cybercom creation

JAVASCRIPT

API CALLS

By Pritey Mehta

TABLE OF CONTENTS

01 WHAT IS API?

02 API PROCESS

03 TYPES OF API

04 COMMON METHODS

05 AJAX/XHR

06 FETCH

ABOUT

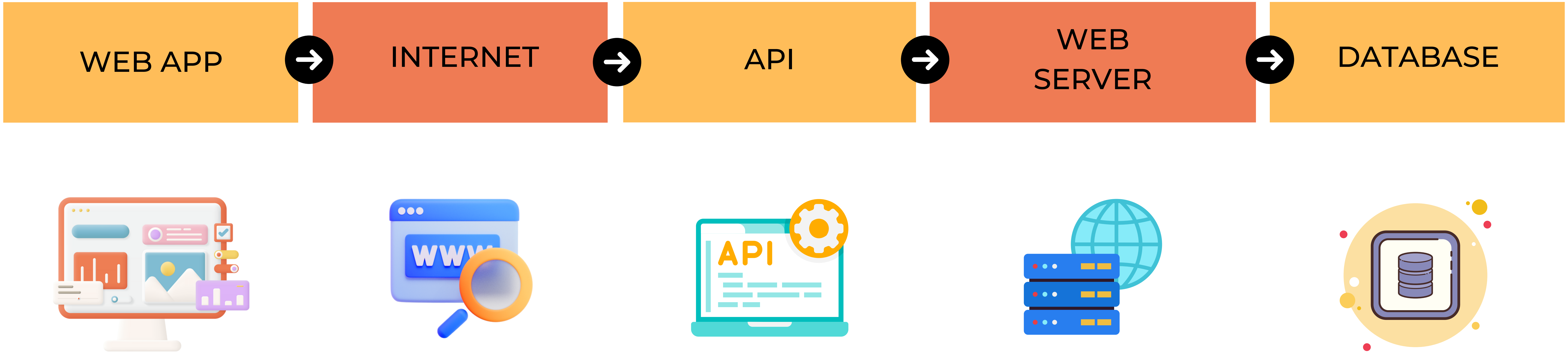
WHAT IS API?

An API (Application Programming Interface) allows two separate systems to communicate with each other, such as a front-end JavaScript application and a back-end server.

API calls are used to retrieve or modify data from a server using the HTTP protocol.

ABOUT

API PROCESS



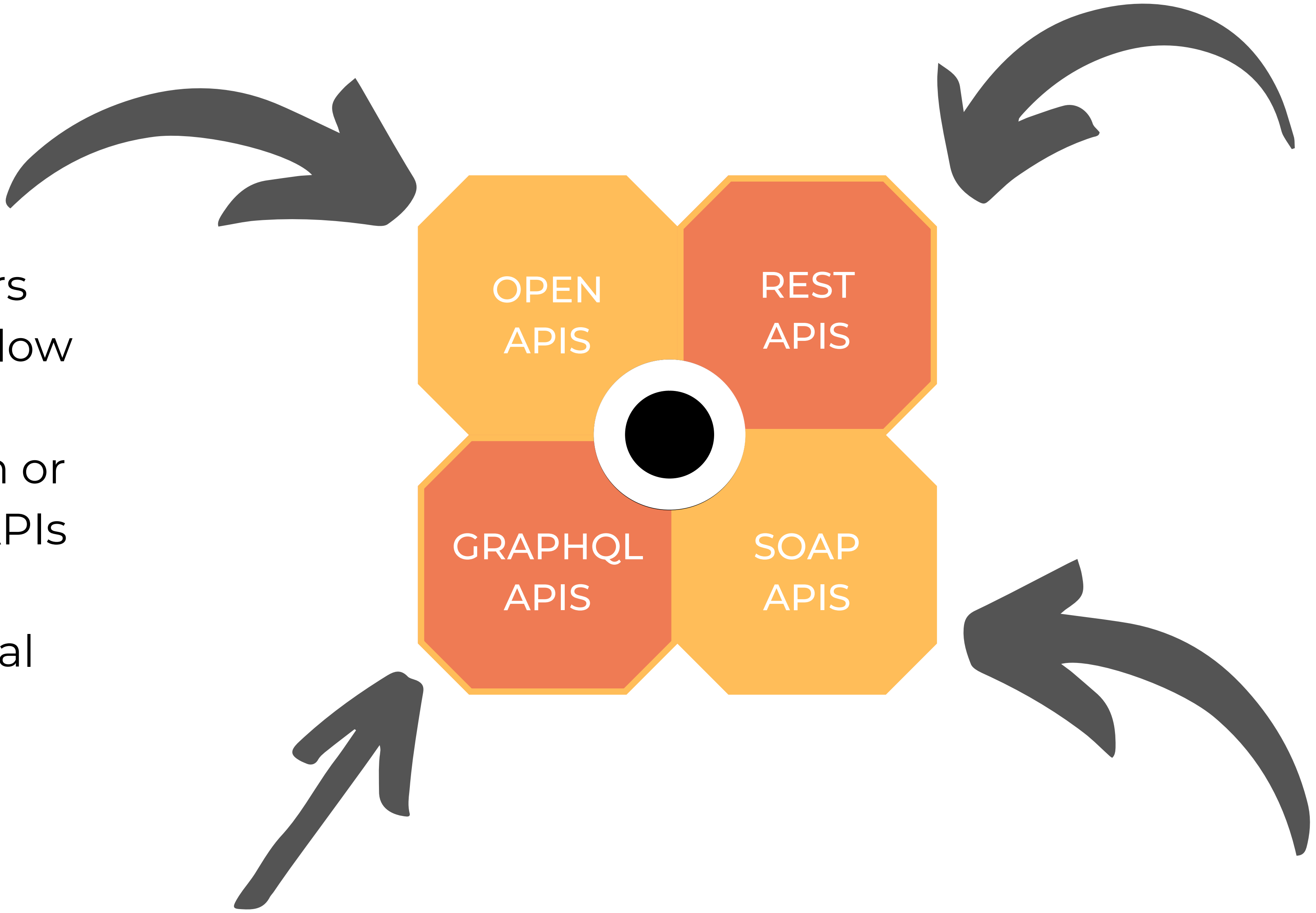
WEB APP → INTERNET → API → WEB SERVER → DATABASE

The client sends an API request to the API server using a specific endpoint URL.
The API server receives the request and processes it.
The API server sends a response back to the client.
The client receives the API response and processes the returned data.

API

TYPES

Open APIs, available to developers and other users with minimal restrictions, allow the accessing of data or services from an application or service. Examples of open APIs include weather APIs, geolocation APIs, and social media APIs.



Representational State Transfer (REST) APIs are a type of web API that uses HTTP requests to GET, PUT, POST, and DELETE data. REST is a stateless architecture in which clients can access and manipulate resources on a server.

GraphQL APIs are a newer type of API that were developed as an alternative to REST APIs. They allow for more flexible and efficient querying of data, as well as the ability to retrieve only the data that's needed.

Simple Object Access Protocol (SOAP) APIs are a protocol for exchanging structured information in the implementation of web services. They use XML as the format for sending and receiving data.

API

COMMON HTTP METHODS

GET

Retrieves data from a specified resource. A GET request is used to retrieve information from the server, and should not cause any side-effects (such as modifying data).

POST

Submits data to be processed to a specified resource. A POST request is used to submit data to the server for processing, such as creating a new resource or updating an existing resource.

PUT

Updates a current resource with new data. A PUT request is used to update an existing resource on the server, such as updating a user's information.

DELETE

Deletes a specified resource. A DELETE request is used to delete a resource on the server, such as deleting a user account.

API'S

AJAX

AJAX (Asynchronous JavaScript and XML) is a technique for creating fast and dynamic web pages. It allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

XHR (XMLHttpRequest) is an API that provides the ability to make HTTP requests from a web page's JavaScript code. XHR is the original mechanism used by AJAX to perform asynchronous requests and has since been widely adopted by other technologies and APIs.

To use XHR, you create an instance of the XMLHttpRequest object, define the request method (such as GET or POST), specify the URL to send the request to, and register a callback function to handle the response. The response can be processed using JavaScript to update the page's content dynamically.

OVERVIEW

XMLHTTPREQUEST

The XMLHttpRequest (XHR) is an API in the form of an object, available in JavaScript, that provides methods for making HTTP requests, such as GET, POST, PUT, and DELETE, from a web page to a server.

It allows for a web page to retrieve data from a server or to send data to a server without having to refresh the entire page. This enables the creation of dynamic and interactive web pages, making it a key technology for modern web development.

Here is the basic syntax for using the XMLHttpRequest object:

```
var xhr = new XMLHttpRequest();
```

To open a request, you use the open() method:

```
xhr.open(method, url, async);
```

OVERVIEW

XMLHTTPREQUEST

To set request headers, you use the setRequestHeader() method:

```
xhr.setRequestHeader(header, value);
```

The header parameter is the header name, and the value parameter is the header value.

To send a request, you use the send() method:

```
xhr.send(data);
```

The **data** parameter is the request data, which is only necessary for POST and PUT requests.

Note that this is just a brief overview of the XMLHttpRequest API, and there are many additional methods and properties that you can use to customize and refine your API requests.

OVERVIEW

XMLHTTPREQUEST

To monitor the state of a request and retrieve the response, you use the onreadystatechange event:

```
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE) {
    console.log(xhr.responseText);
  }
};
```

The **readyState** property of the XMLHttpRequest object returns the state of the request, and the **DONE** constant indicates that the request has completed. The **responseText** property contains the response data as a string.

XHR EXAMPLE

GET

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.example.com/data", true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText);
  }
};
xhr.send();
```

Note that these examples are just a starting point, and you'll need to handle errors, status codes, and other details for a production-ready implementation. Also, in modern JavaScript, you might consider using the Fetch API or a library such as Axios for making API requests, as they provide a more convenient and modern interface for making HTTP requests.

XHR EXAMPLE

POST

```
var xhr = new XMLHttpRequest();
xhr.open("POST", "https://api.example.com/data", true);
xhr.setRequestHeader("Content-Type", "application/json");
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 201) {
    console.log(xhr.responseText);
  }
};
xhr.send(JSON.stringify({ key: "value" }));
```

Note that these examples are just a starting point, and you'll need to handle errors, status codes, and other details for a production-ready implementation. Also, in modern JavaScript, you might consider using the Fetch API or a library such as Axios for making API requests, as they provide a more convenient and modern interface for making HTTP requests.

XHR EXAMPLE

PUT

```
var xhr = new XMLHttpRequest();
xhr.open("PUT", "https://api.example.com/data/123", true);
xhr.setRequestHeader("Content-Type", "application/json");
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText);
  }
};
xhr.send(JSON.stringify({ key: "new value" }));
```

Note that these examples are just a starting point, and you'll need to handle errors, status codes, and other details for a production-ready implementation. Also, in modern JavaScript, you might consider using the Fetch API or a library such as Axios for making API requests, as they provide a more convenient and modern interface for making HTTP requests.

XHR EXAMPLE

DELETE

```
var xhr = new XMLHttpRequest();
xhr.open("DELETE", "https://api.example.com/data/123", true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 204) {
    console.log("Data deleted successfully.");
  }
};
xhr.send();
```

Note that these examples are just a starting point, and you'll need to handle errors, status codes, and other details for a production-ready implementation. Also, in modern JavaScript, you might consider using the Fetch API or a library such as Axios for making API requests, as they provide a more convenient and modern interface for making HTTP requests.

ABOUT

FETCH

The fetch API is a modern alternative to XMLHttpRequest for making HTTP requests in JavaScript. It provides a simple, streamlined interface for making asynchronous network requests, and is widely supported across modern browsers.

Here is the basic syntax for using the fetch API:

```
fetch(url)
  .then(response => response.text())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

OVERVIEW

FETCH

The fetch function takes a single argument, the URL of the resource to be accessed. It returns a Promise that resolves to the Response object representing the response to the request.

The Response object has several properties, such as status, statusText, and headers, that provide information about the response, as well as several methods, such as text(), json(), and blob(), that allow you to extract the response data in different formats.

In the previous example, we use the text() method to extract the response data as a string, and then log the data to the console. We also use the catch method to log any errors that occur during the request.

You can also specify additional options for the request, such as the HTTP method, request headers, and request data, by passing an options object as the second argument to the fetch function.

OVERVIEW

FETCH

```
fetch(url, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    key: 'value'
  })
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

Note that the fetch API is just one way to make API requests in JavaScript, and there are other libraries and tools available that provide additional features and abstractions on top of the underlying fetch API.

cybercom creation

KEEP CODING!