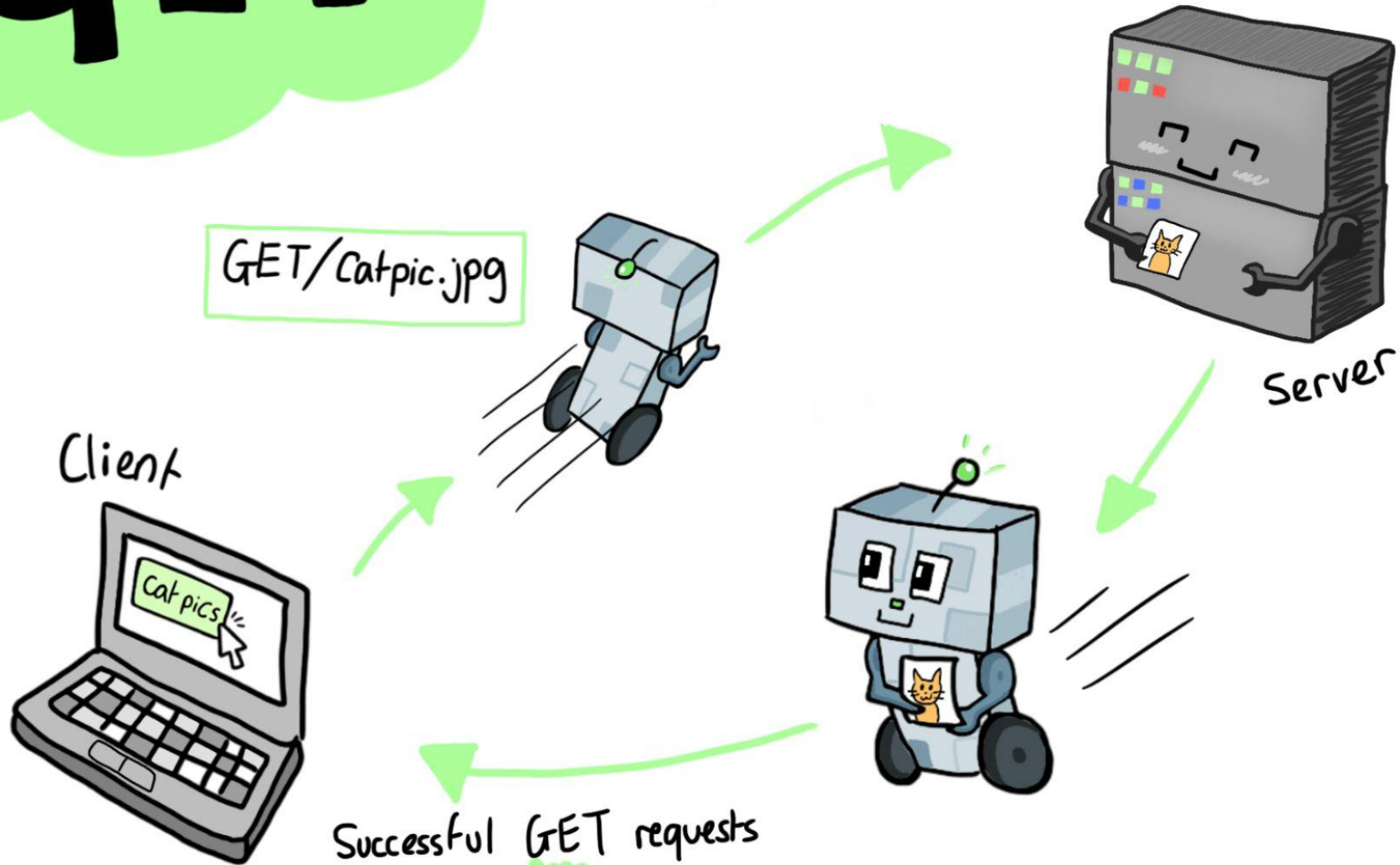


Methods

GET

GET requests retrieve a resource from a server.

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

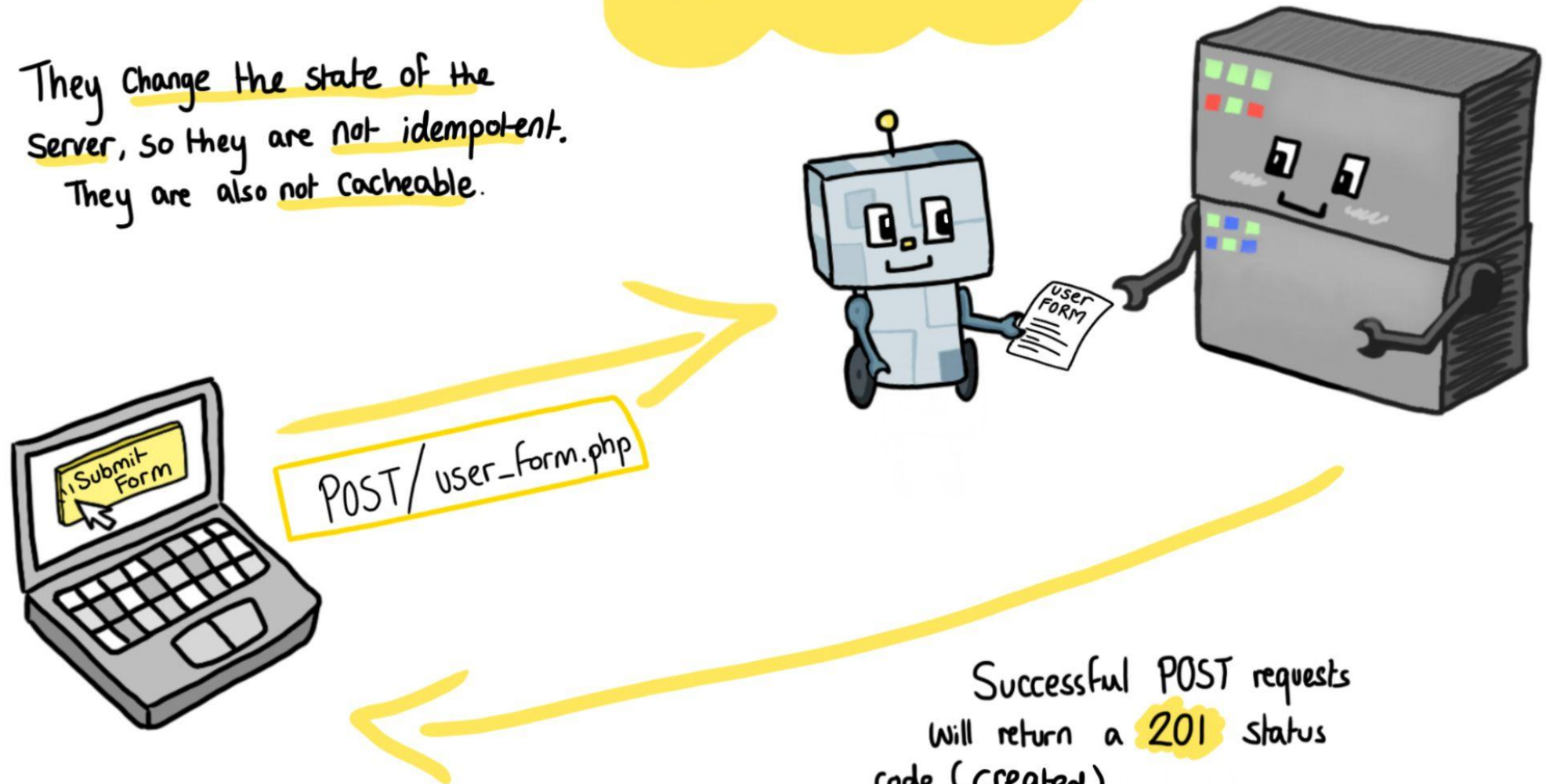


Successful GET requests return a 200 status code.

POST requests submit information to the server.

POST

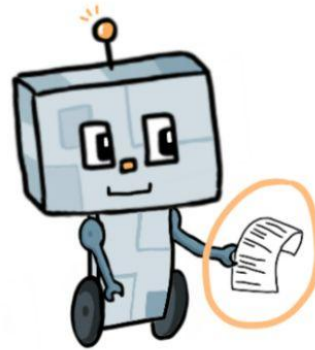
They change the state of the server, so they are not idempotent.
They are also not cacheable.



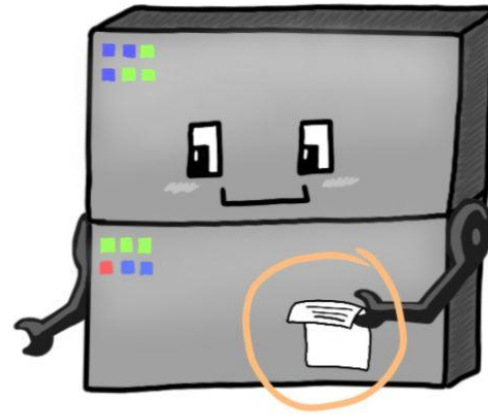
PUT

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.



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



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

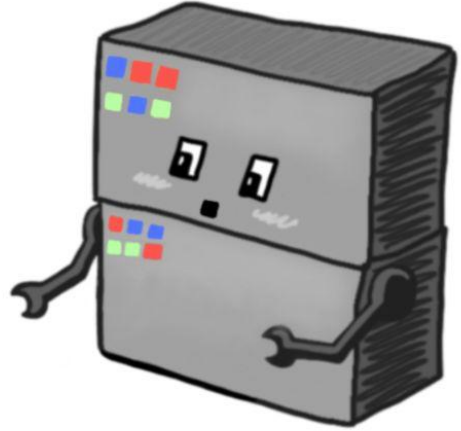
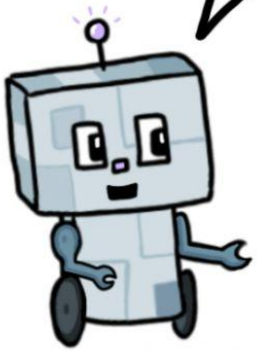
PATCH

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



```
PATCH /users/1  
{ "email": "coolbot96@robotmail.com" }
```

Coolbot96@robotmail.com ❌
↓
Coolbot97@robotmail.com ✅



Successful PATCH methods will return **2XX**

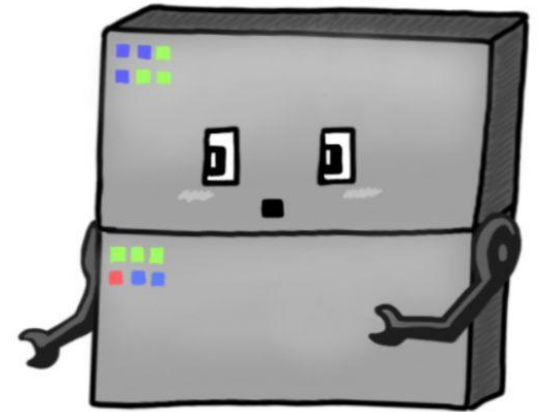
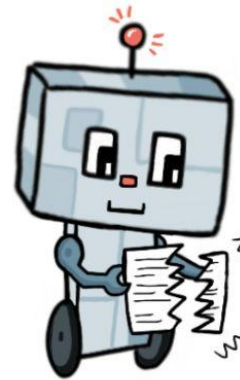
DELETE

The delete method is idempotent.

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



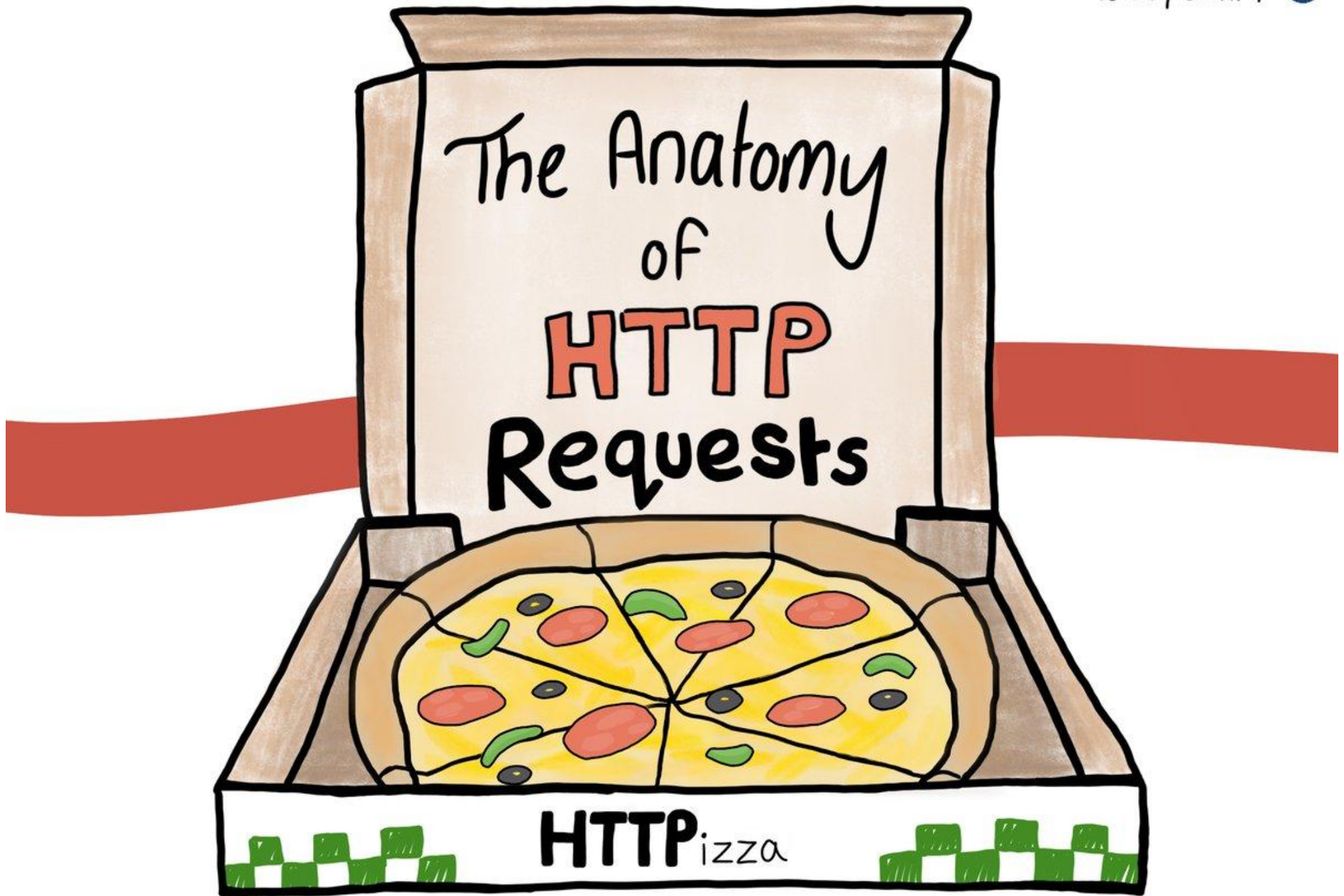
DELETE /file.html



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

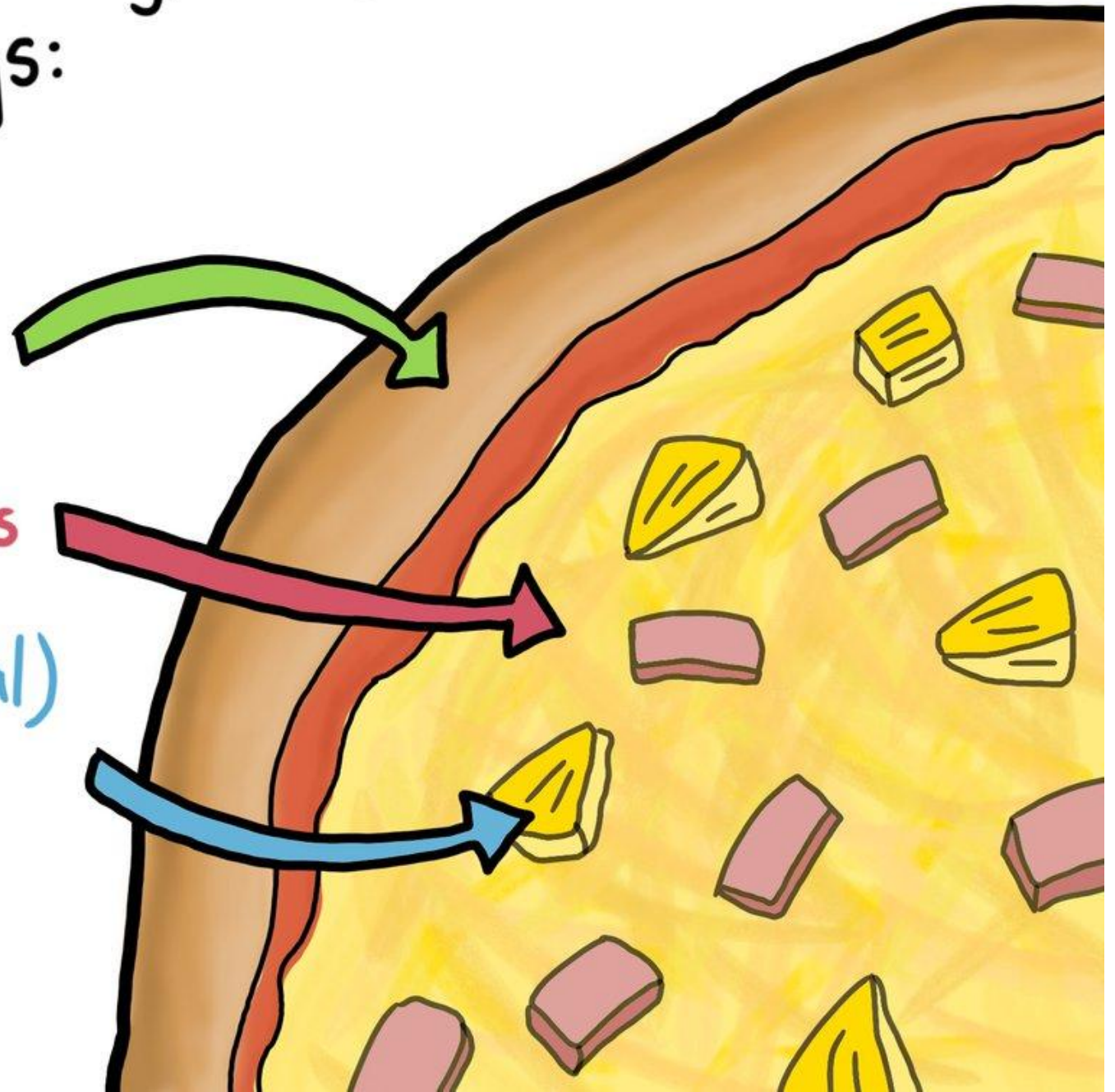
DELETE requests the server to remove the resource identified by the request URL.

The anatomy of HTTP Request



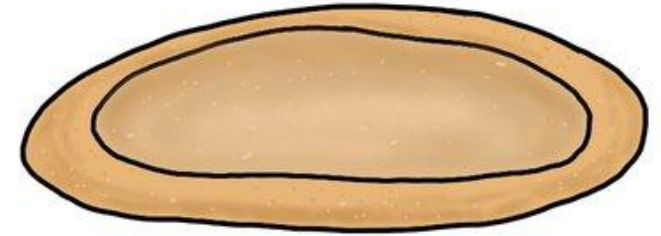
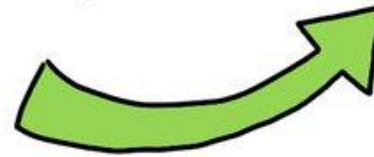
HTTP Requests typically Contain 3 things:

- 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

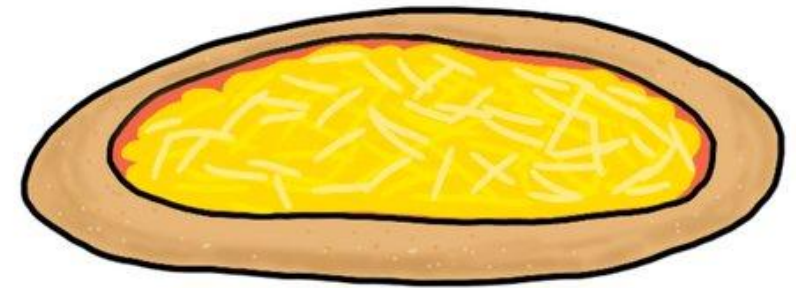
Current HTTP version

GET/orders/{orderId} HTTP/1.1

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

Content-Length: 30

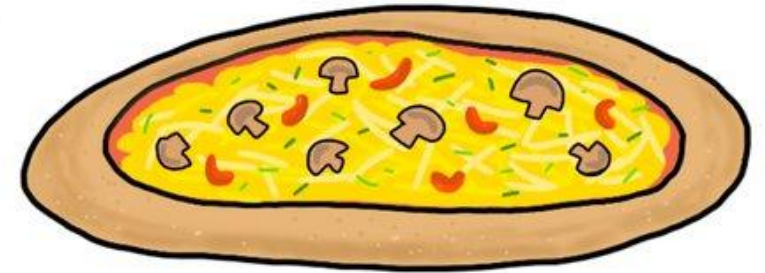
Accept-Language: en, de

Length of data in bytes

Indicates the languages accepted by the client

③ Body (data)

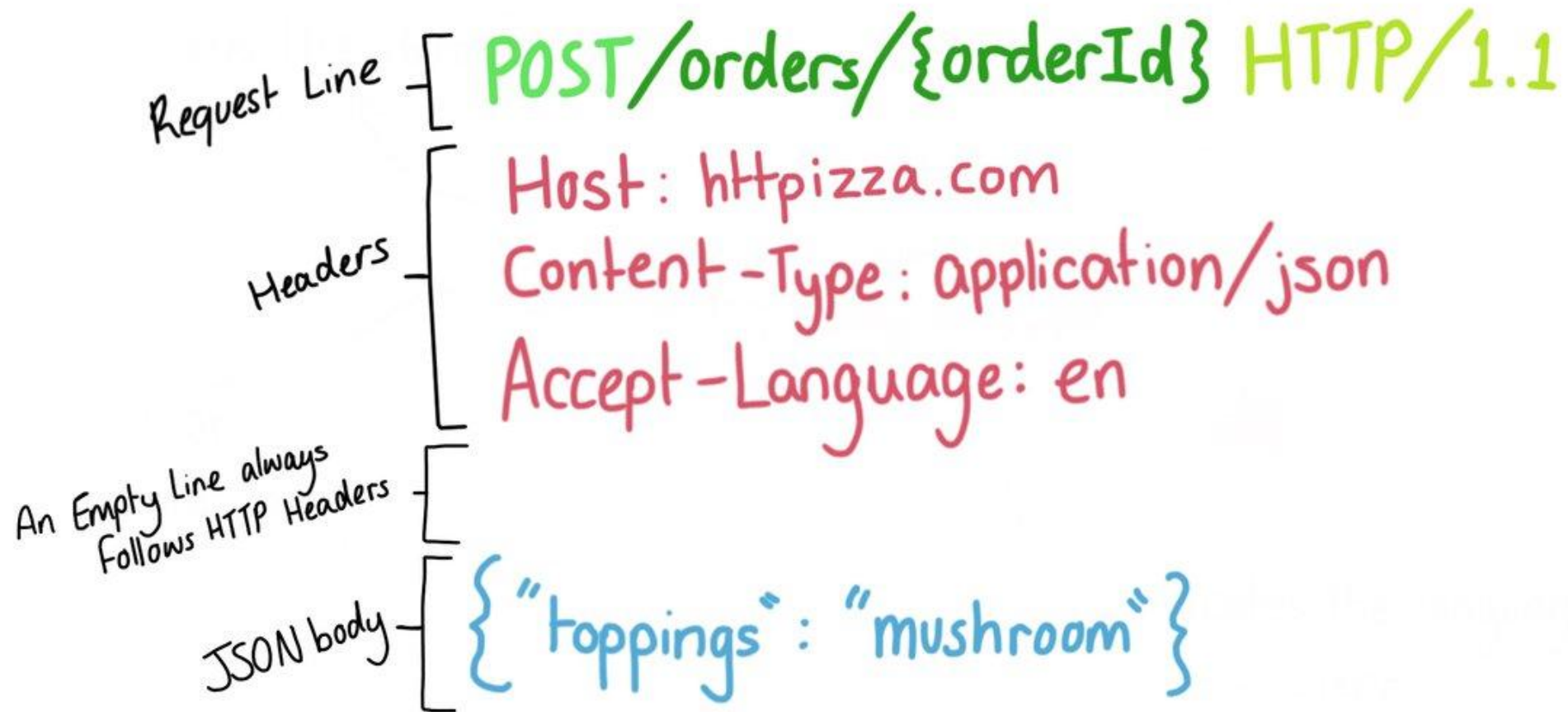
The body is only needed if the HTTP Method is POST, PUT, or PATCH. It contains the data being sent to the server.



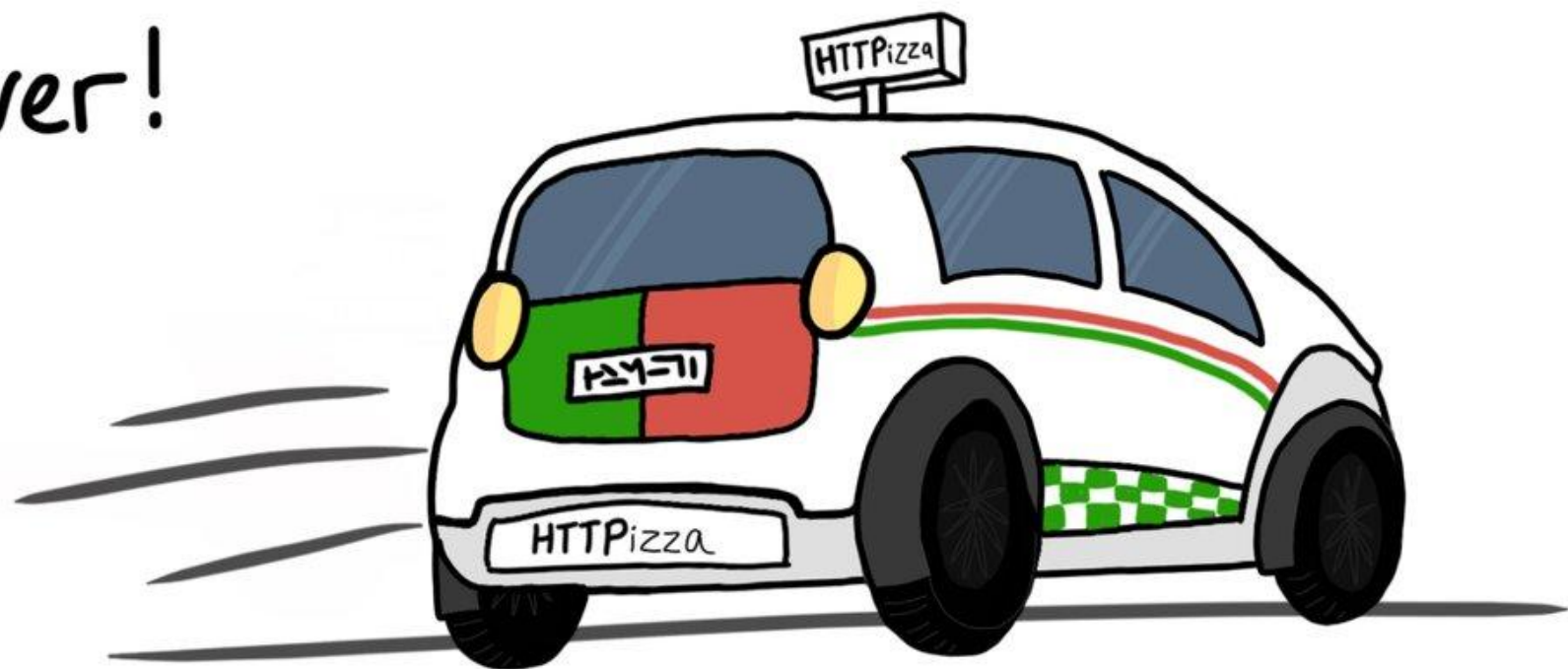
For example, the body could be the details of our pizza order.

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

HTTP Request Structure in full:

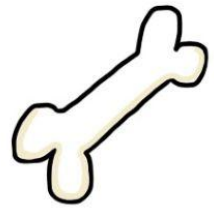
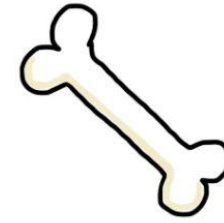


Now the request is ready
to be sent to the
Server!

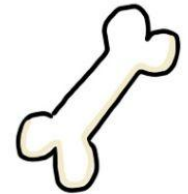


How API endpoints work

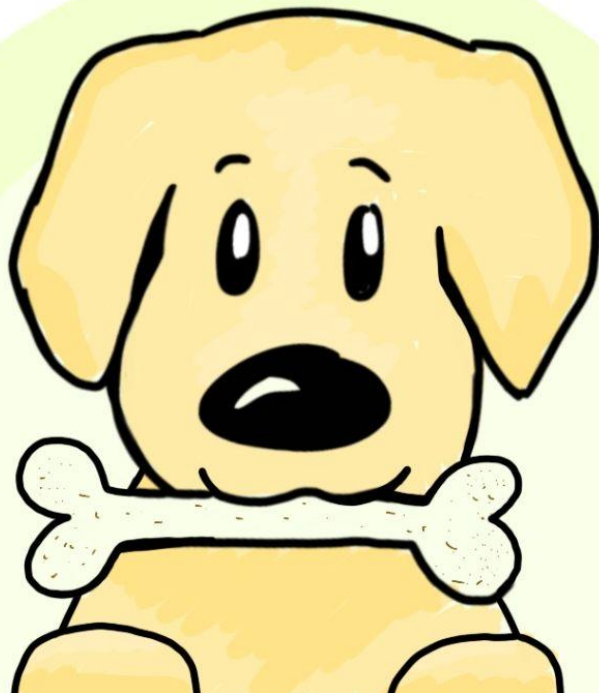
HOW API



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.

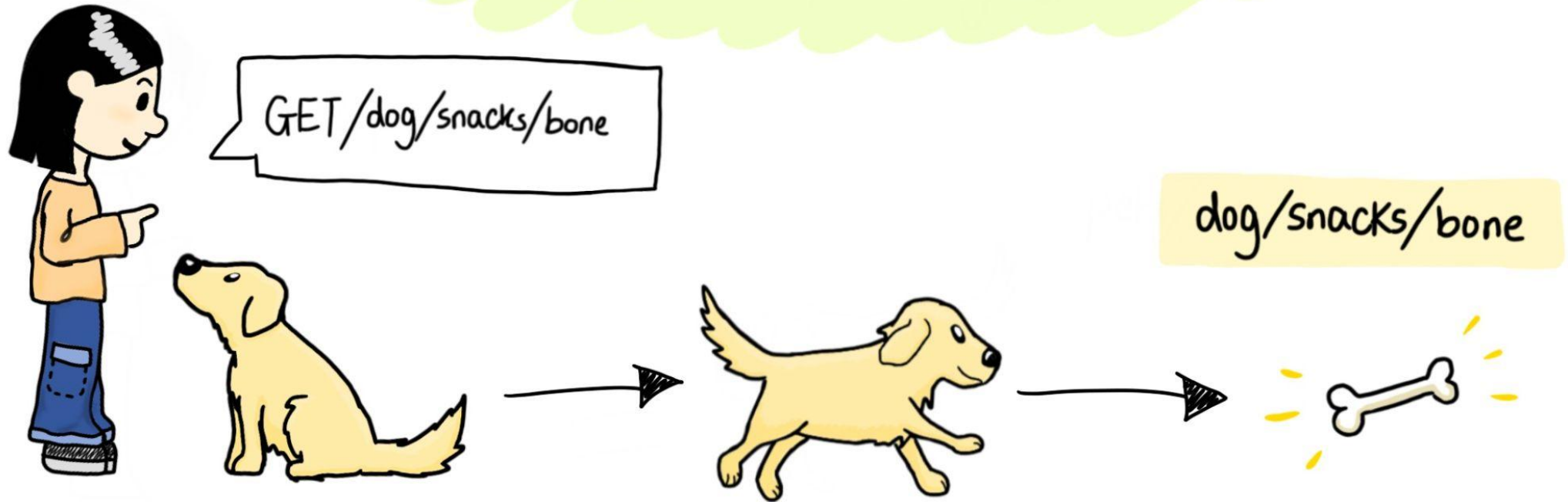
They typically look like this

`pets/dog/snacks/bone`



The role of an endpoint:

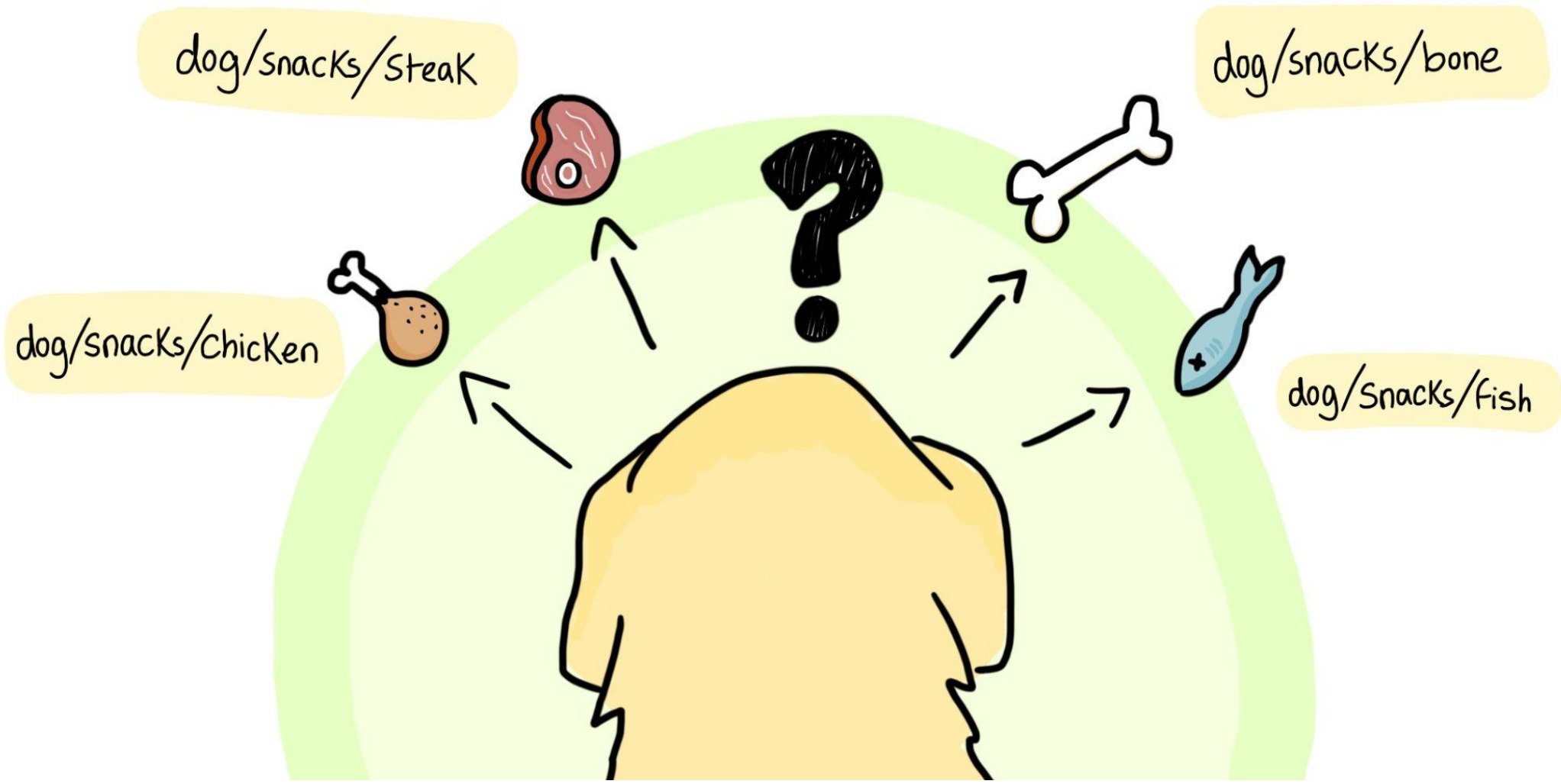
In this example, a resource is being fetched using the GET method.



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 its crucial that they are clearly named after the entity they represent.



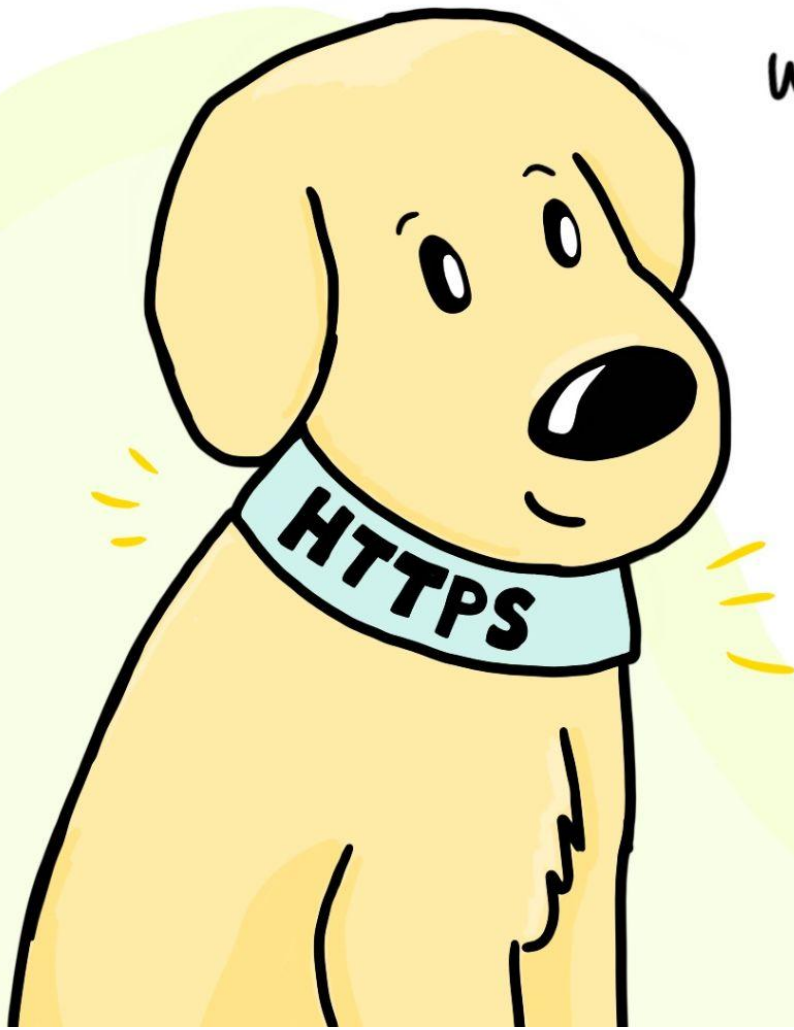
If endpoints are faulty or inaccurate, the API will not be able to locate the correct resource.



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

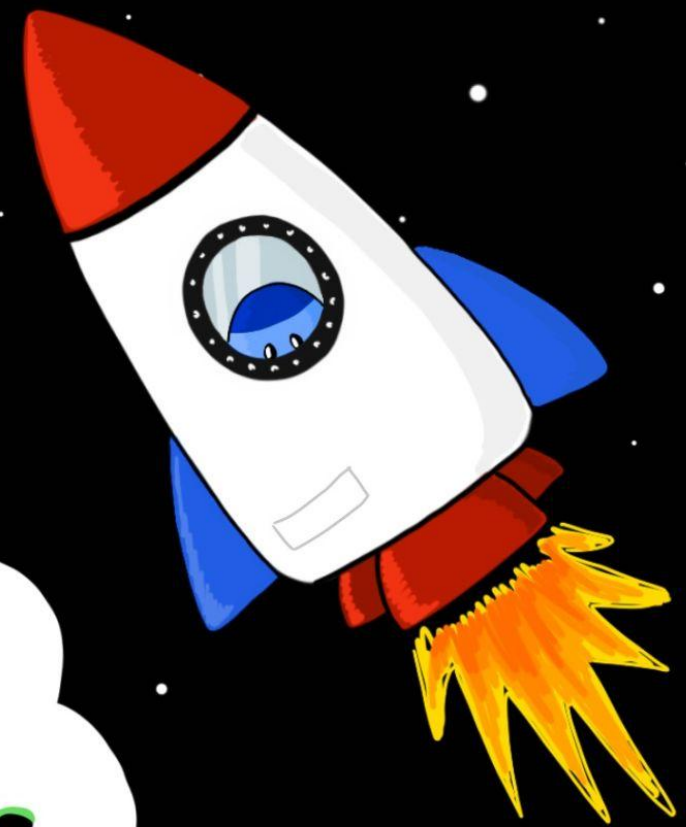


API design best practice



API design

Best Practices



Use Nouns for endpoint paths

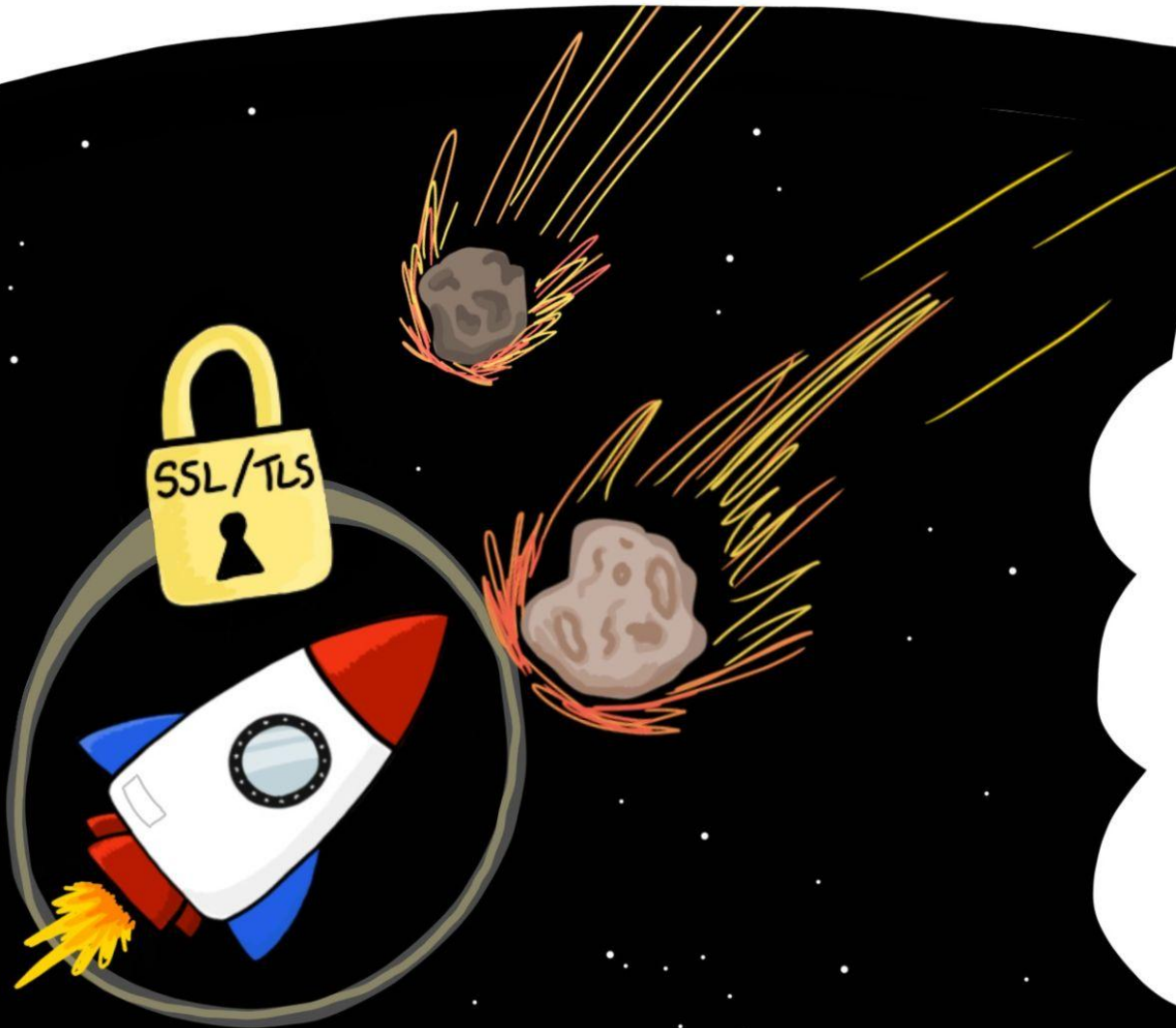
Endpoint paths should always be named after the entity they represent.

The HTTP method is the verb indicating the action. ↴

`GET/rocks/lunar`



Maintain good Security Practices



Always use a
SSL/TLS
connection to keep
data encrypted and
safe from basic
security attacks.

Use JSON

JSON is the standard for transferring data.

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



Pagination

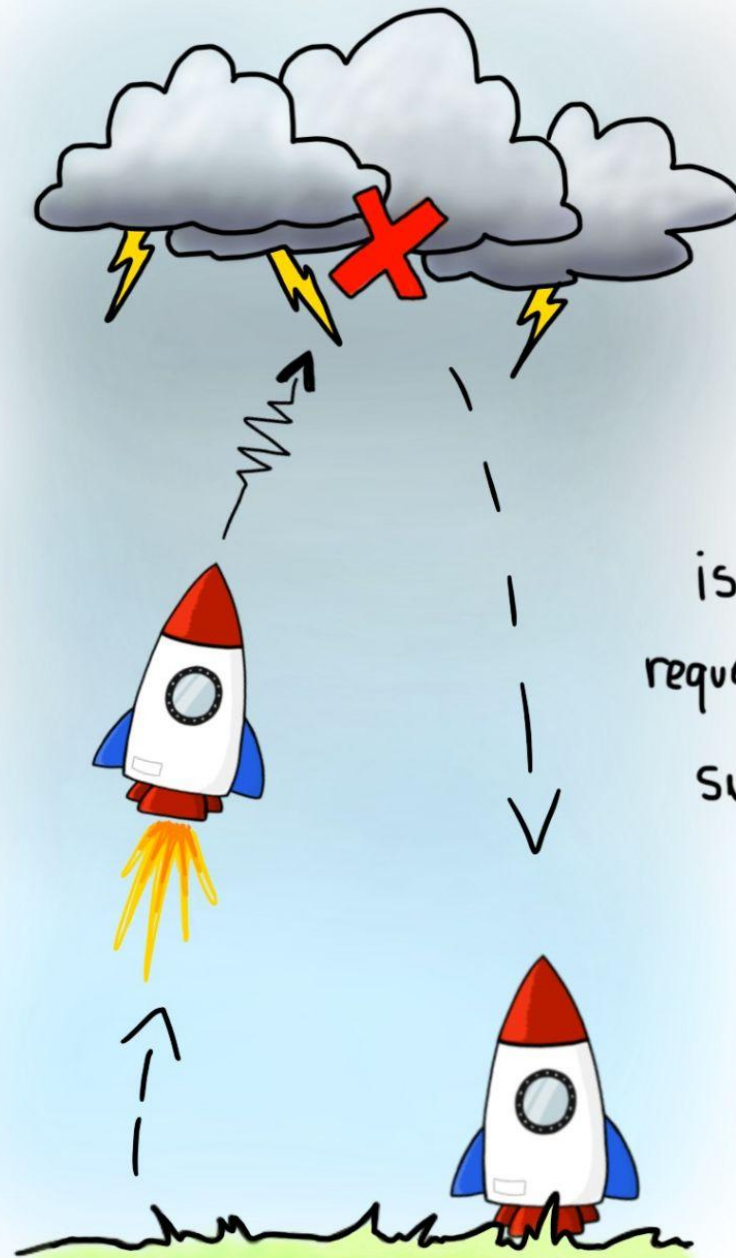


If endpoints return a large amount of data, it can slow a system down.

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.

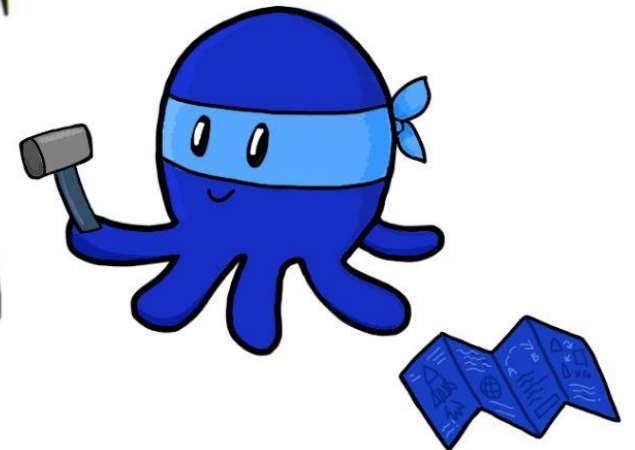
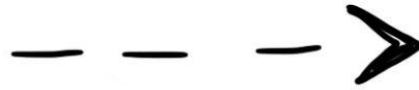
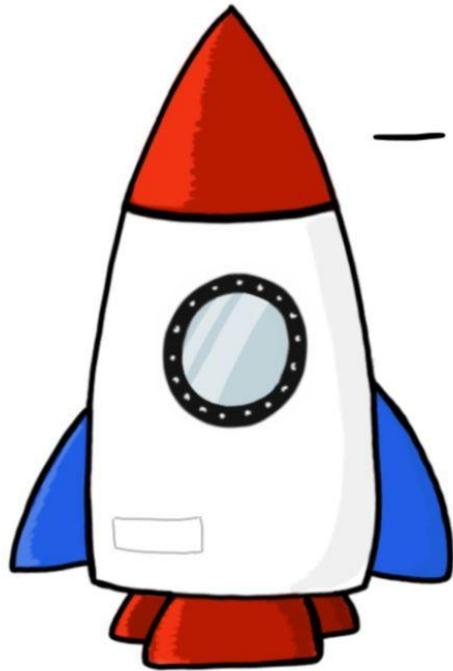


This means the client isn't left waiting on a request if there is an issue, such as a network connection issue.

Versioning

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

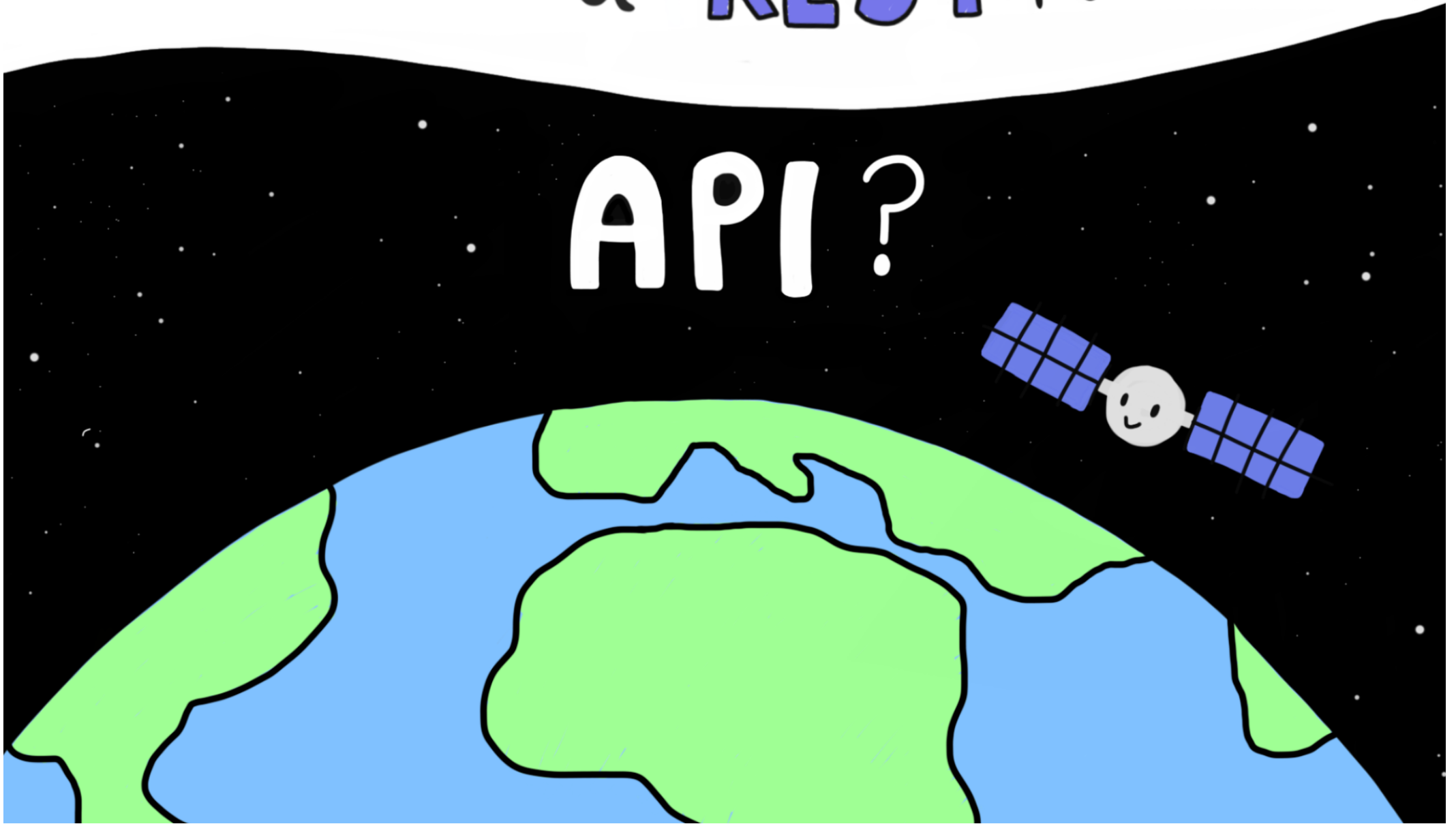
So it's crucial to have previous versions as a backup.



What is a RESTful API

What is a **REST**ful

API?



RESTful APIs follow

REST architecture

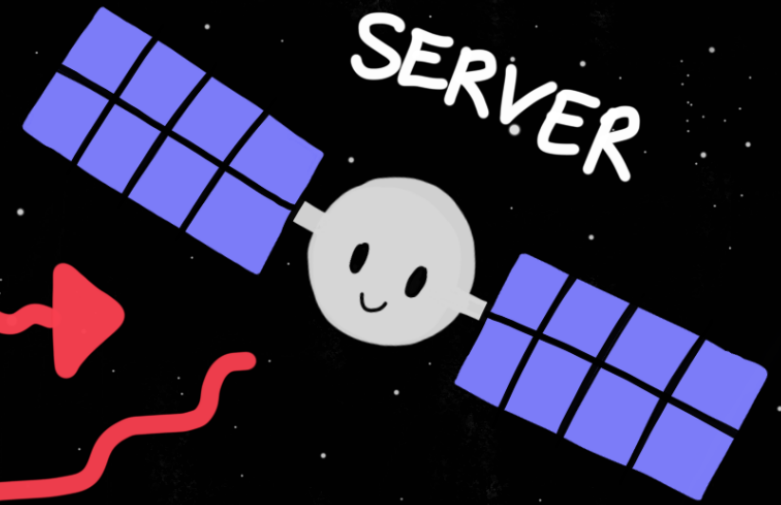
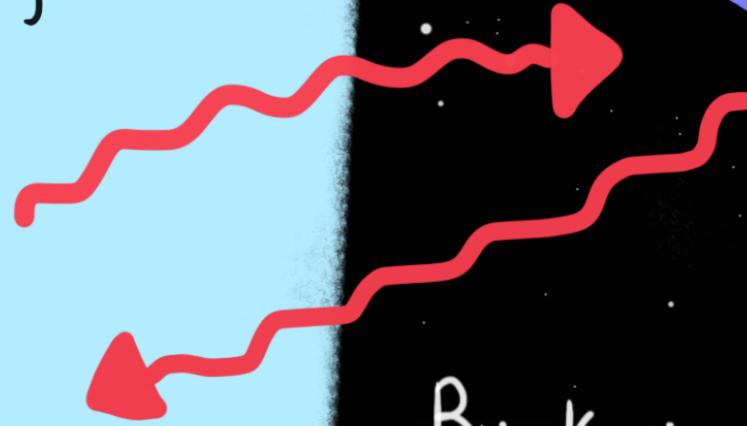
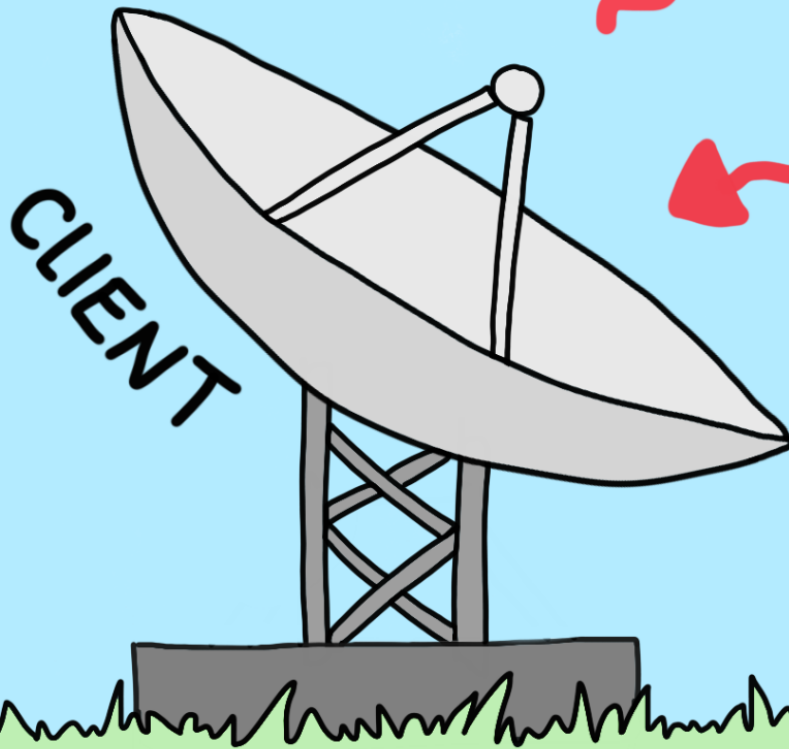
↳ Representational State Transfer

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.



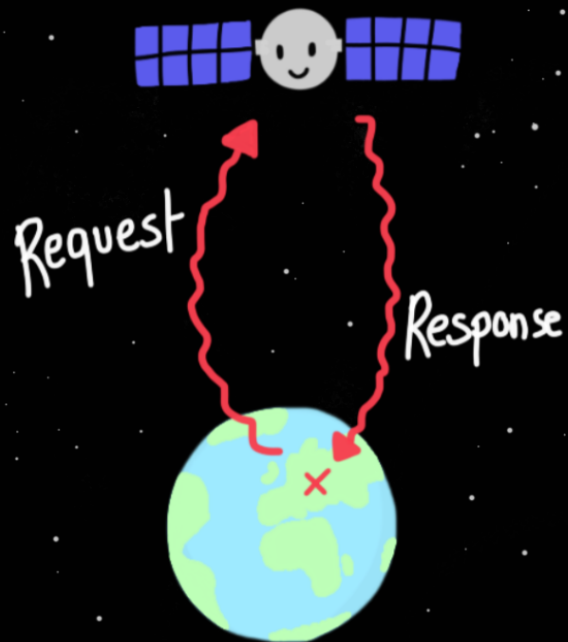
By keeping these two independent, each can be modified without affecting the other.

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:

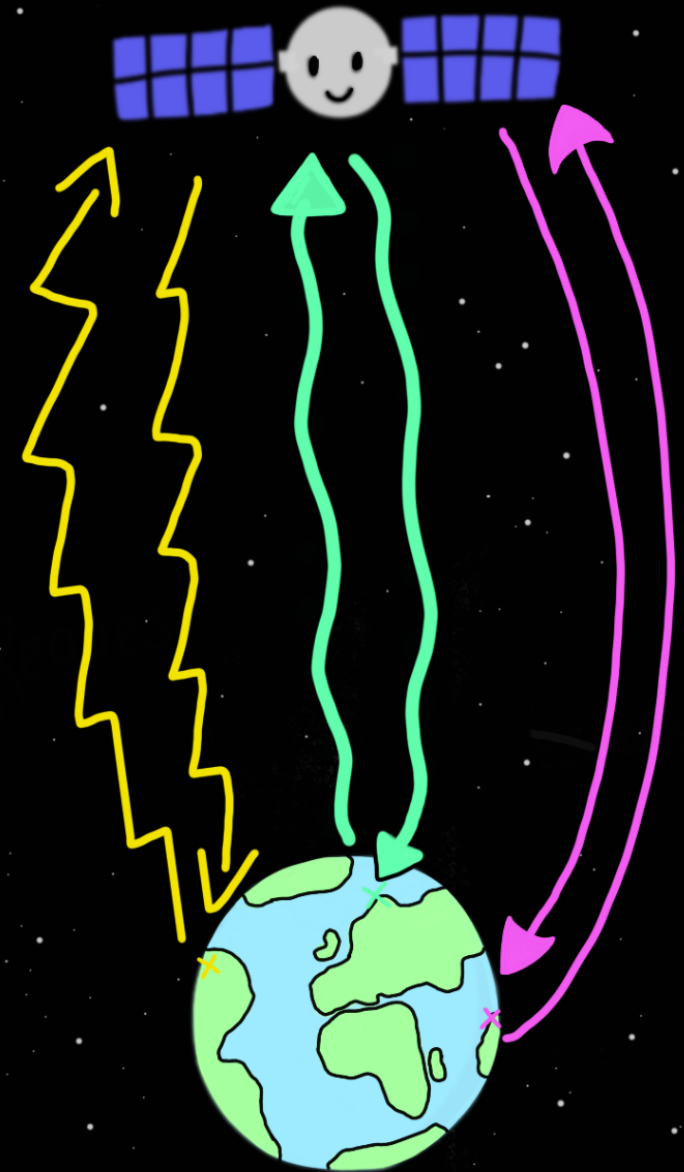
- GET — Retrieves a resource.
- POST — Creates a new resource.
- PUT — Updates an existing resource.
- DELETE — Deletes a resource.

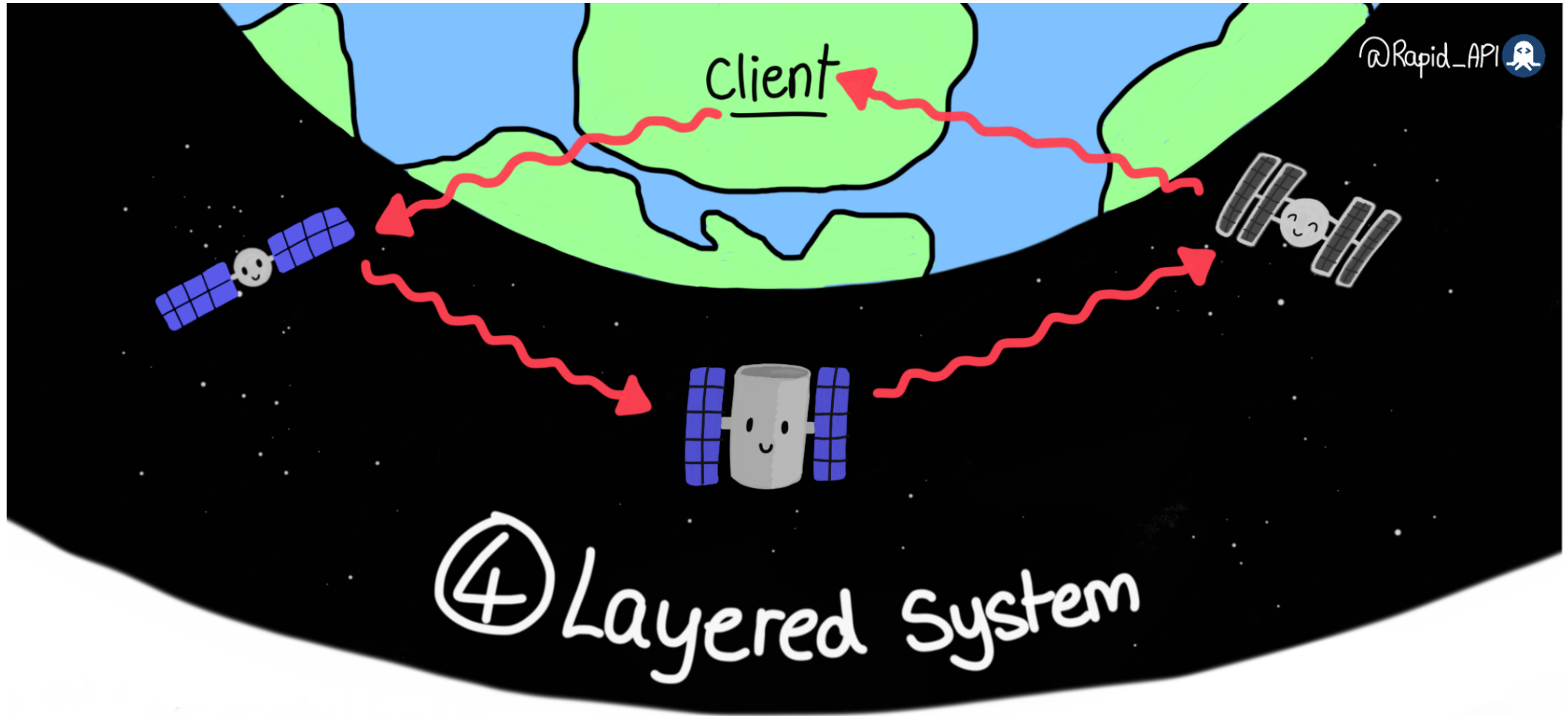
③ Stateless

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.





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.



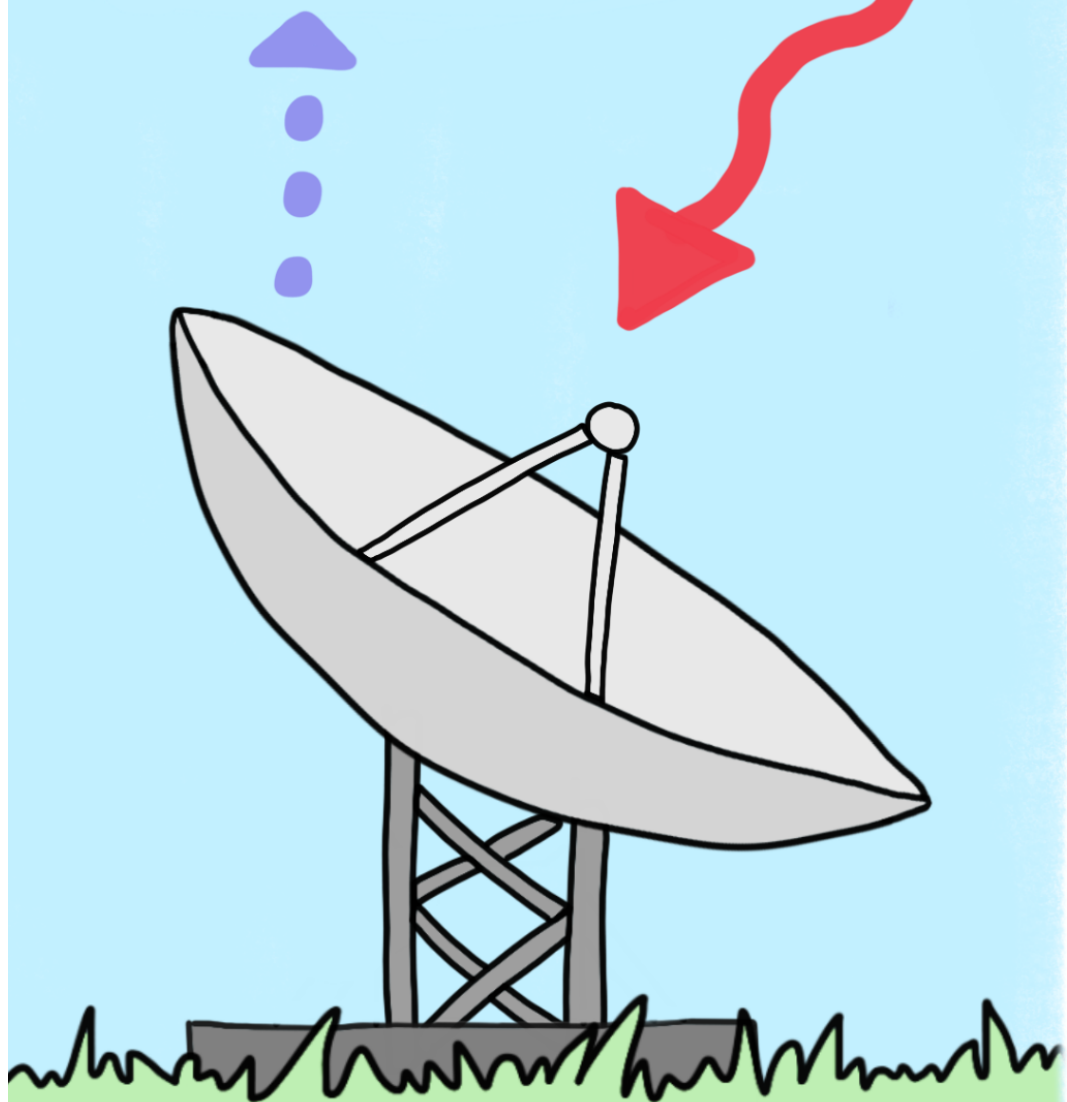
⑤ Cacheable

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.

Client saves received data to local storage.

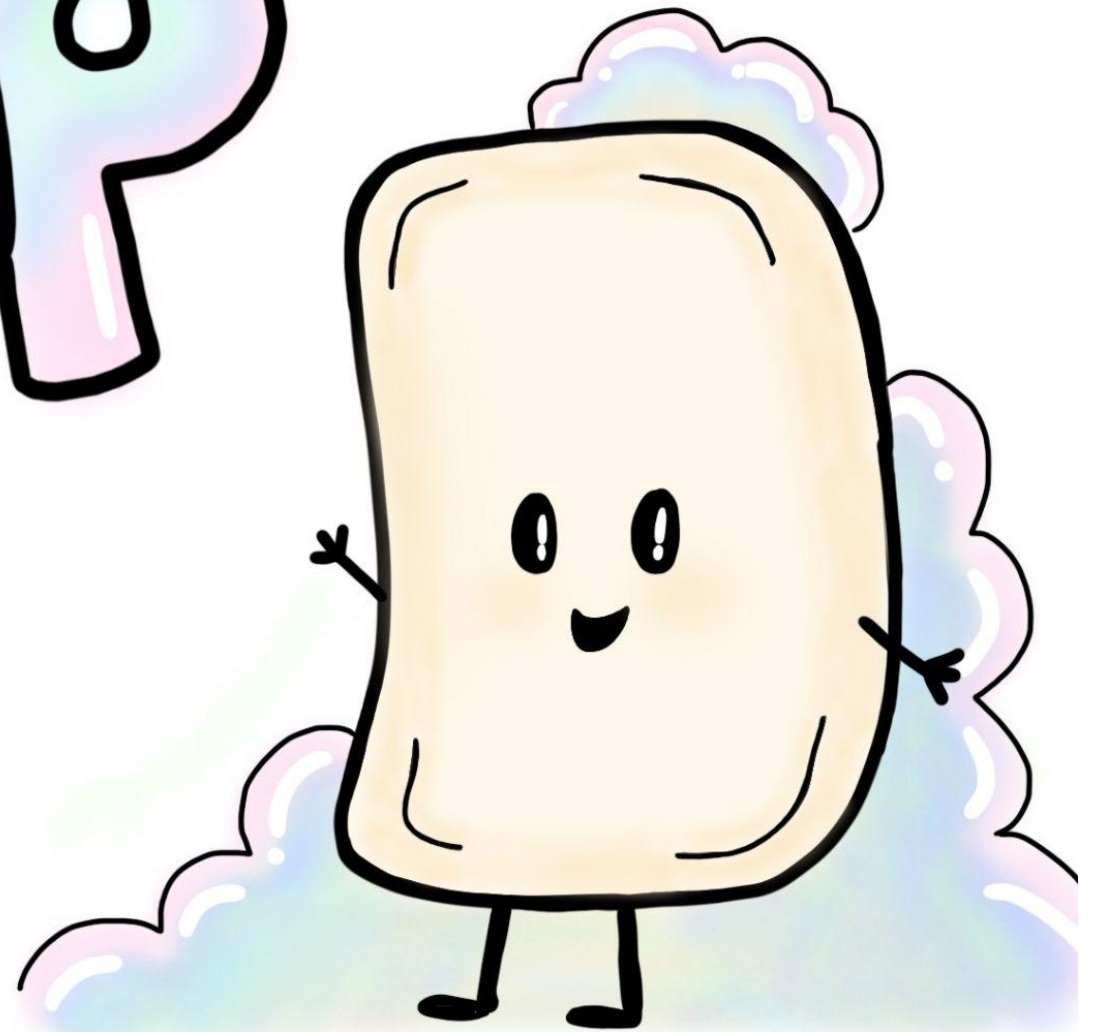


An introduction to SOAP API

An Introduction to

@Rapid_API 

SOAP API



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

HTTP

TCP

FTP

SOAP can use a range of different protocols.

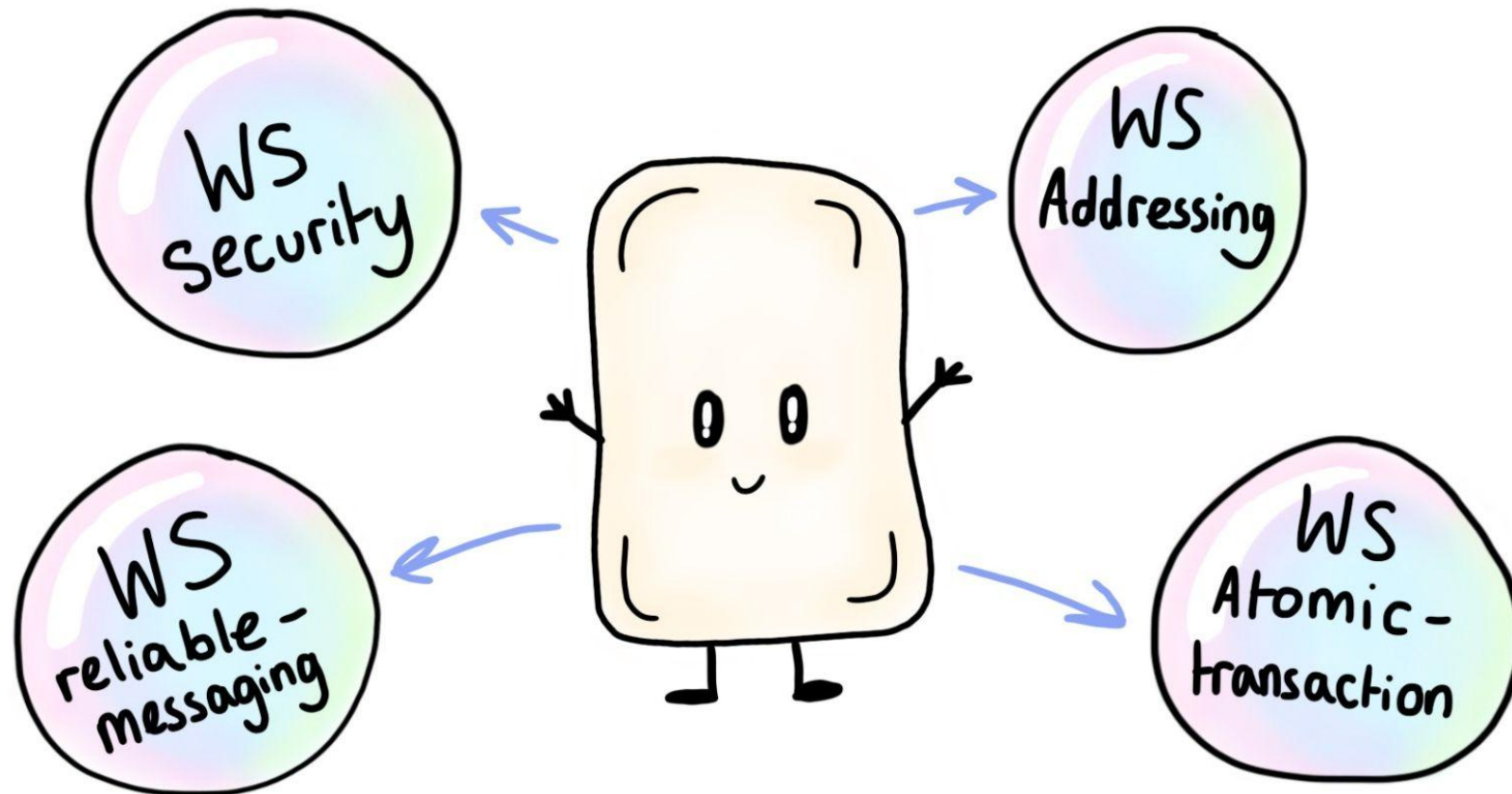
SMTP

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

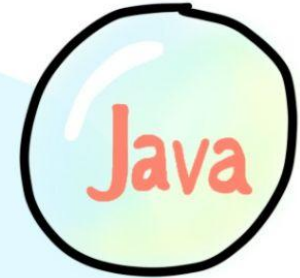
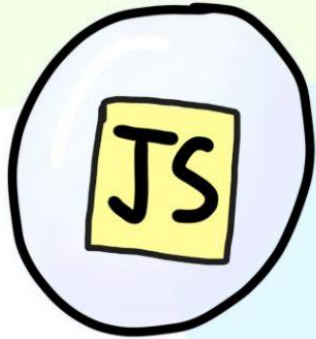
Key Features of SOAP

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.



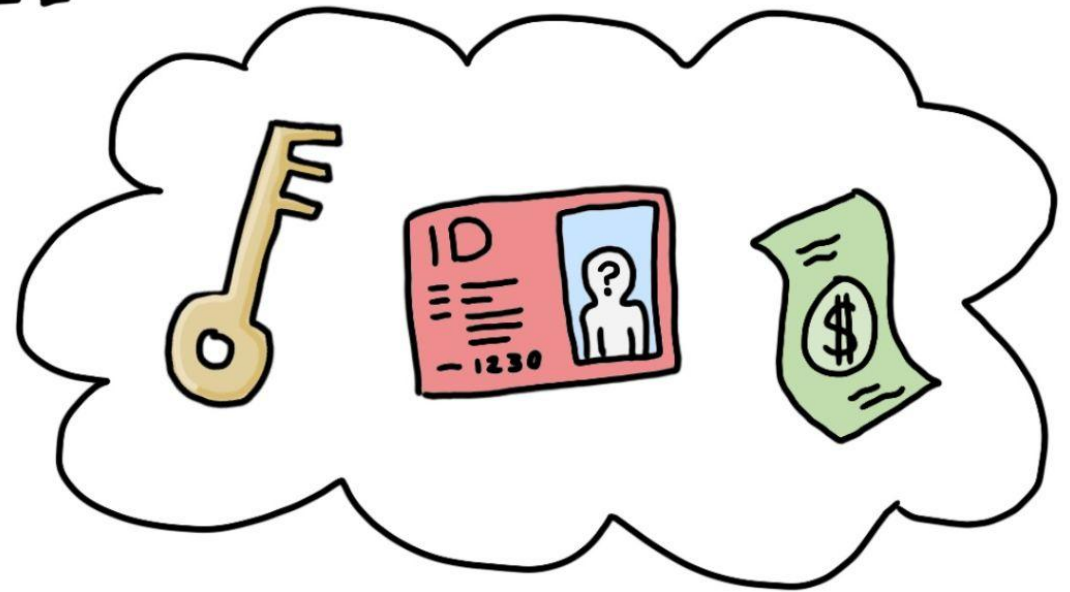
3 Built-in error handling

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

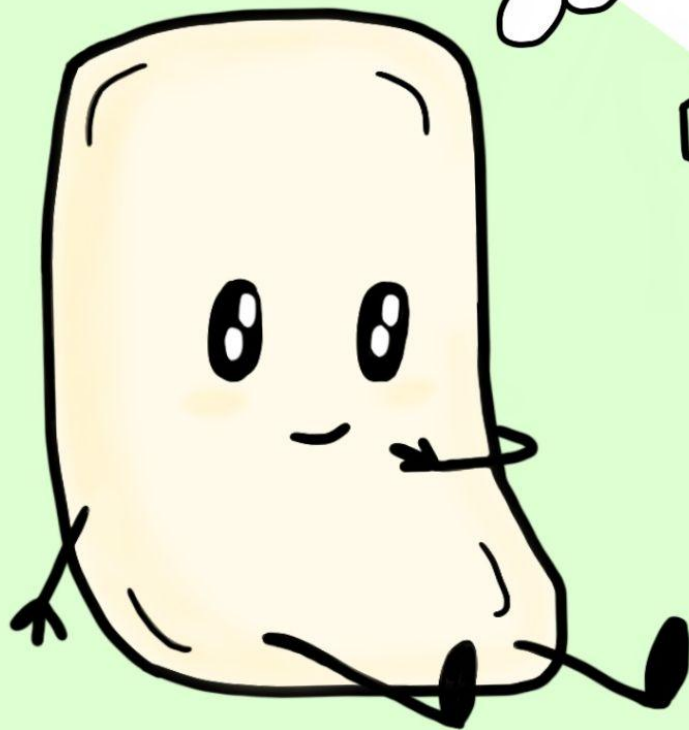


Uses of SOAP API

@Rapid_API 



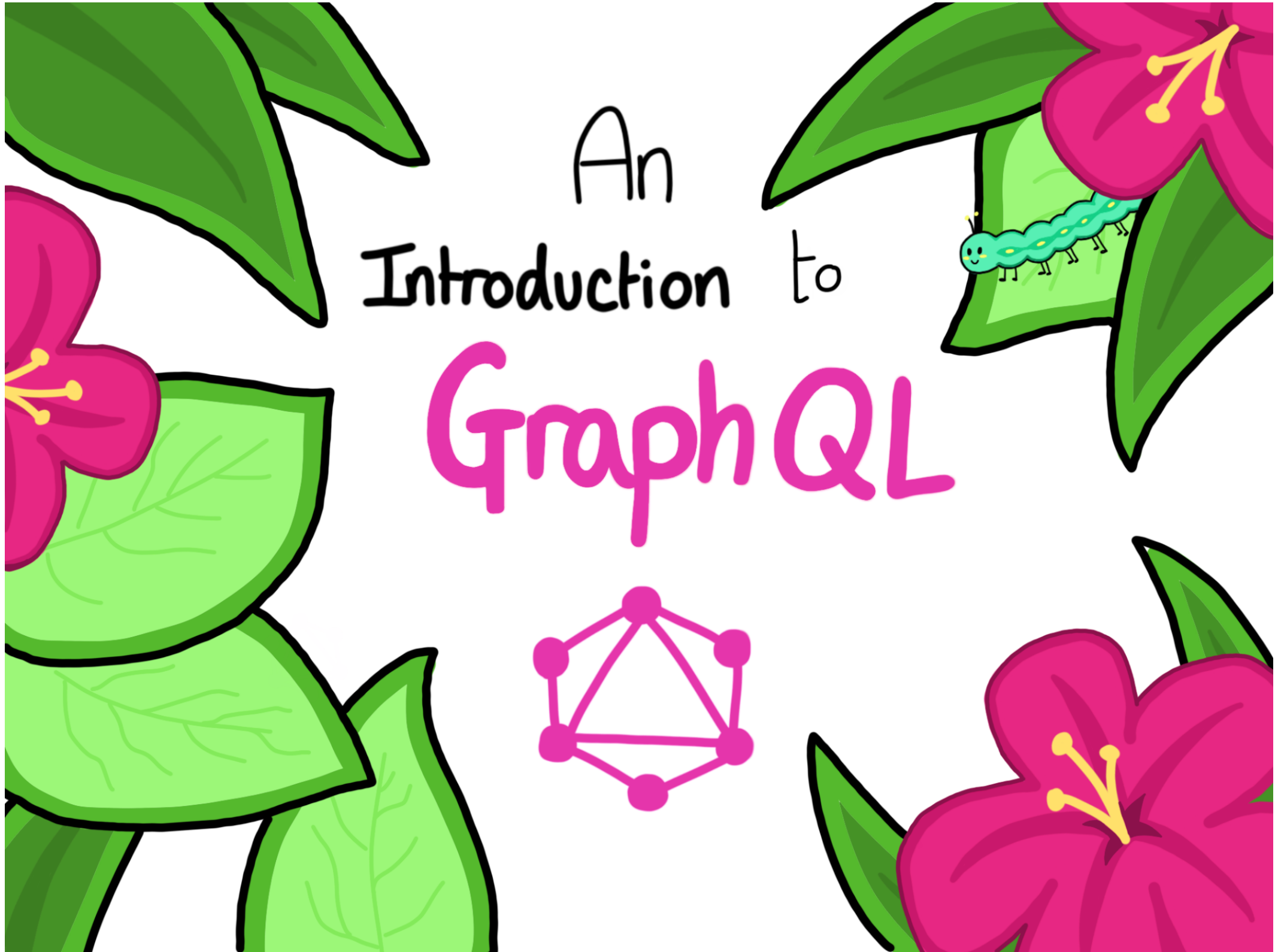
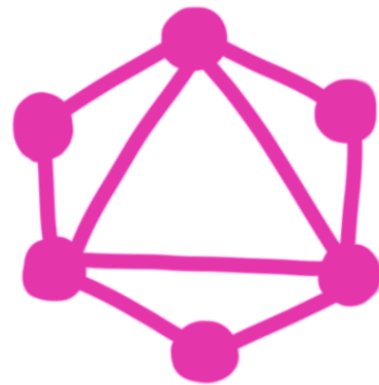
Due to its security extensibility, SOAP is used mainly in enterprise-level web services such as financial services, identity management, and complex transactions.



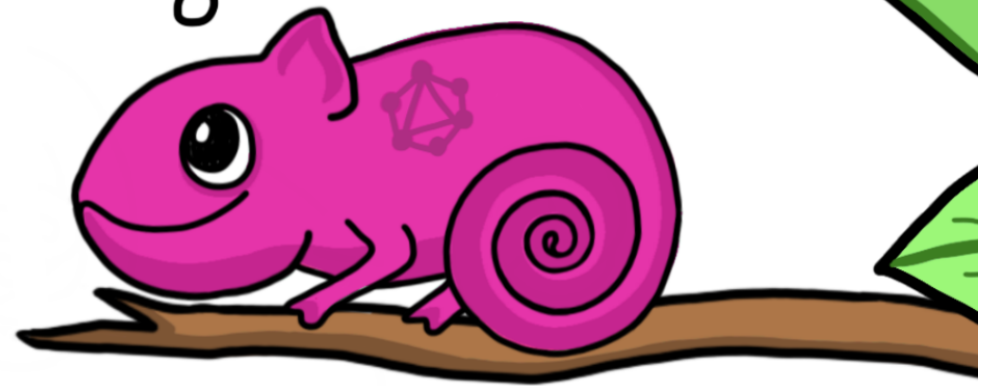
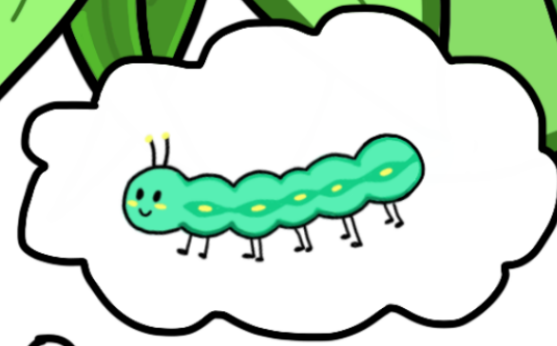
An introduction to GraphQL

An
Introduction to

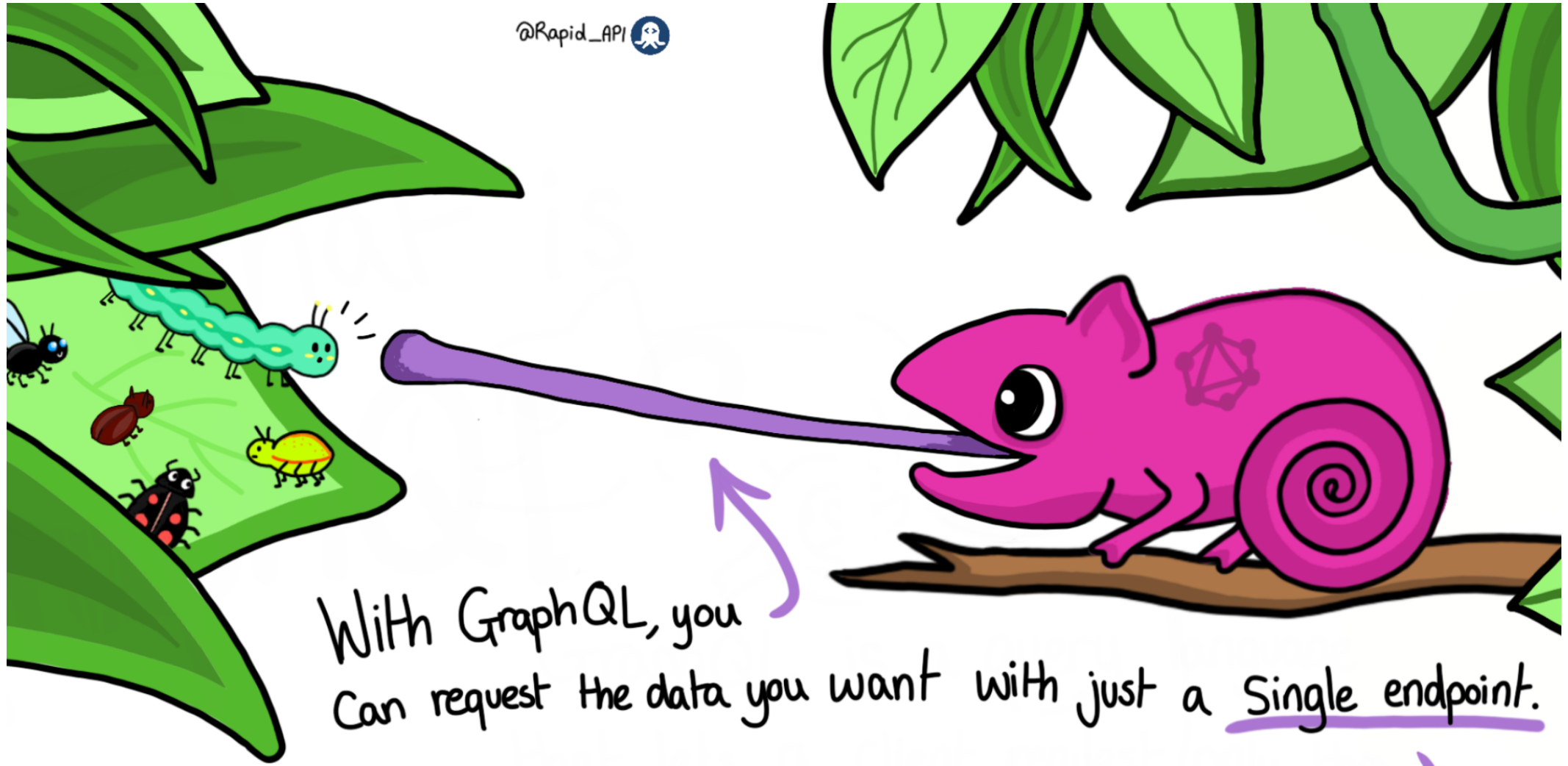
Graph QL



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.

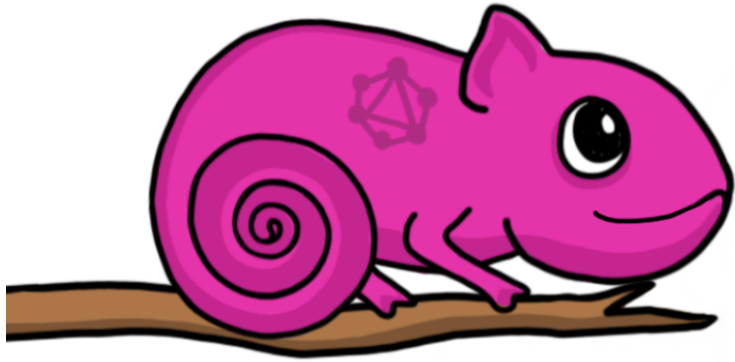
The Schema definition language (SDL)

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

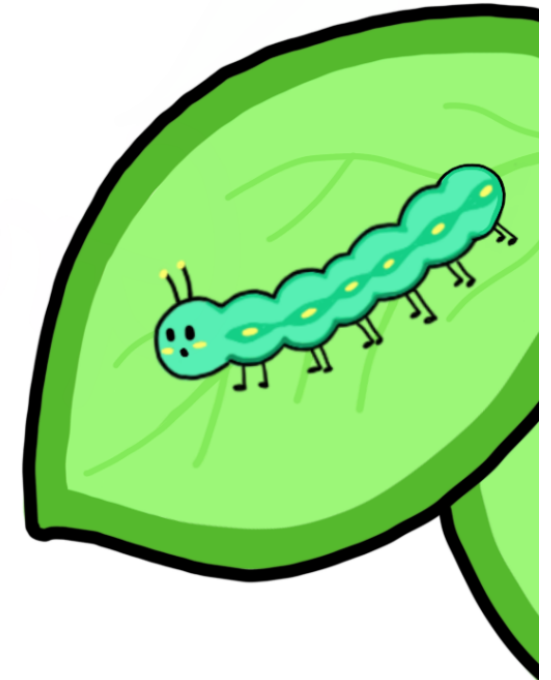
```
type User {  
  name: String!  
  email: String!  
}
```

> 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

Tailored requests.

Reduces bandwidth.

No over-fetching or under-fetching of data.

Suitable for smaller networks and microservices.

Fetches data with a single API call.

API growth without versioning.



An introduction to Webhooks

An Introduction to



Webhooks

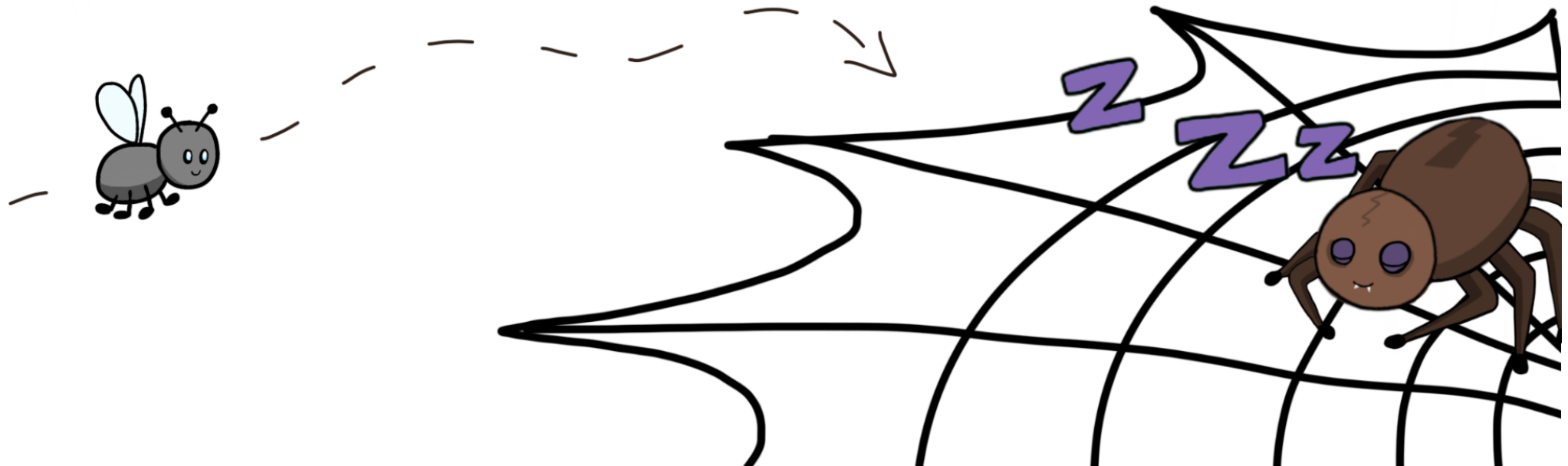


What are Webhooks?

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.

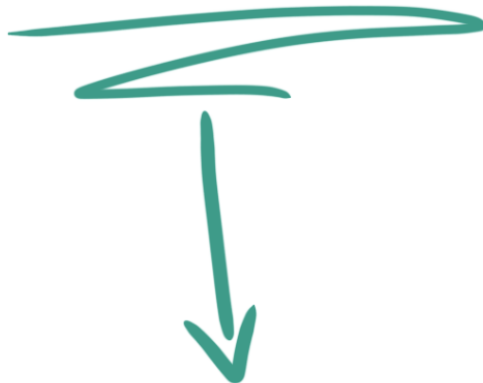
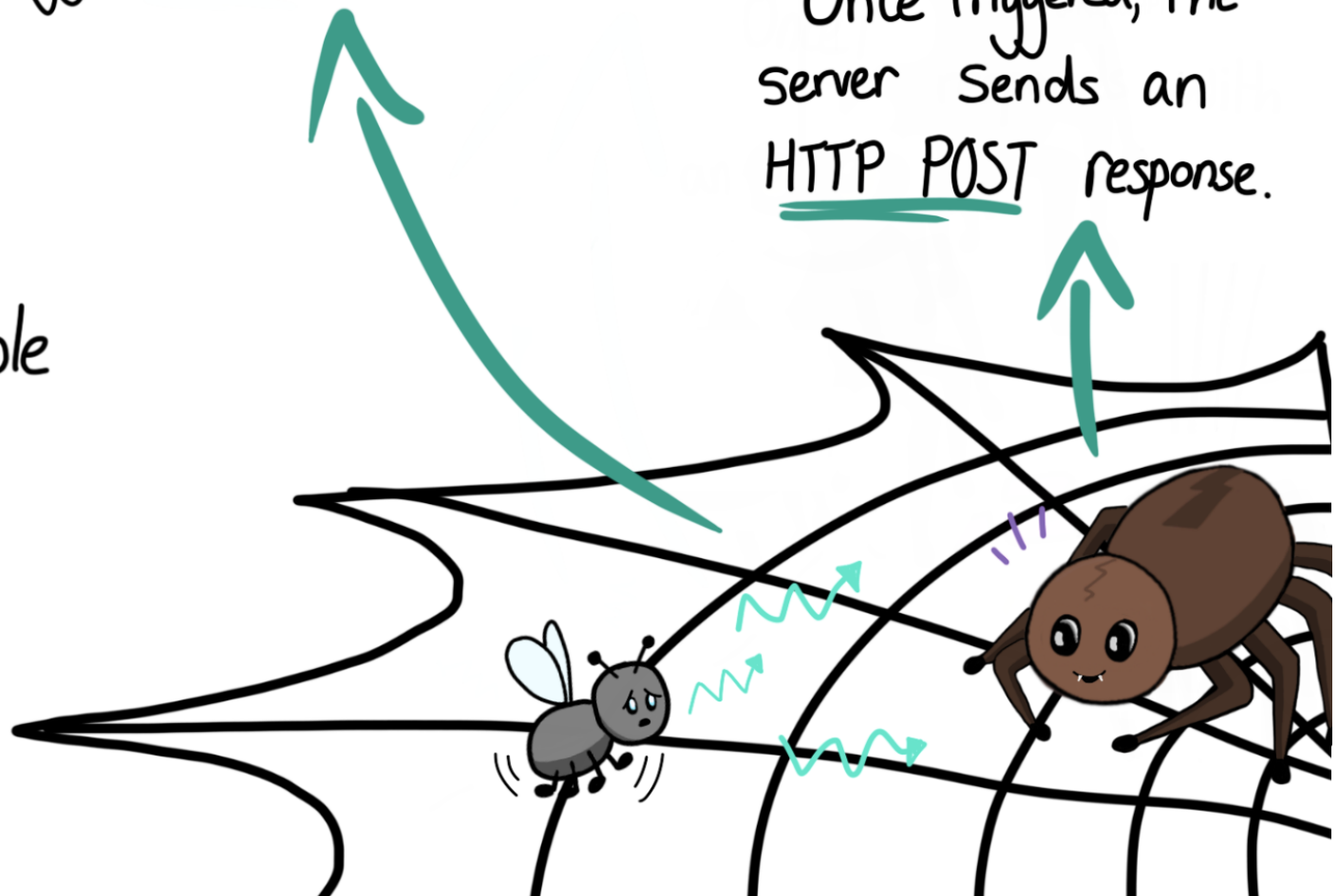


How do Webhooks Work?

A set event or action triggers webhooks.

Once triggered, the server sends an HTTP POST response.

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



1

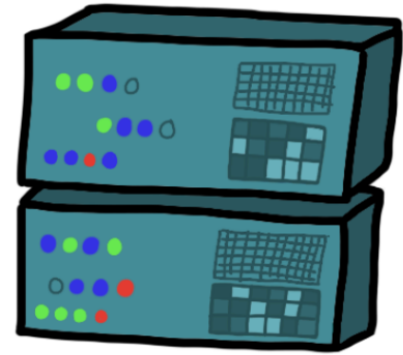


User visits an online store and buys some items, which confirms an order.

The online store has a webhook setup that listens for any 'order/confirmed' events. This is the trigger.

2

Store server



The Store server sends user and order details through a webhook URL via an HTTP POST request.

HTTP POST

3

Third-party app
(E.g. Email sending service)



Client receives Confirmation Email.



4

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

Why use Webhooks?



Ecommerce

Payment Software



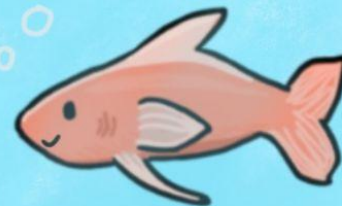
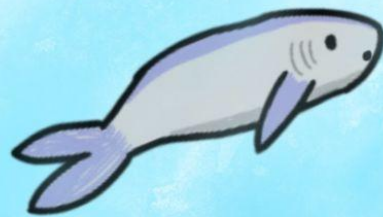
Social media



REST vs GraphQL

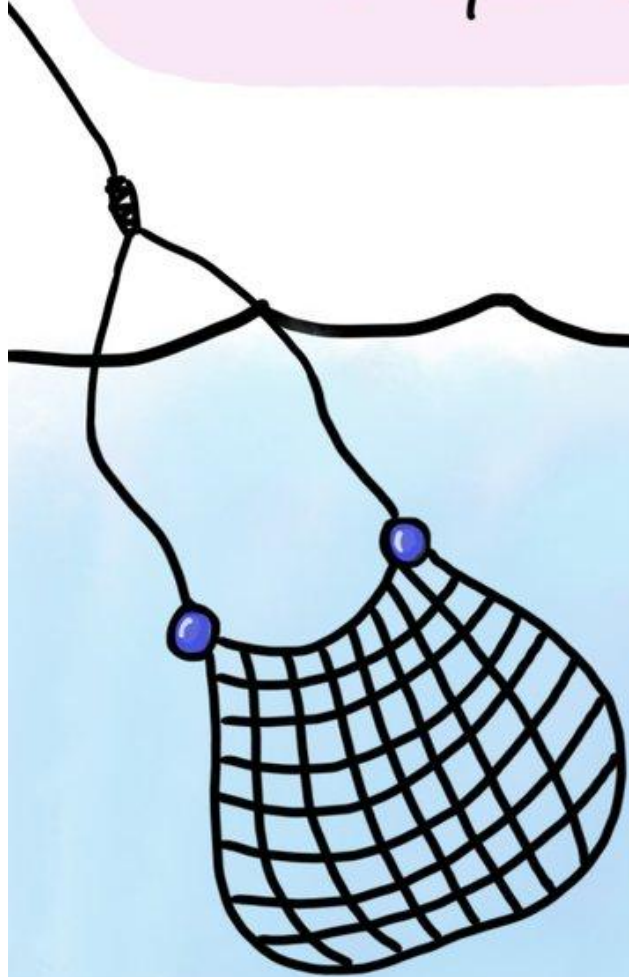
What's the difference between

REST and GraphQL?



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

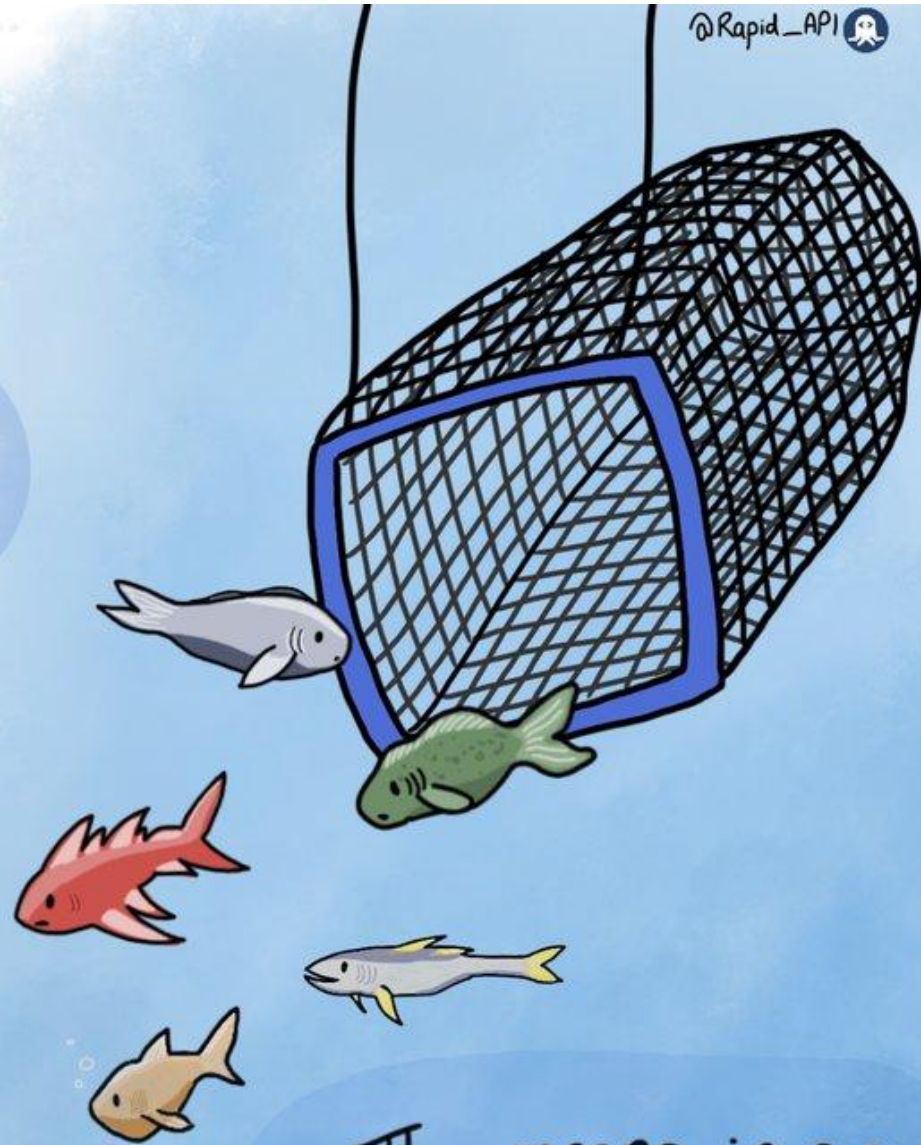
REST is an architectural style that conforms to several constraints. REST APIs are flexible, scalable, and easy to use, making them popular in web applications.



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.

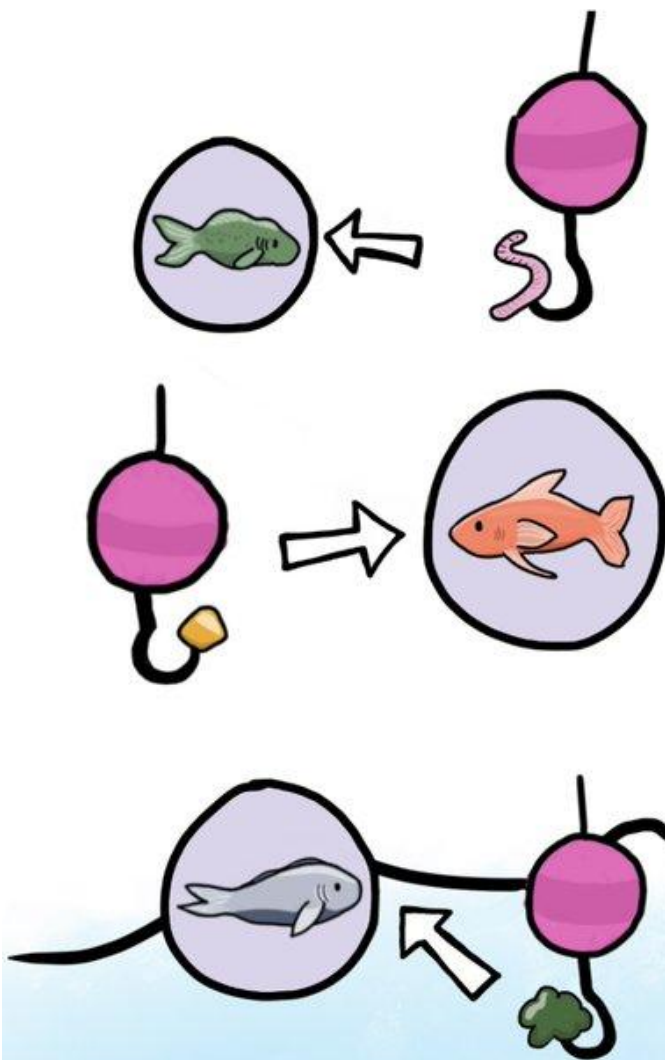


This means in some cases you might be over-fetching data, which means returning some data you won't need.

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.



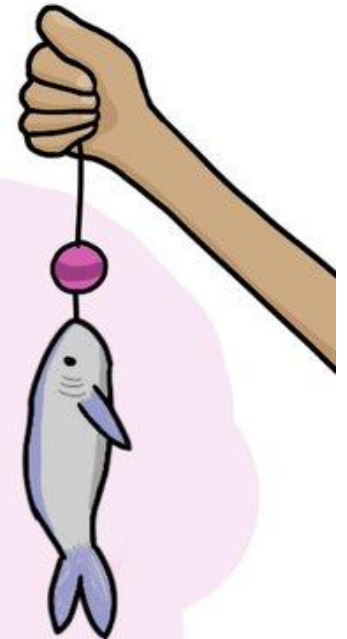


REST APIs are:

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

GraphQL is:

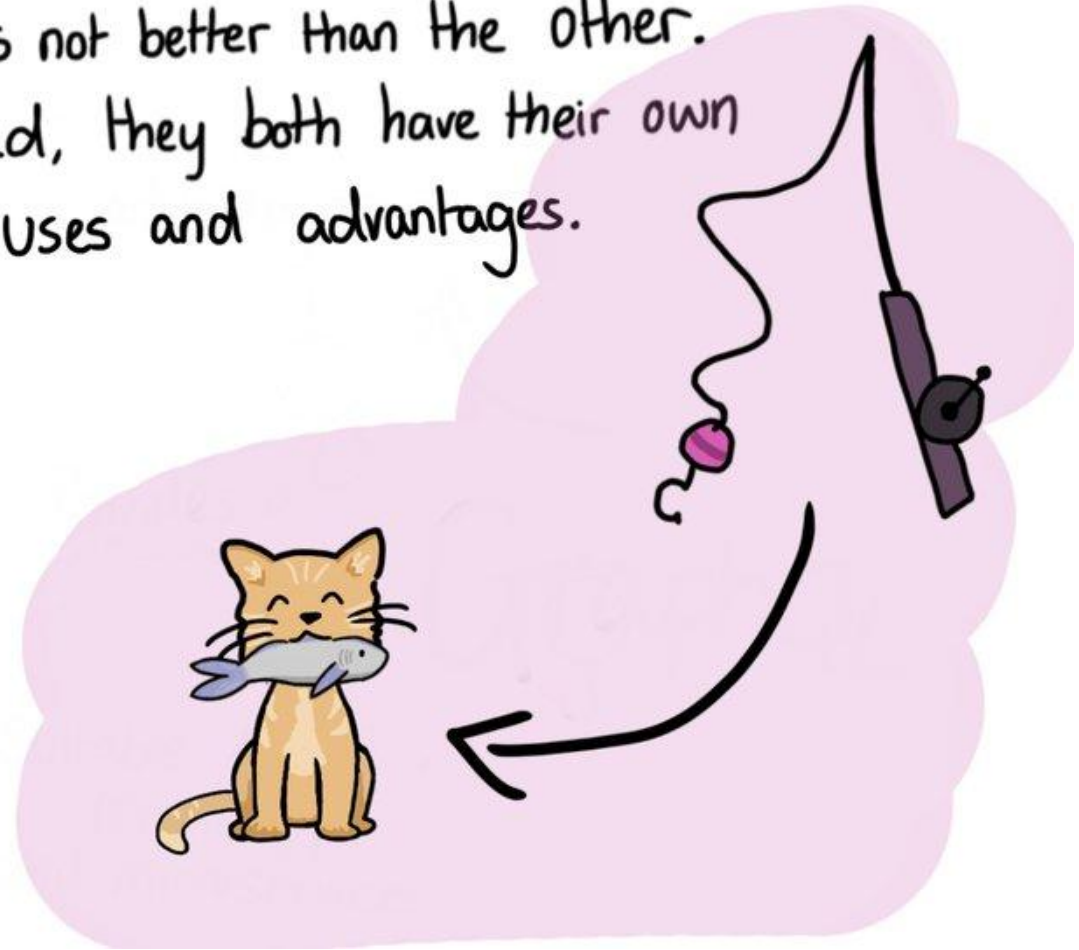
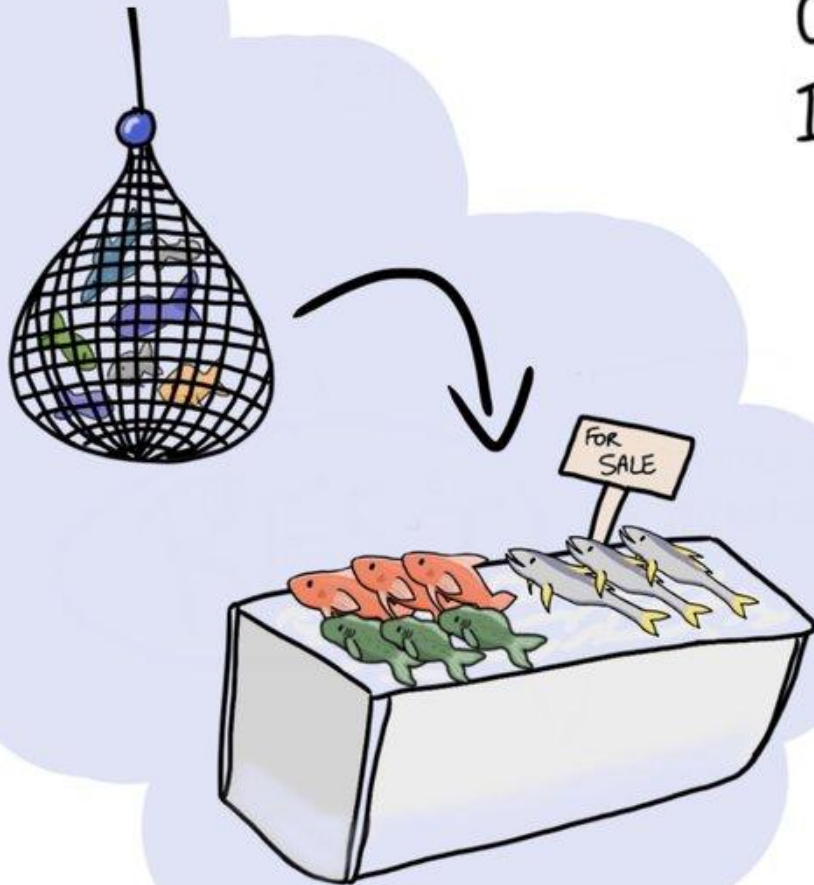
- Centered around one API endpoint
- Tailored to your data requirements
- Requires less bandwidth



Which should you use?

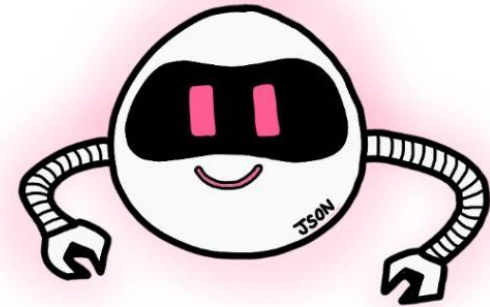
It depends entirely on the requirements of your application.

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

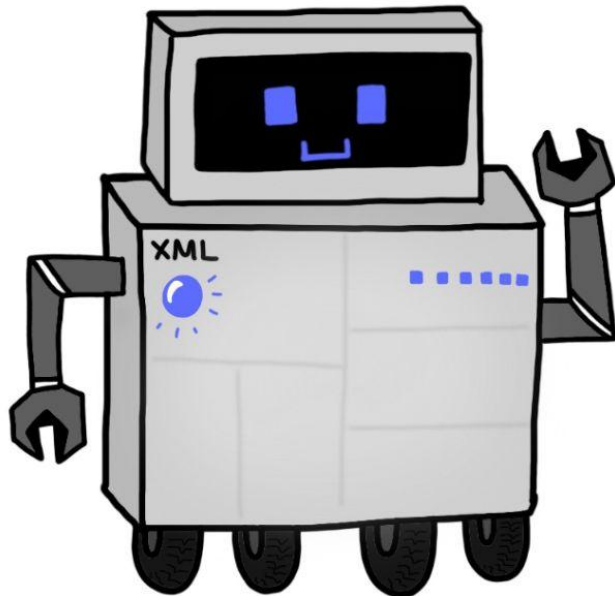


JSON vs XML

JSON

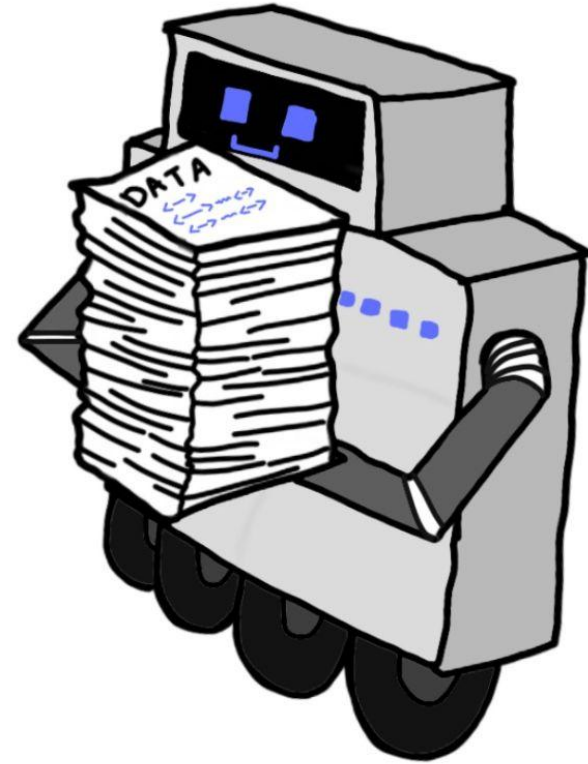
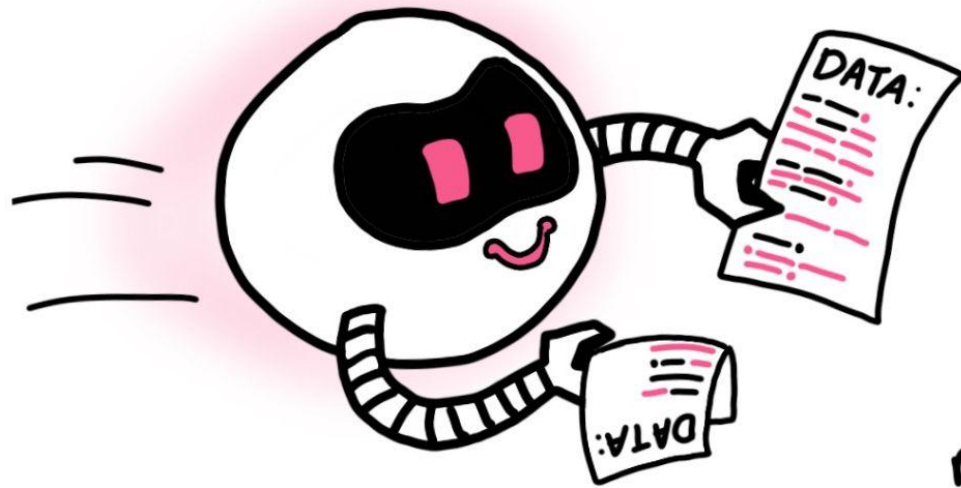


VS



XML

Both **JSON** and **XML** are data formats used to send and receive data from web servers.

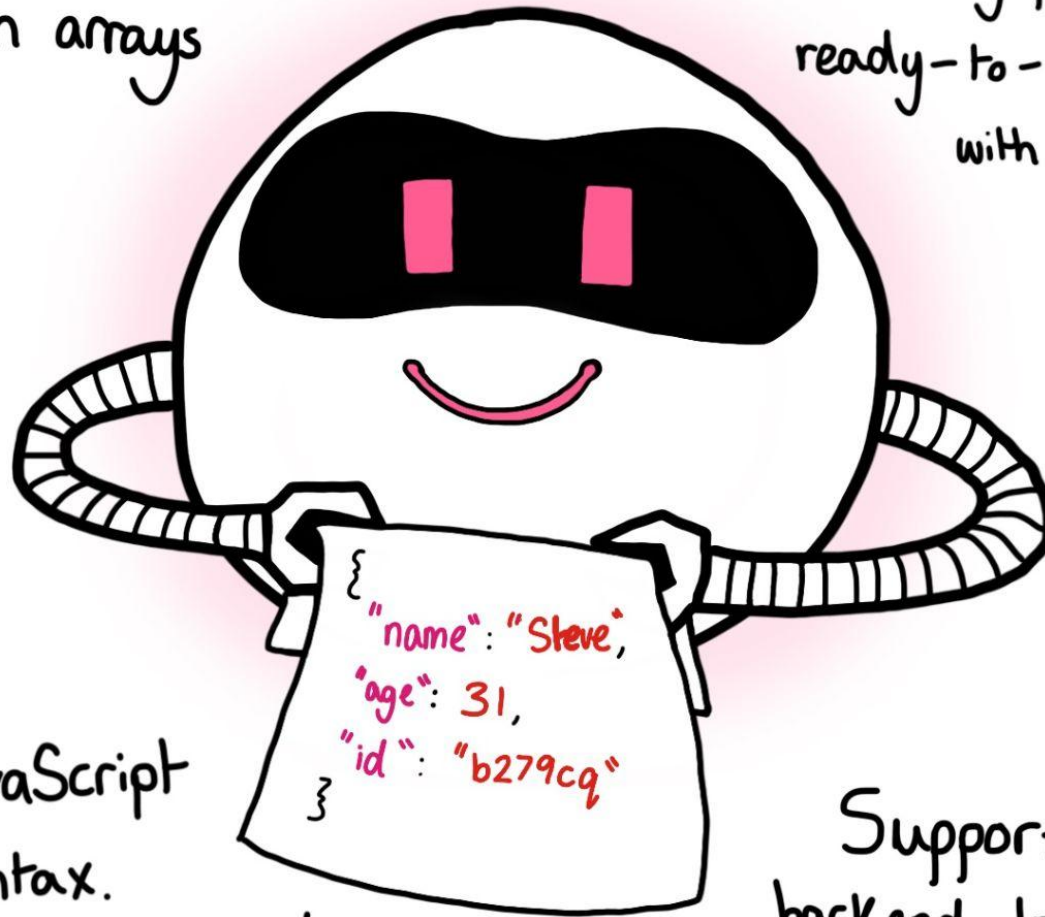


Both play an important role in organizing data into a readable format in many different languages and APIs.

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.



Easy to read and write.

Based on JavaScript object literal syntax.

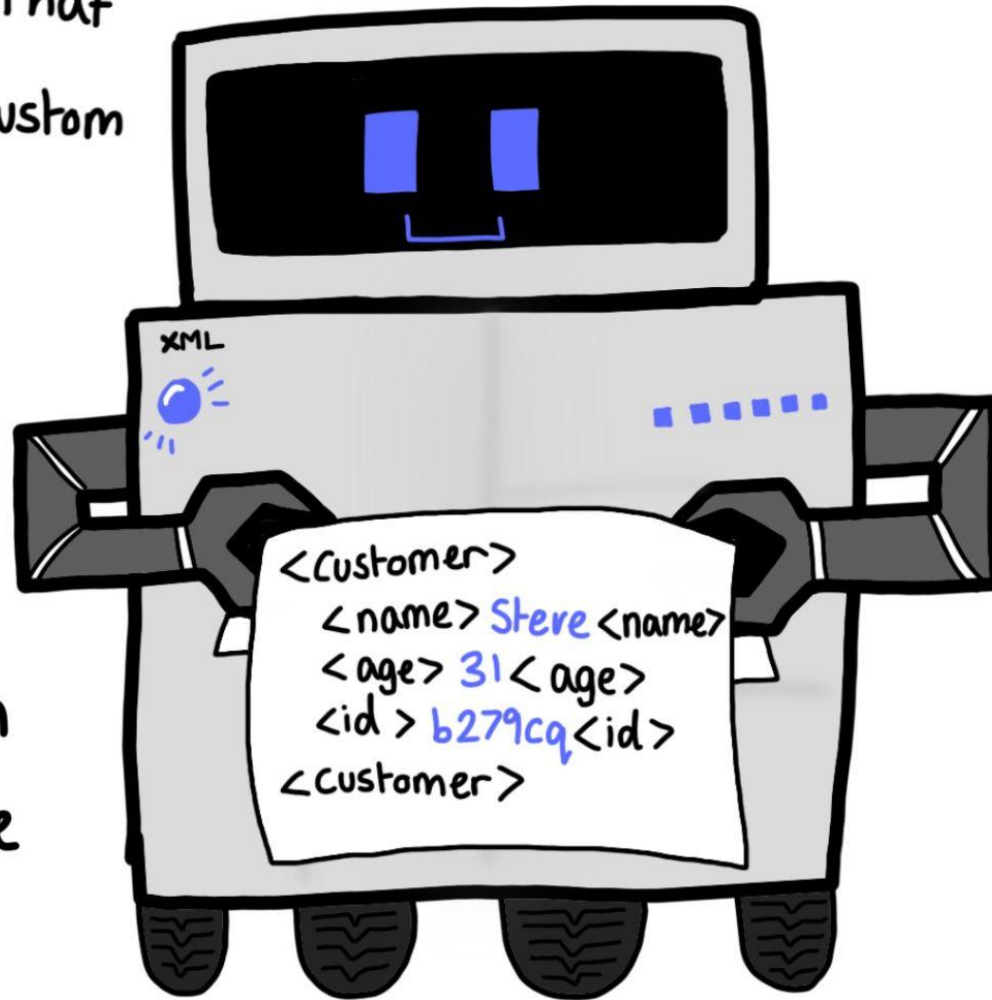
Supported by most backend technologies and modern programming languages.

XML: "Extensive Markup Language"

Markup language that allows creating custom user defined tags.

Complex data structure that must be parsed.

Manages data in a 'tree' structure hierarchy.



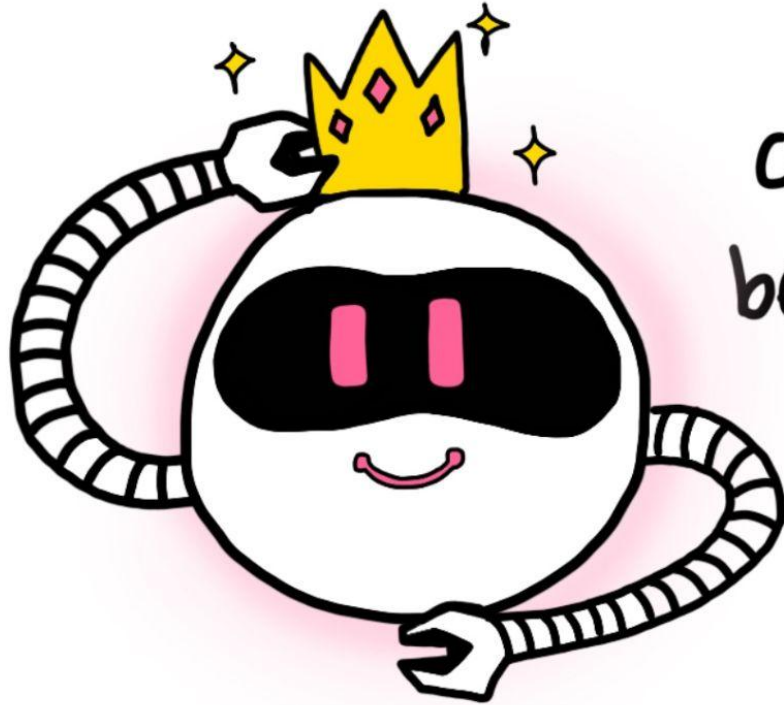
Similarities and differences

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:

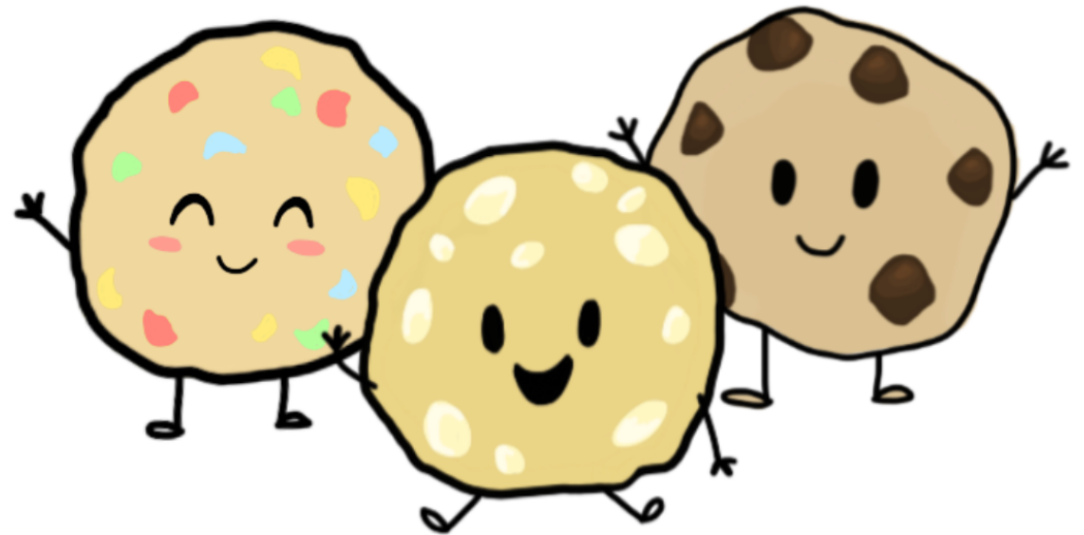
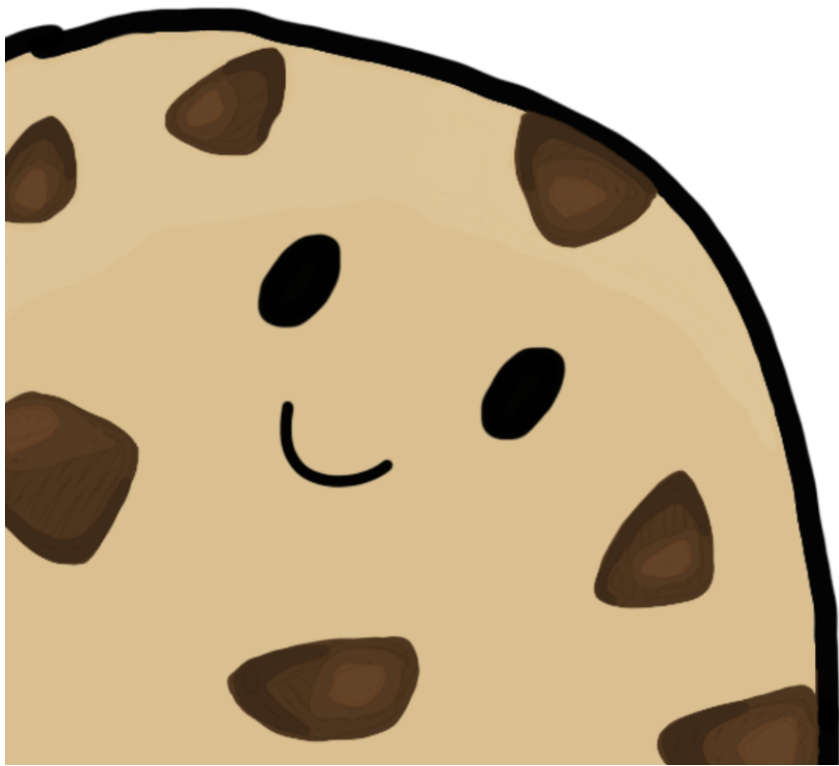
- JSON object has a type, XML
is typeless
- XML has display capabilities unlike
JSON
- JSON is less secure than XML
- XML is much more complex and
slow to parse



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:

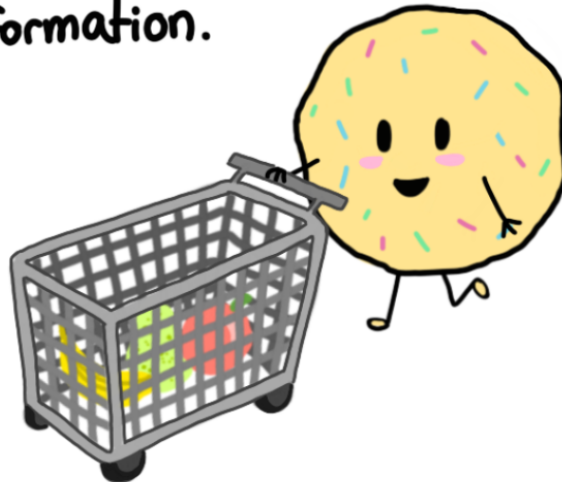
Tracking

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.

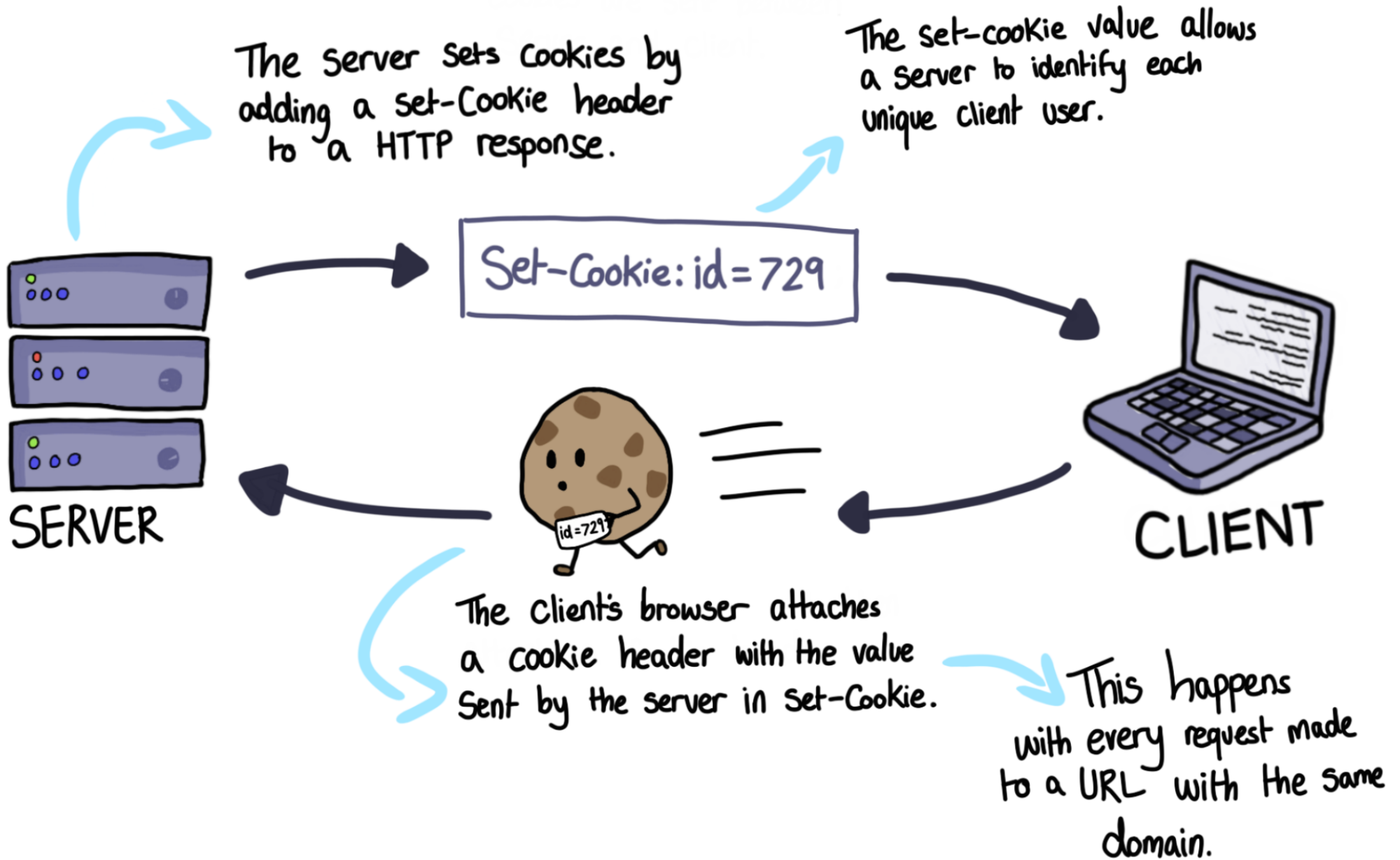


Personalization

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

or

Session Cookies

↓

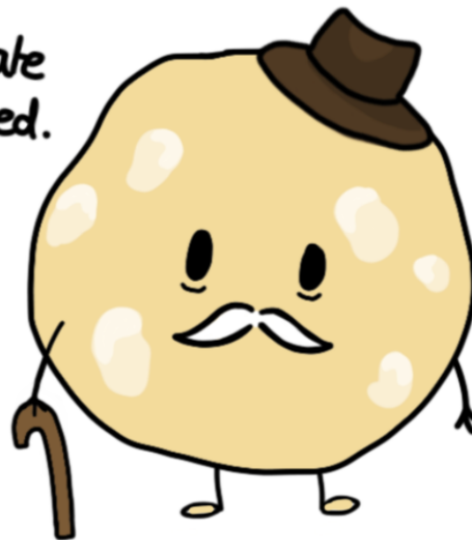
This type uses the Expires attribute to set a specified date that the cookie will be deleted.

Set-Cookie:

id=8125;

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

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



↓

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

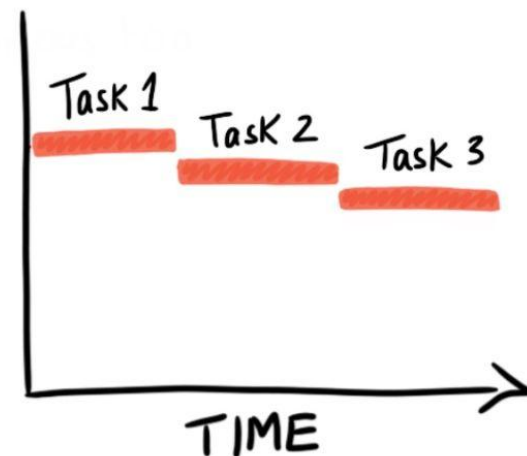
Synchronous and Asynchronous Programming



Synchronicity refers to the way code is executed.

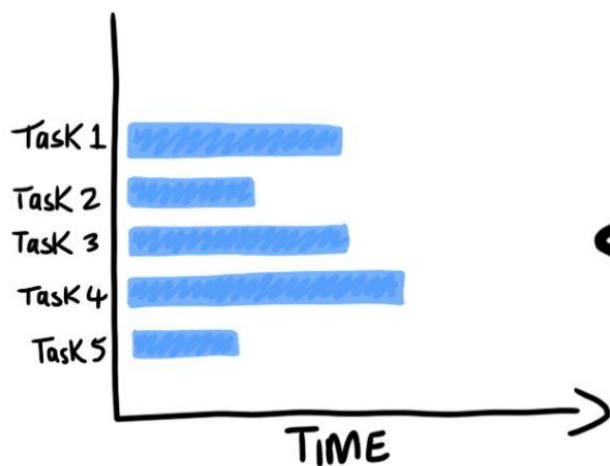
Synchronous

Synchronous calls are blocking.
This means all other code execution is halted until the call is returned.



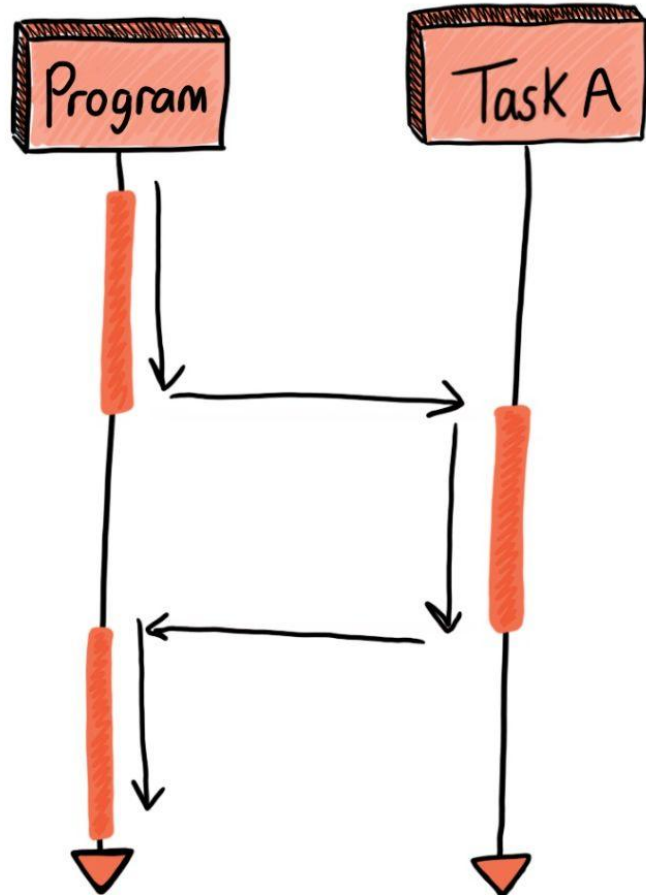
Asynchronous

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



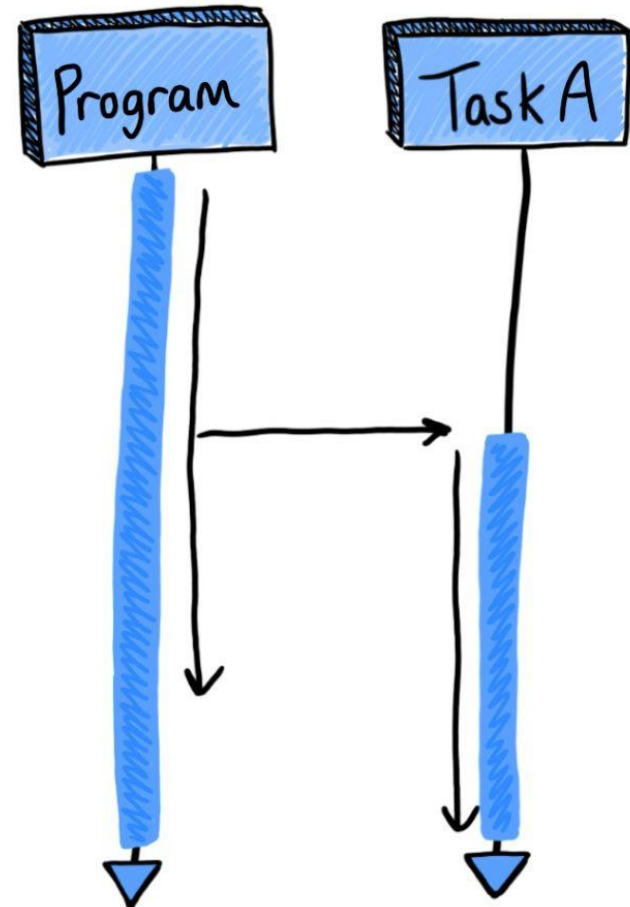
Synchronous

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



Asynchronous

Independent tasks carried out in parallel.



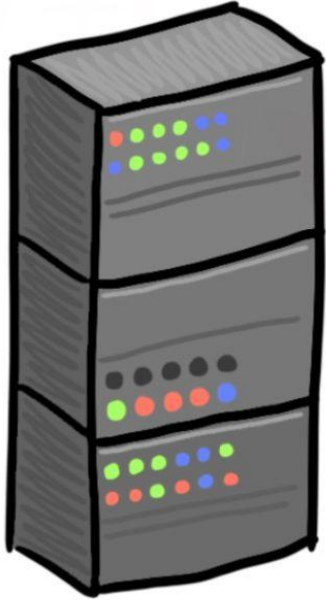
Asynchronous tasks mean the browser can maintain functionality rather than get held up waiting on a request.



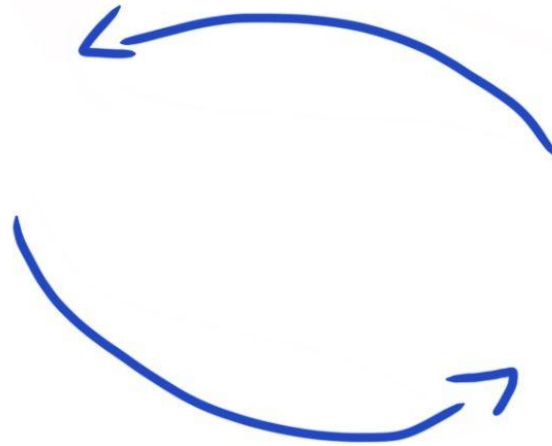
Async programming can lead to greater performance and user experience.

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

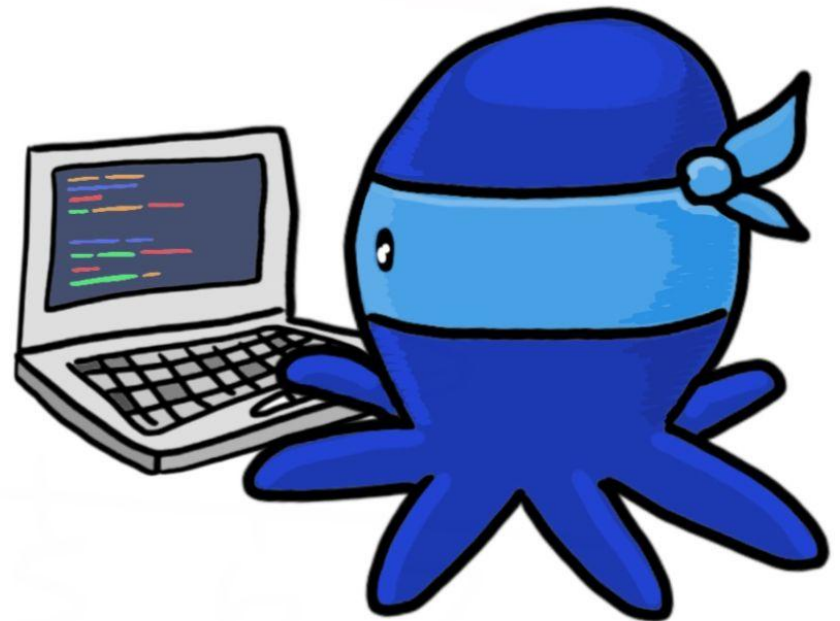
Synchronous programming is great for programs that react to environmental input.



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

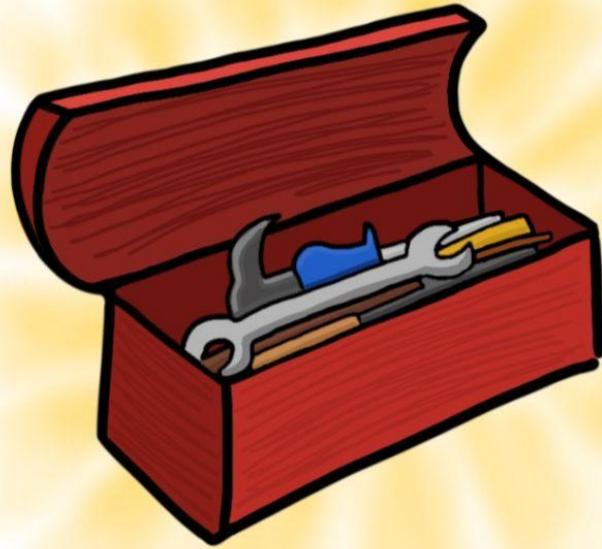
What's the difference
between



an **SDK** and

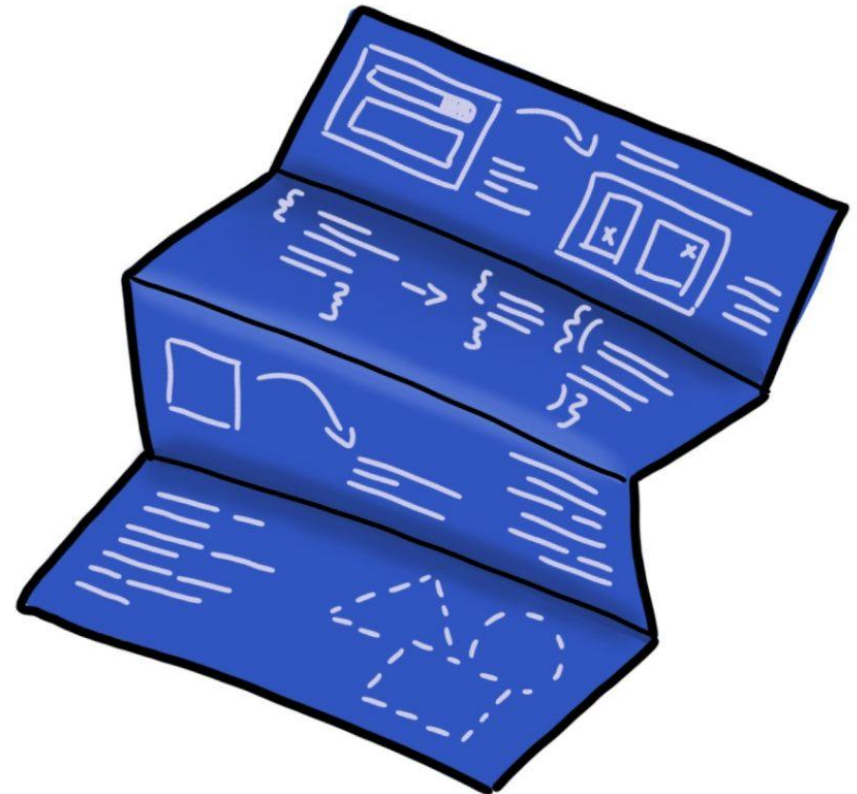


an **API** ?



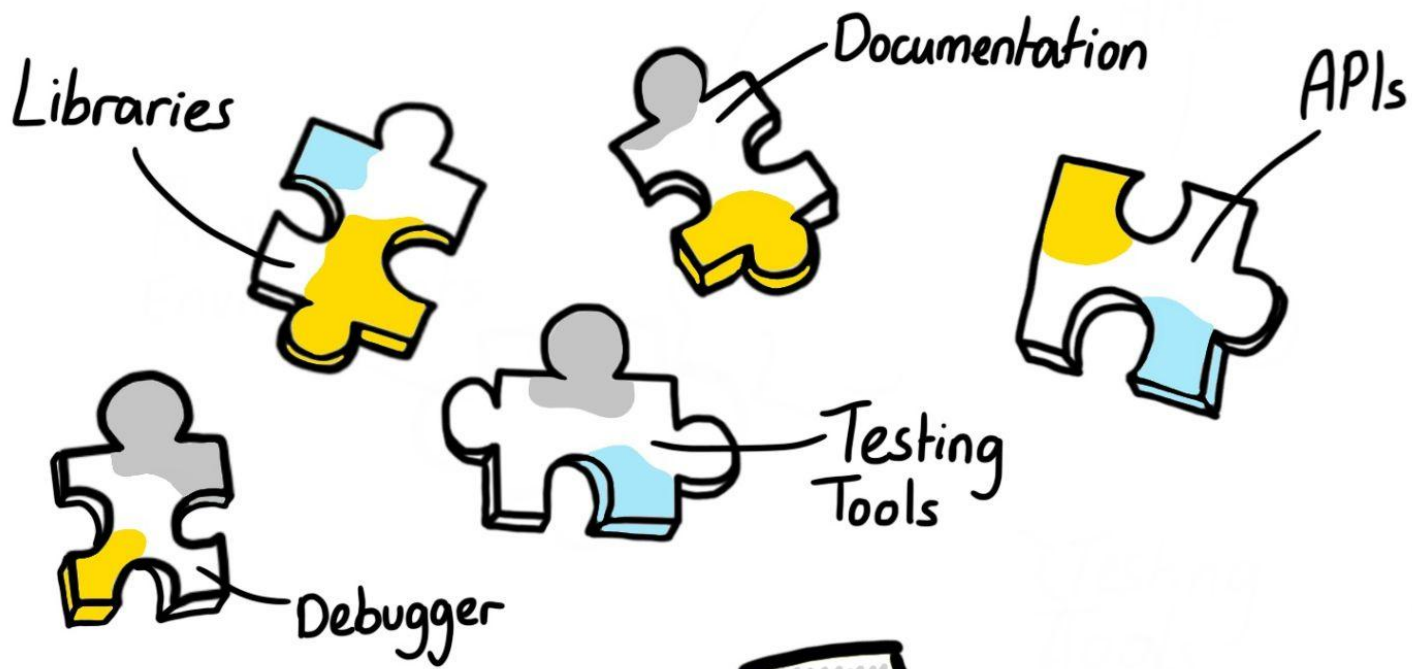
SDK Stands For
Software development Kit.

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

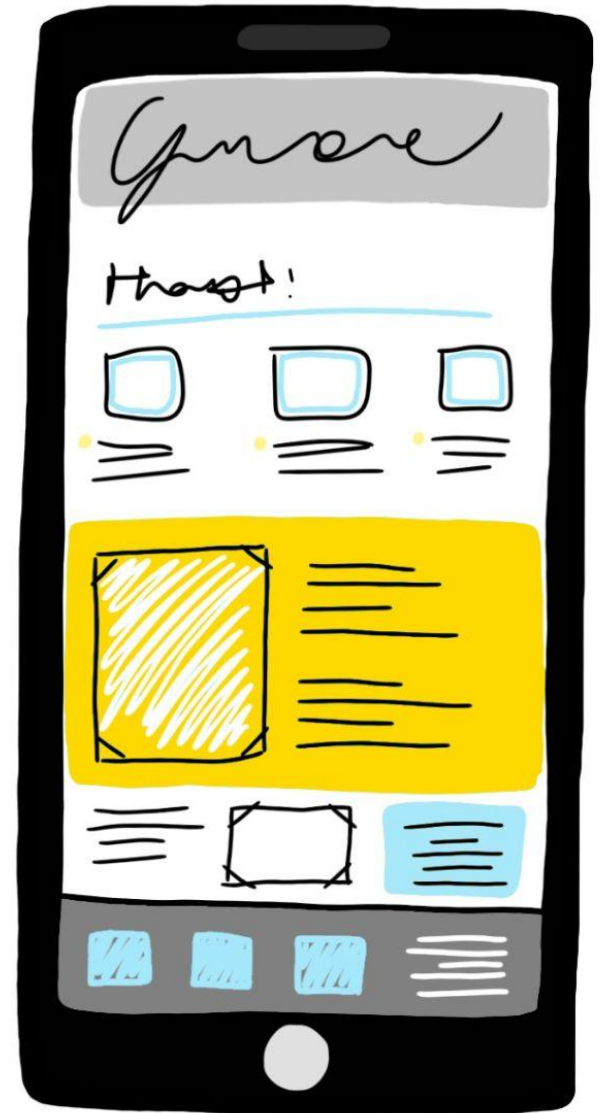
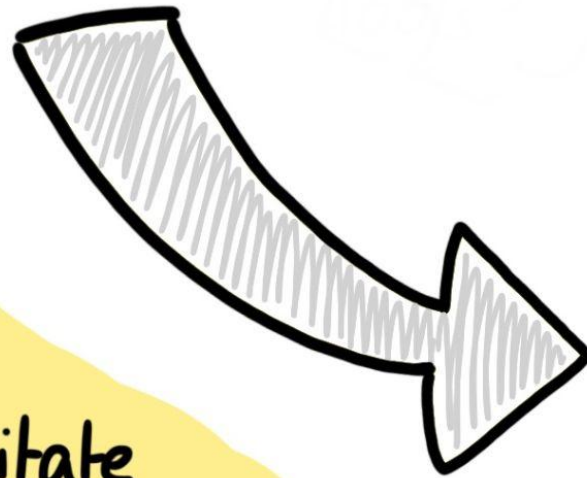


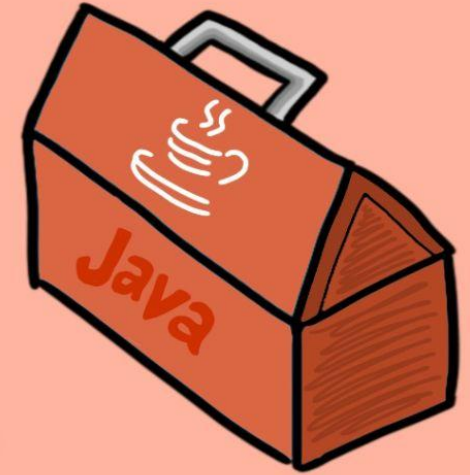
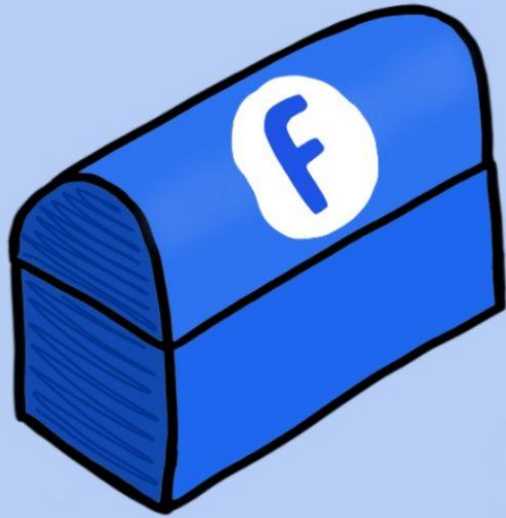
APIs are often another
'tool' included in an SDK.





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

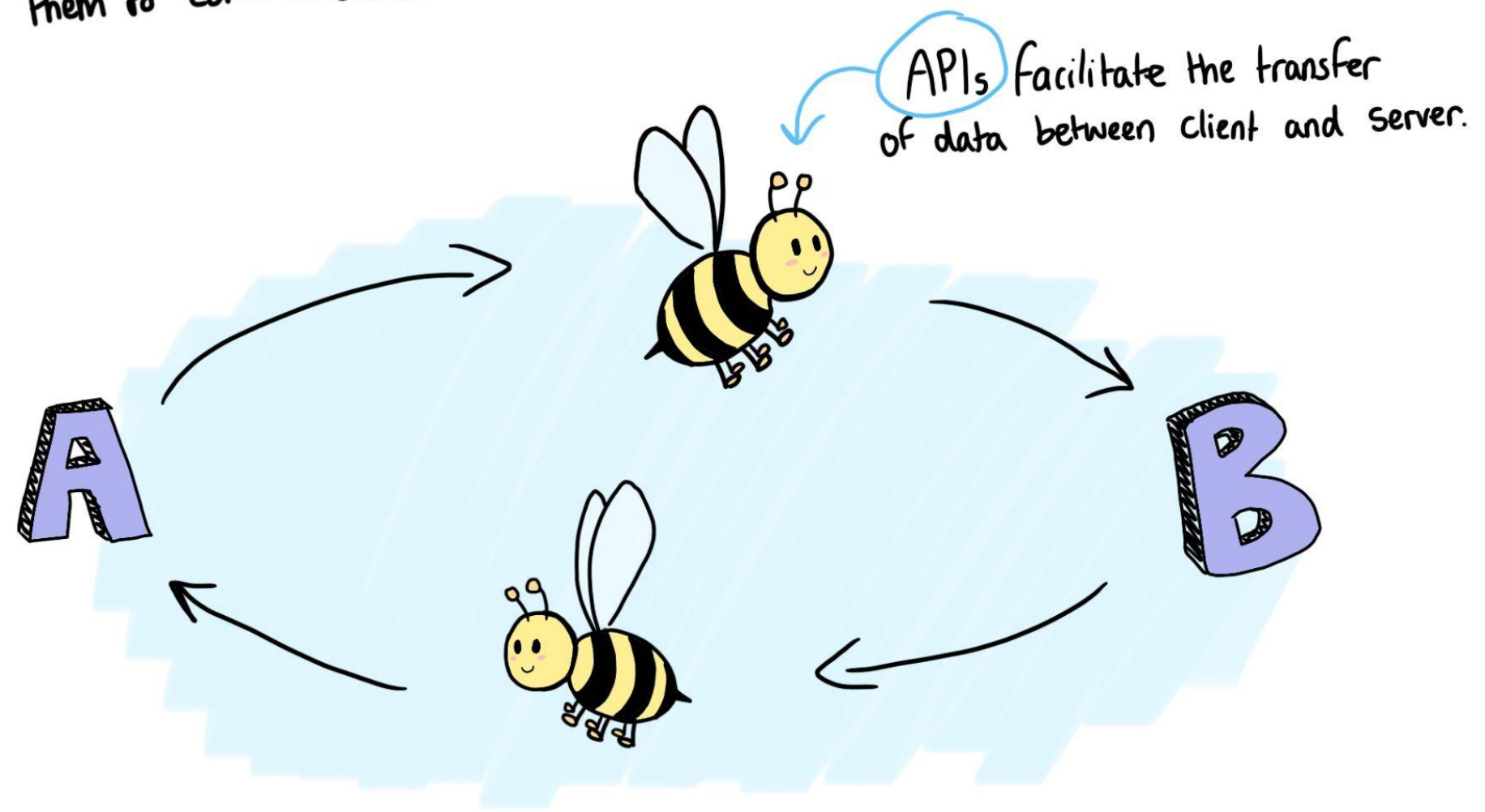


VS



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.

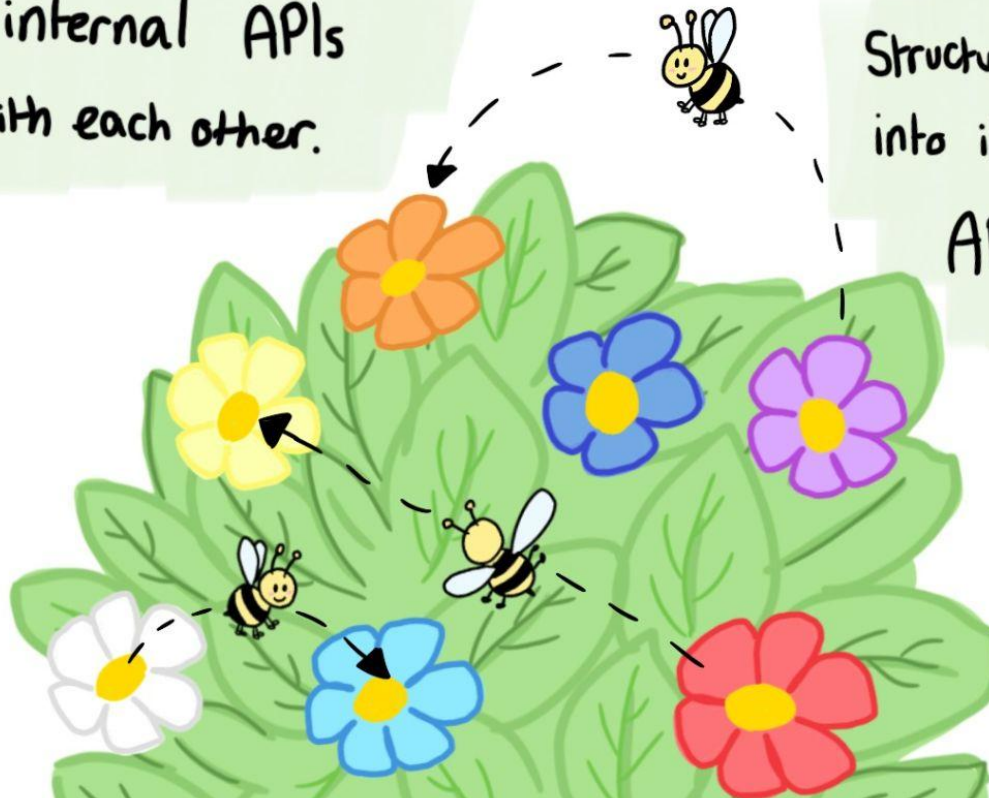
Each service is independently deployable, and has a purpose of its own.



How do they work together?

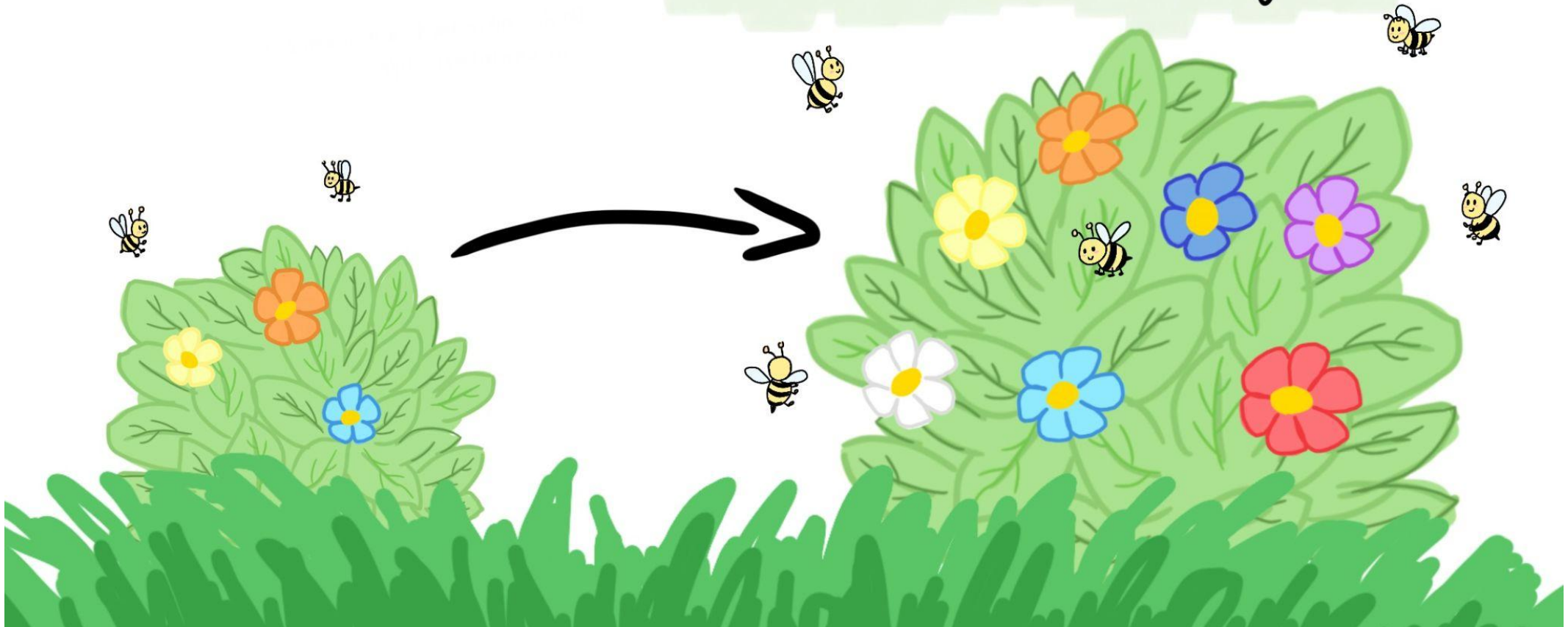
Each individual service within a Microservice uses internal APIs to communicate with each other.

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

How

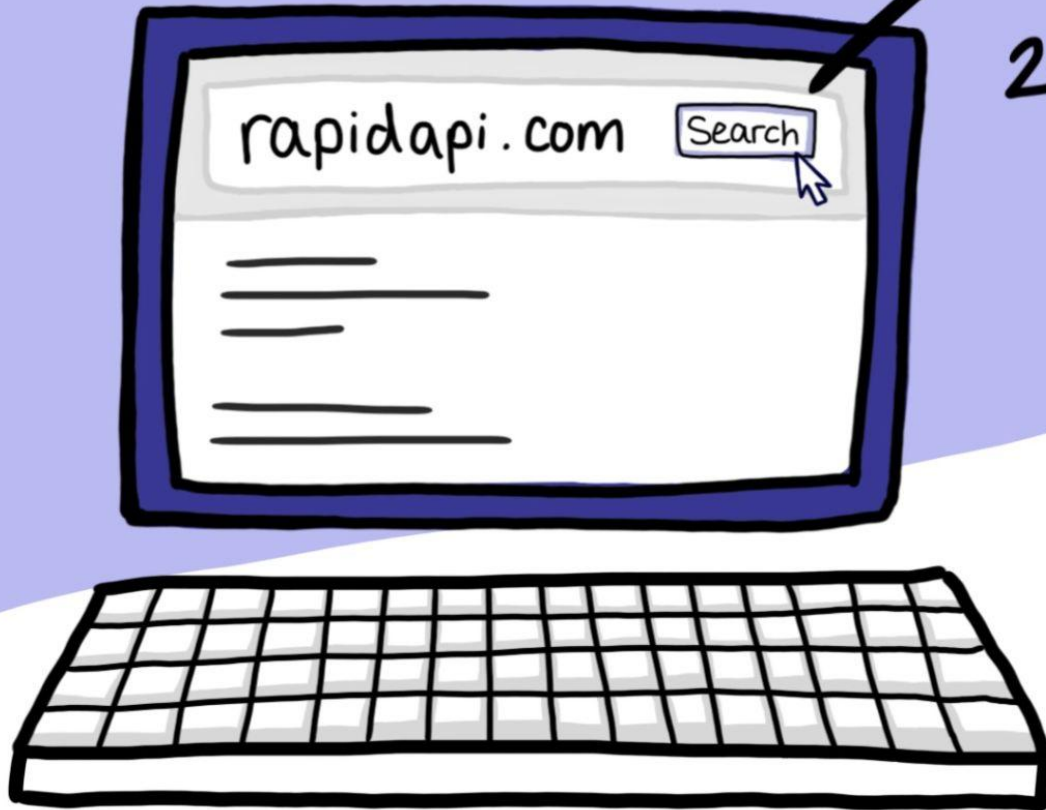
DNS

works

@Rapid_API 

DNS stands for Domain Name System

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



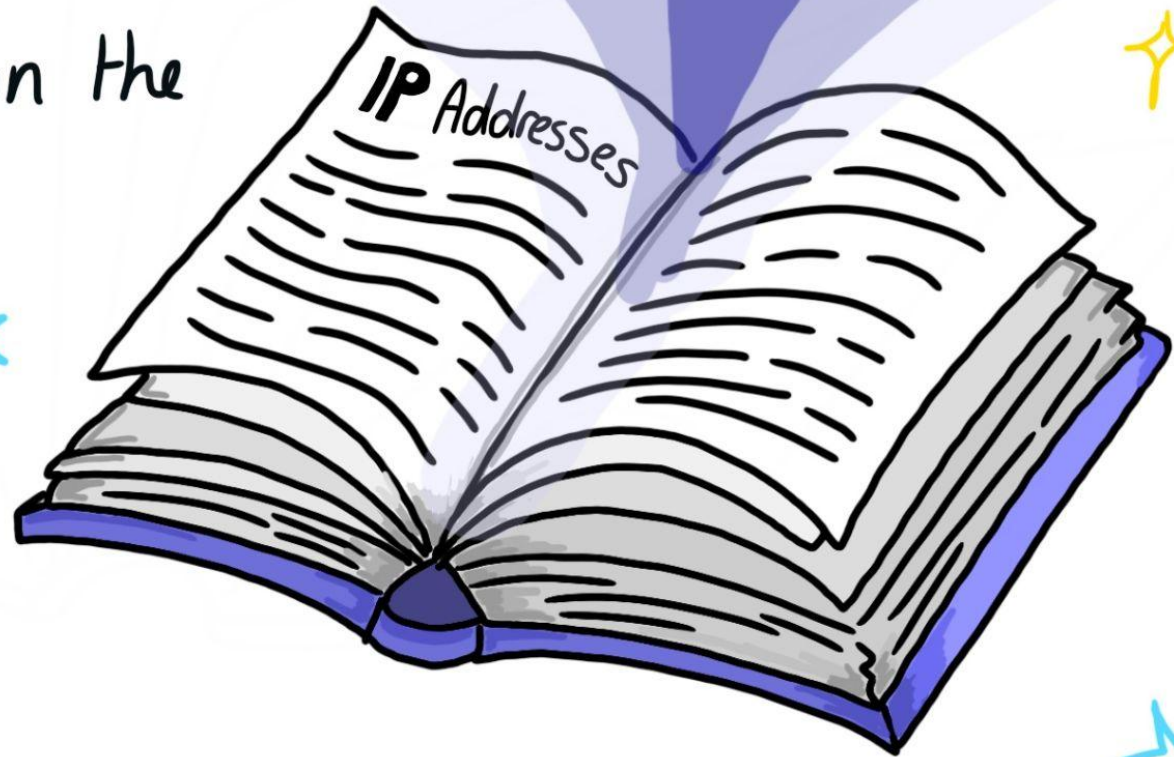
rapidapi.com



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:

Step 1

DNS Cache

User searches for a site



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

Your browser



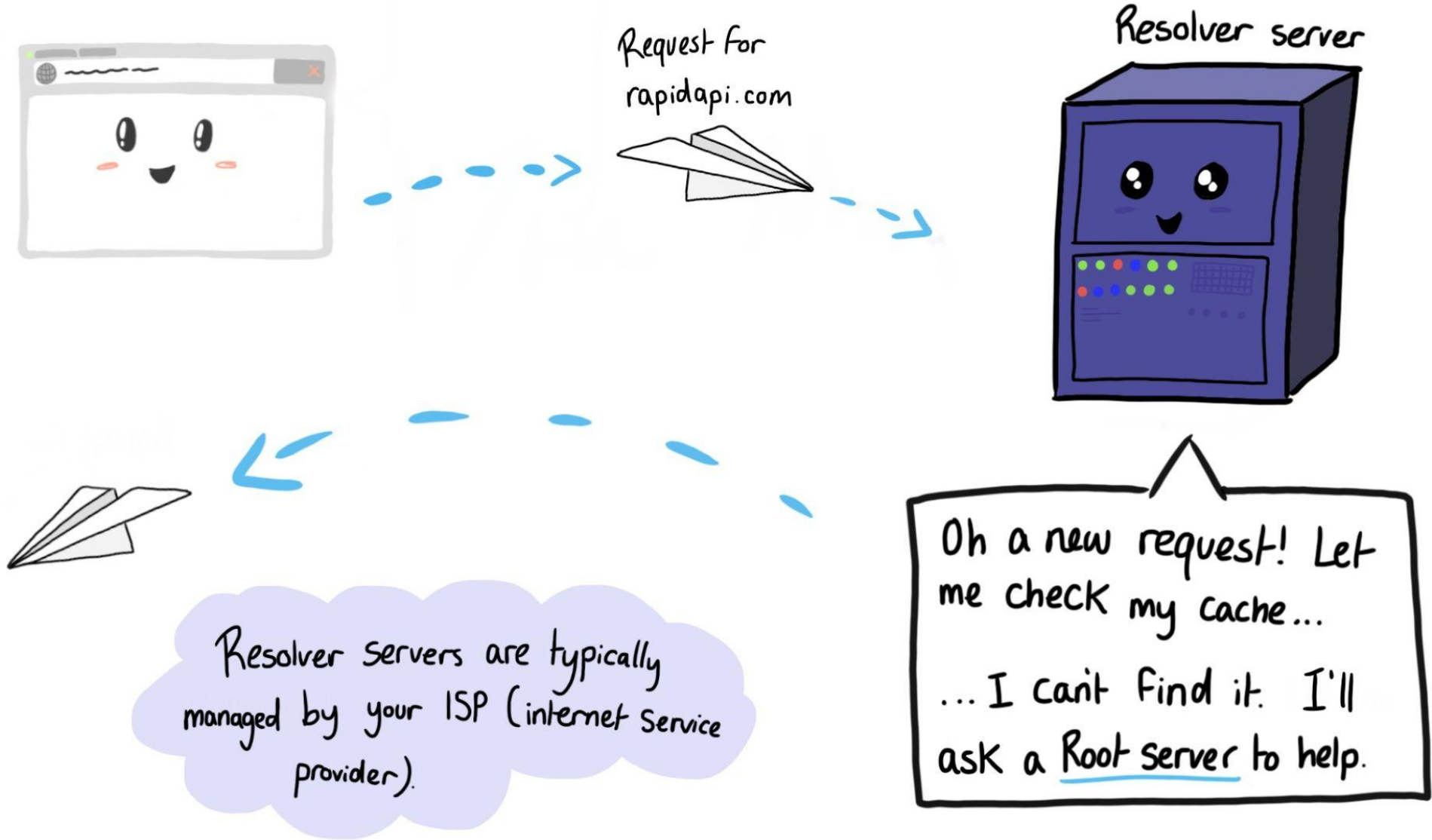
Nope! I don't
know this one.
Let me send
it to a
Resolver.



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

Step 2

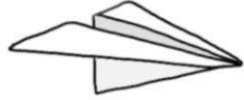
Resolver Server



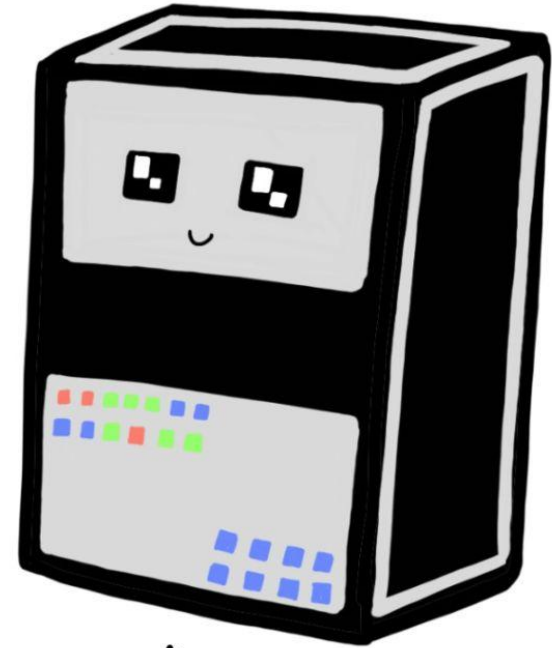
Step 3

Root Server

rapidapi.com

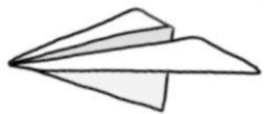


Root Server



Root servers sit at the top of the DNS hierarchy. They redirect Resolver servers to another type of server called TLD servers.

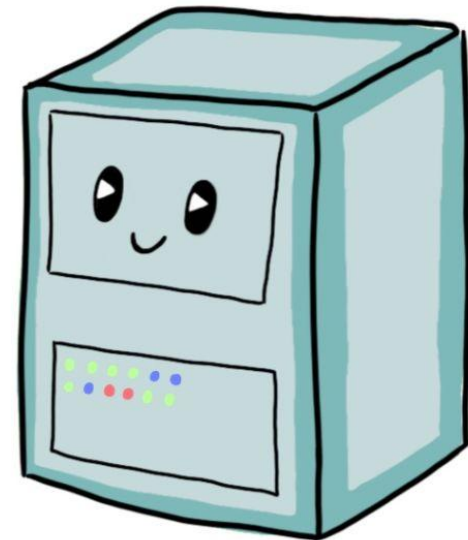
I don't know this site, but I can redirect you to the .COM TLD server!



Step 4

TLD Server

.COM 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.

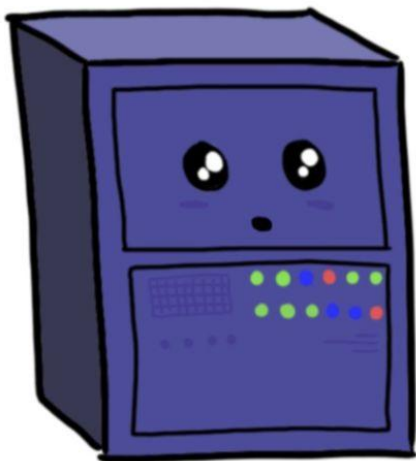
I don't know the IP address for rapidapi.com, but I can redirect the Resolver to the correct Authoritative Name Server.

Step 5

Authoritative Name Server

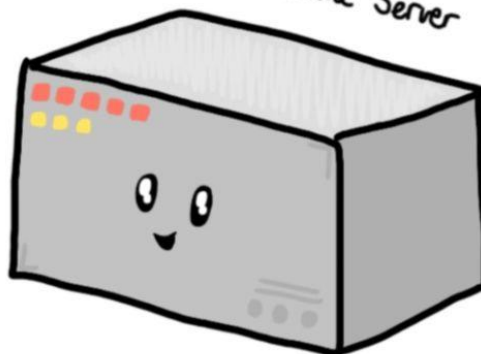
I'm looking for rapidapi.com.

Resolver Server



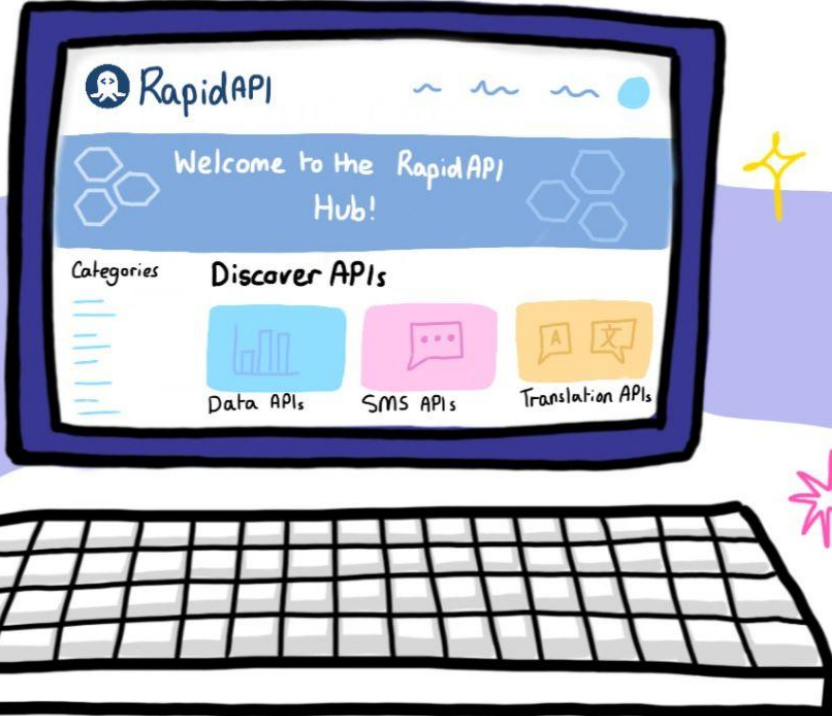
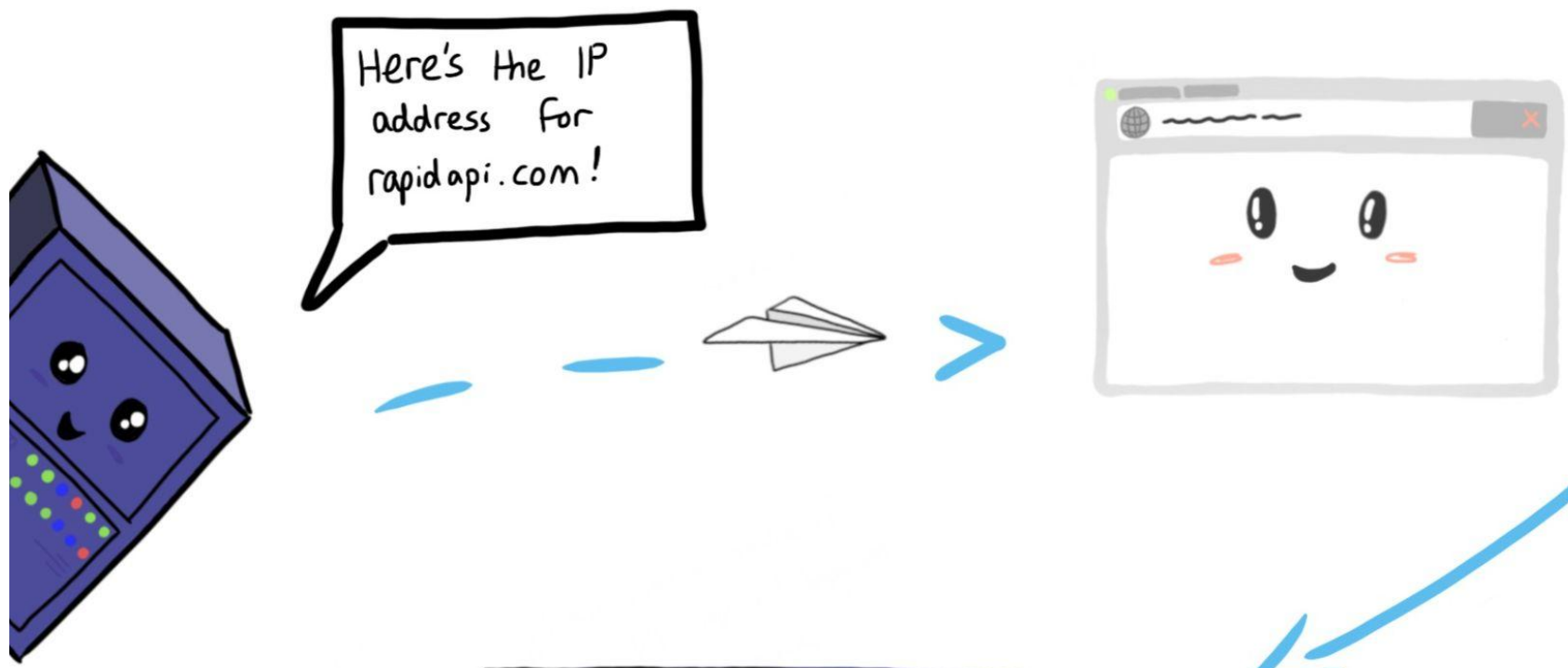
Got it! Here is the IP address for rapidapi.com.
2606:4700:3108::ac42:2918

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.



The website is loaded in the browser! All of this happens within a few milliseconds.

That's the end of the DNS system!

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