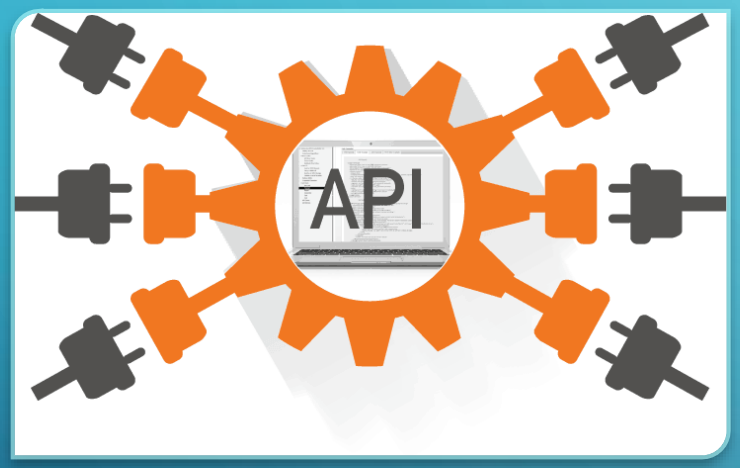
# RESTFUL WEB API

# WHAT IS AN API?

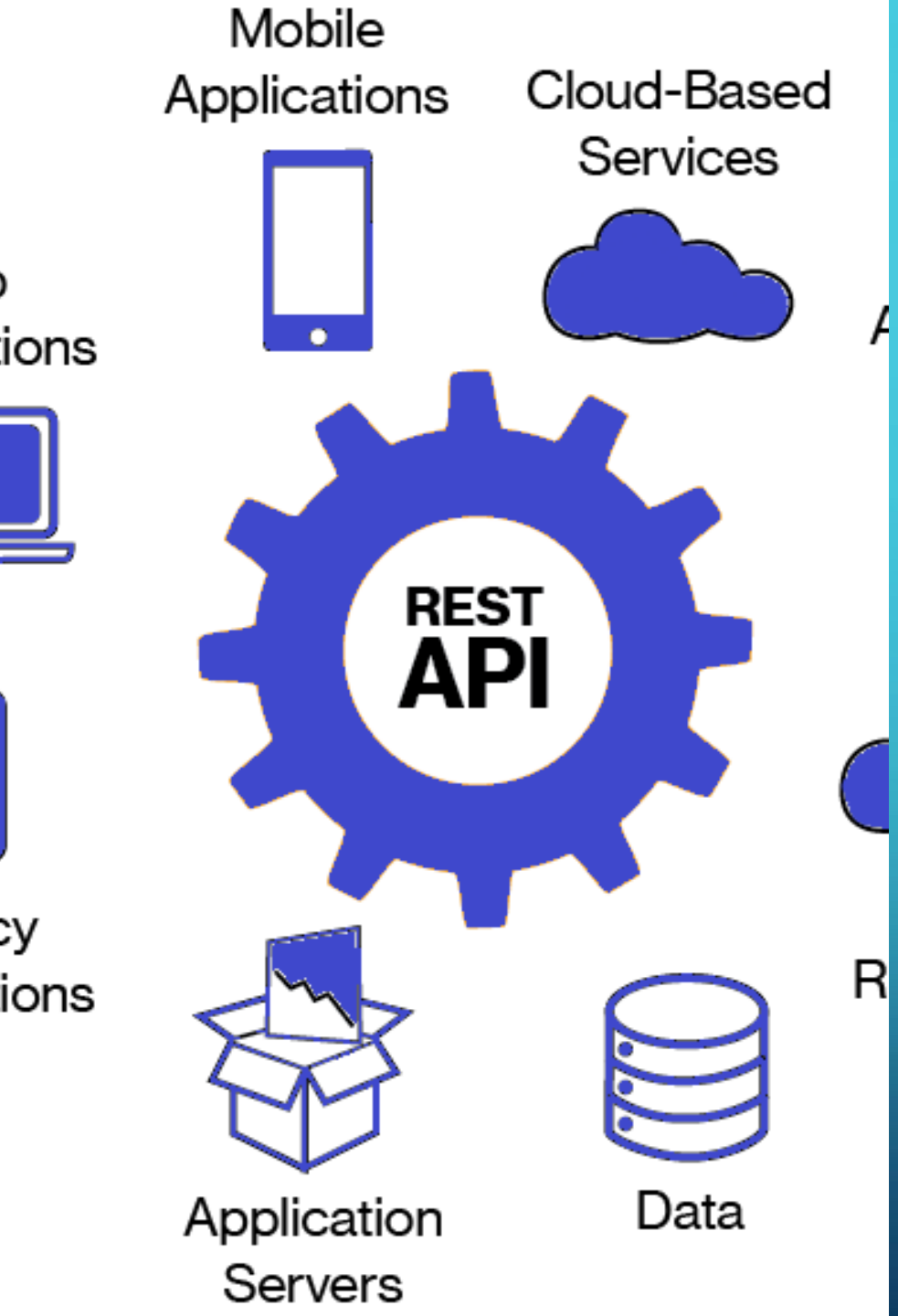



An API (Application Programming Interface) is a mechanism that is designed to  provide a standardized way for two systems to communicate to each other.

Because different systems may be written in different languages and be operating on different platforms, they are often not directly compatible with each other.

For example, a C# application may not be able to directly communicate with something written in Rust or Python.

In this situation we can use and API to manage the interactions between the 2 systems so they can communicate with each other.




# REST API

REST stands for Representational State Transfer. It is an architecture style for designing loosely coupled applications over a network, that is often used in the development of web services.

REST or RESTFUL APIs are a specific implementation of an API that follow this architectural pattern to allow communication between systems over a network.

# THE RESTAURANT ANALOGY

You can think of an API as a waiter at a restaurant. When you go to a restaurant you get provided a menu of items that you can choose from. You can also sometimes add specific instructions about how you want those items prepared.

Once you know what you want, you don't walk into the kitchen and request it from the chef. Instead, you tell your waiter (the interface between you and the kitchen) what you want to eat (your request) and he manages the interaction with the kitchen.

The waiter takes your order and passes it to the kitchen, who prepares your meal to your specifications.

Once your meal is prepared, the waiter returns to you at your table with the outcome of your order.

# THE RESTAURANT ANALOGY

## API

1. API's outline a series of predefined methods or endpoints that can be used.
2. Clients can send a request to an endpoint to trigger a specific method
3. The Request is passed via HTTP from the client to the API, where it is handles and any communication between other systems is managed.
4. Once finished, the API will send a response via HTTP to the client containing the requested resource and/or information regarding the status of the request

## Restaurant

1. Restaurants prepare a menu, of the services or dishes they offer
2. Customers can select an item from the menu based upon what is available to order.
3. The order is given to the Waiter, who then handles the order and coordinates with the kitchen to fulfill your request.
4. Once your order is finished the Waiter will return with either a completed dish or some information regarding the order if there is a problem.

# THE MENU

## API

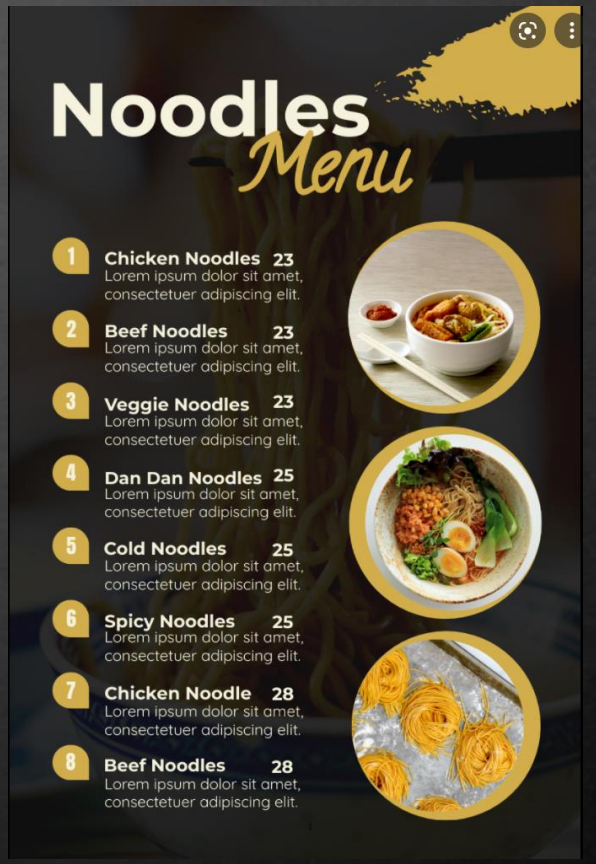
### HTTP method and URI list

Table 1. Cluster endpoints

| HTTP method | URI path | Description |
|---|---|---|
| GET | */pools* | Retrieves cluster information. |
| GET | */pools/default* | Retrieves cluster details. |
| POST | */controller/addNode* | Adds nodes to clusters. |
| POST | */node/controller/doJoinCluster* | Joins nodes into clusters |
| POST | */controller/ejectNodeentry* | Removes nodes from clusters. |
| GET, POST, PUT, DELETE | */pools/default/serverGroups* | Manages Server Group Awareness (server groups). |
| POST | */controller/rebalance* | Rebalances nodes in a cluster. |

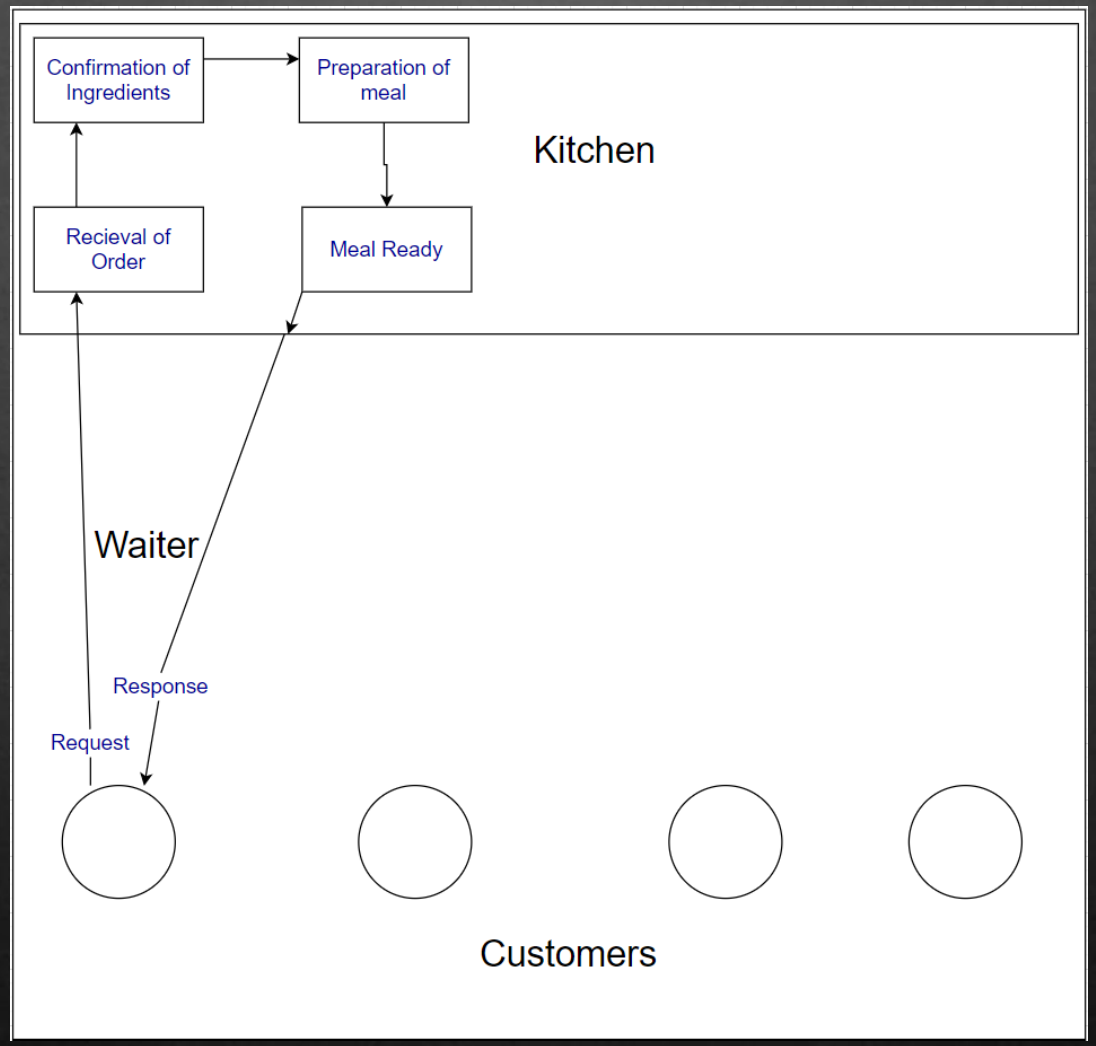
https://docs.couchbase.com/server/current/rest-api/rest-endpoints-all.html

## Restaurant

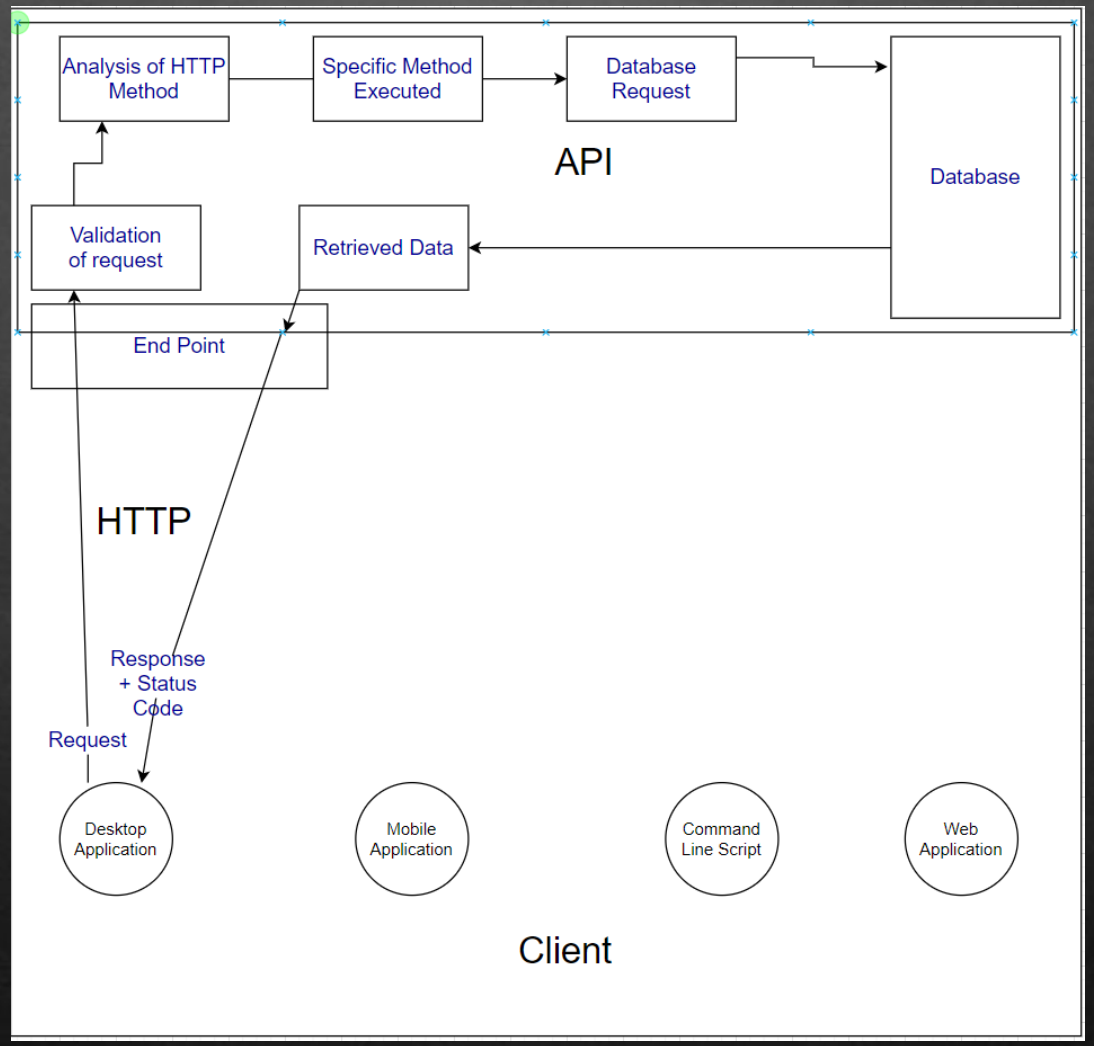



https://warren2lynch.medium.com/6-tips-on-restaurant-menu-design-double-your-revenue-33c921ad2c65

# THE WAITER


Confirmation of Ingredients
Preparation of meal
Kitchen
Recieval of Order
Meal Ready
Waiter
Response
Request
Customers

# API


Analysis of HTTP Method
Specific Method Executed
Database Request
API
Database
Validation of request
Retrieved Data
End Point
HTTP
Response + Status Code
Request
Desktop Application
Mobile Application
Command Line Script
Web Application
Client

# RESTFUL API PRINCIPLES

To class as following the REST or RESTful pattern, the API must be able to meet the following rules:

- Provide a uniform interface between components so that information is transferred in a standard form. This is generally done using a combination of URLs and Request types.
- A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
- Use Stateless client-server communication, meaning no client information is stored between requests and each request is treated separately and unconnected.
- Must offer the ability to Cache data that streamlines client-server interactions.
- Layered system: An application architecture needs to be composed of multiple layers. Each layer doesn't know anything about any layer other than that of the one layer they are immediately interacting with.

https://www.redhat.com/en/topics/api/what-is-a-rest-api

# ENDPOINTS

Each HTTP call from the client will be directed at a URL sub address of the API which is called an endpoint. These endpoints are similar in structure to how web pages manage navigation between pages. Each endpoint will be associated with a particular resource that the client is trying to access.

Examples: **https://mywebapi.com/api/users**

The example above is the endpoint for the users' resource section of the API. This could possibly be linked to multiple items of functionality that interacts with the Users table in a database.

To interact with a specific user, you can add an additional section to the endpoint to specify the desired user ID or adding query parameters to the URL.

**https://mywebapi.com/api/users/1 or https://mywebapi.com/api/users?userId=1**

# HTTP METHODS

Modern REST APIs also use a series of HTTP verbs to describe the action the user is requesting. Each verb is associated with a particular type of action being performed.

- **GET** – used to retrieve information from a service – used by default when navigating to a page in a browser
- **PUT or PATCH** – used to specify the intent of updating an item - generally accompanied by identifying information about the item to update + the updated object attached
- **POST** – Used to specify the Creation of a new object, or the intent to do so. Is Accompanied by the data to be created (this is commonly used when the 'submit' button is clicked on an online form)
- **DELETE** – Used to specify that the resource specified should be removed. Will contain enough detail to identify a single item (or a range of items)

# SENDING A REQUEST

Depending on the combination of endpoint and HTTP verb used, different actions will be performed on the desired resources.

You can see in the table below that even though we are only using 2 different endpoint addresses, we are getting different functionality performed by applying a different HTTP verb to them.

| URI | HTTP verb | Functionality |
|---|---|---|
| api/users | GET | Get all users |
| api/users/1 | GET | Get a user with id = 1 |
| api/users | POST | Add a user |
| api/users/1 | PUT | Update a user with id = 1 |
| api/users/1 | DELETE | Delete a user with id = 1 |

# POST REQUESTS

- When sending a POST request to an API, we need to include the object we want to create
- The object sent in the body of a request to an API endpoint should conform to JavaScript Object Notation (JSON)
- JSON can be described as a lightweight data-interchange format
- JSON uses a “key”: value structure to connect object properties to values (where the property name is the “key”)
- Other requests such as PUT and PATCH will also often do this but it is not always necessary.

# SENDING DATA

In most REST APIs, when sending complete objects to or from the API, it is normally done using a data interchange language like JSON. Below is an example of how a simple data class (POCO) may be represented C# and in how objects created against this will be represented in JSON

## C#

```
public class Tool
{
    public int ToolId { get; set; }
    public string ToolName { get; set; }
    public string Brand { get; set; }
    public string Condition { get; set; }
    public bool IsAvailable { get; set; }
}
```

## JSON

```
{
    "toolId":1,
    "toolName":"18v Circular saw",
    "brand":"Ozito",
    "condition":"Good",
    "isAvailable":true
}
```

# API TESTING

Because APIs do not have a visual interface by default, we need other ways to interact with them for testing if we have not yet built an API client system. This is necessary to make sure all our endpoints are functional during development.

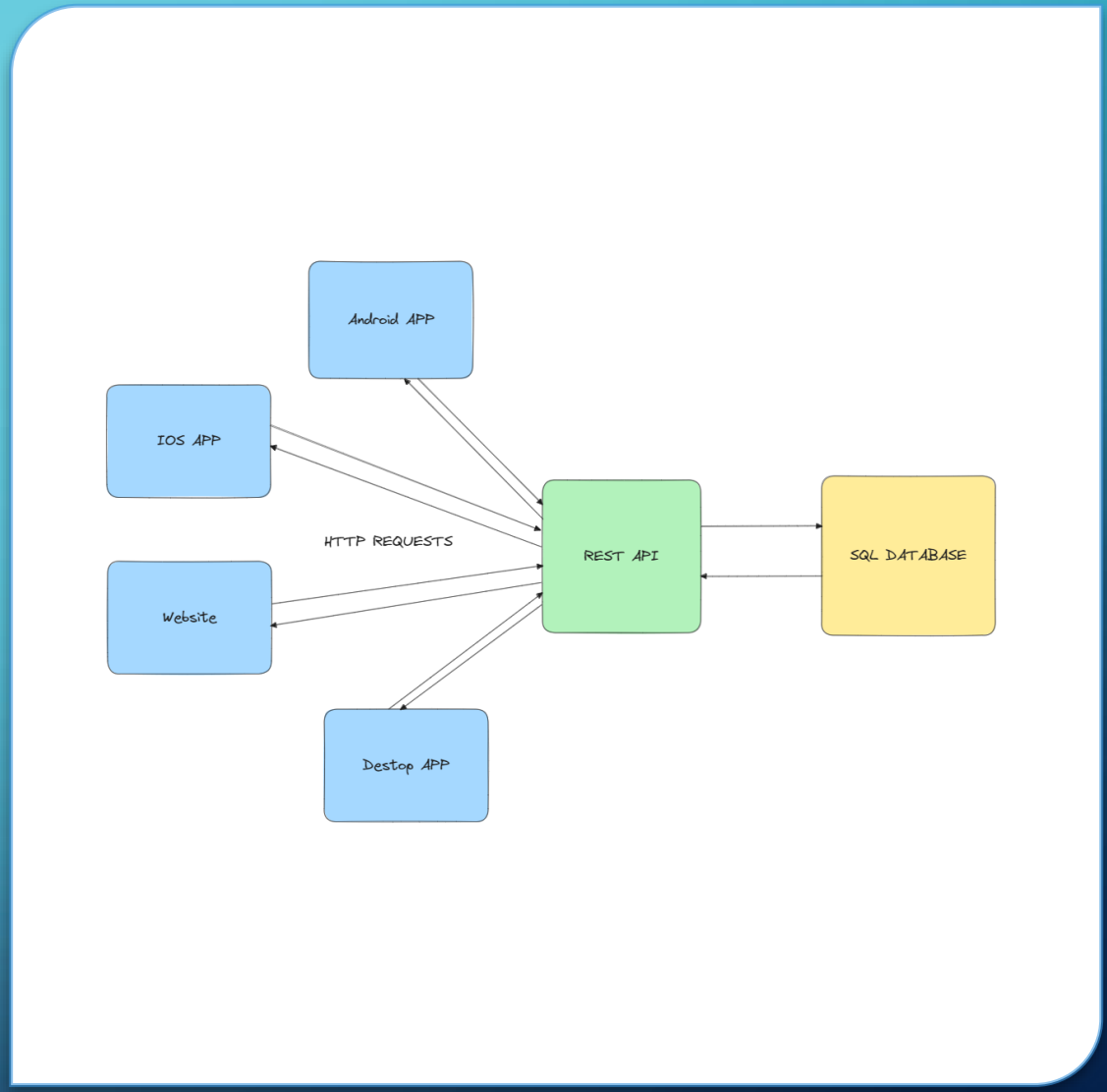
- GET methods can be tested with a browser by typing into the URL.
- Some API frameworks provide a simple interface for testing like the Swagger UI in .NET
- Postman is a purpose-built Web API testing application which allows us to test an API using a provided interface.

# REST API CLIENT

- Rest API does not have a visual front end and is normally interacted with by a separate front-end system which is commonly referred to as a client.
- Rest API Clients can be built with any language that can send and receive HTTP Requests and responses.
- Multiple clients developed in different languages and frameworks can all be interacting with the same API, as long as they all are able to send the required requests and process the provided responses.

# REST API CLIENTS

As you can see in the example here, we can have multiple different front-end clients all interacting with the API to retrieve or send data to the connected database.



# ACTIVITY

- Send a GET request (navigate in a browser) to https://restcountries.com/v3.1/all This is an API endpoint that displays information about countries, this request will display information about ALL countries
- Using the documentation at: https://restcountries.com/#api-endpoints-v3-all complete the following:
  - Retrieve the current population of Australia
  - Retrieve a link to the Australian Flag
  - Retrieve the size of Australia in m2
  - The Time-Zone of Venezuela
  - The longitude and latitude of the Capital of Canada

# BUILDING AN API

- It's now time to build your own API. Recommended tooling includes:
- Visual Studio 2022 (Latest)
  - ASP.NET web development package
  - .Net core cross platform development package
- The following NuGet packages:
  - Mongo Db
    - MongoDB.Driver
  - SQL Server or SQLite
    - Microsoft.EntityFrameworkCore
    - Microsoft.EntityFrameworkCore.SQLServer **OR** Microsoft.EntityFrameworkCore.SQLite
    - Microsoft.EntityFrameworkCore.Tools