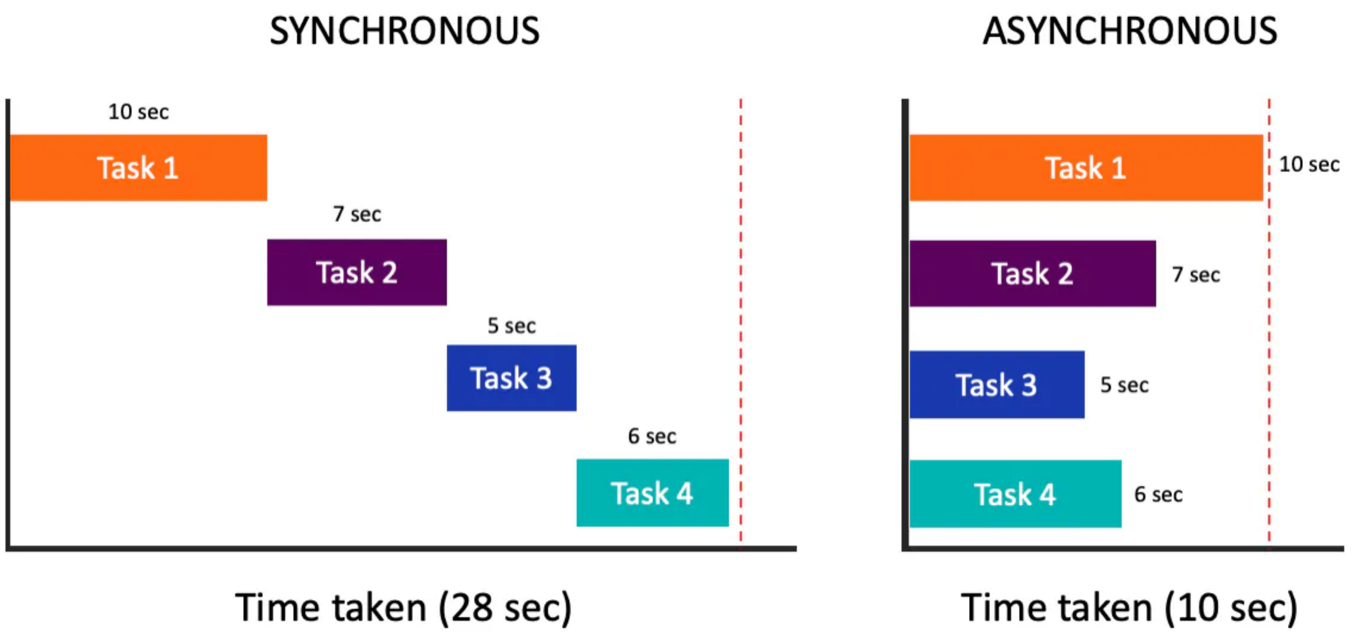
# JavaScript

Session 4

## Synchronous vs Asynchronous



## JS Callback

- A callback is a function that is passed as an argument to another function and is executed after the first function completes its operation.

Simple to understand
But,
No built-in mechanism for error handling

```
function fetchData(callback) {
   setTimeout(() => {
      callback('Data fetched using callback!');
   }, 1000);
}

fetchData((data) => {
   console.log(data);
});
```

// Outputs: "Data fetched using callback!"

## JS Promise

- A Promise represents a value which might be available now, or in the future, or never. Promises have methods .then(), .catch(), and .finally() for handling the asynchronous results.

```
function fetchData() {
   return new Promise((resolve, reject) => {
      setTimeout(() => {
         resolve('Data fetched using promise!');
      }, 1000);
   });
}

fetchData()
   .then(data => {
      console.log(data);
// Outputs: "Data fetched using promise!"
   })
   .catch(error => {
      console.error('Error:', error);
   });
```

Built-in error handling with .catch method
But,
Slightly more complex to understand compared to callback's

## **JS** Async/Await

- The async and await keywords enable asynchronous, promise-based behavior to be written in a clearer style, avoiding the need for configuring promise chains.

Built-in error handling with .catch method
But,
Slightly more complex to understand compared to callbacks

```
function fetchData() {
   return new Promise((resolve, reject) => {
      setTimeout(() => {
         resolve('Data fetched using async/await!');
      }, 1000);
   });
}

async function displayData() {
   try {
      const data = await fetchData();
      console.log(data);
// Outputs: "Data fetched using async/await!"
   } catch (error) {
      console.error('Error:', error);
   }
}
displayData();
```

## What is the output

```
function fetchData() {
	return new Promise((resolve, reject) => {
	console.log(‘Hello there’);
	setTimeout(() => {
	resolve('Data fetched successfully!’);
					}, 2000);
	});
}

fetchData().then(message => console.log(message));
console.log(‘not yet..’);
```

## How would you call this function and handle both the success and error cases?

```javascript
function divideNumbers(a, b) {
	return new Promise((resolve, reject) => {
	if (b !== 0) {
		resolve(a / b);
	} else {
		reject('Division by zero is not allowed’);
		}
	});
}
```

## What is an API?

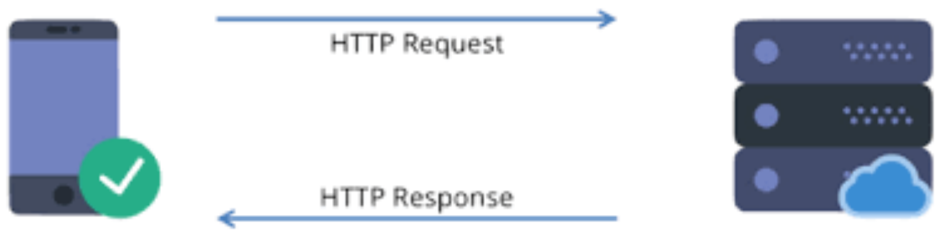
- API stands for **Application Programming Interface**.
- It's a set of rules and protocols that allow different software applications to communicate with each other.
- In the context of web development, APIs often allow a front-end application to communicate with a backend service or an external service.

## Why Use APIs?

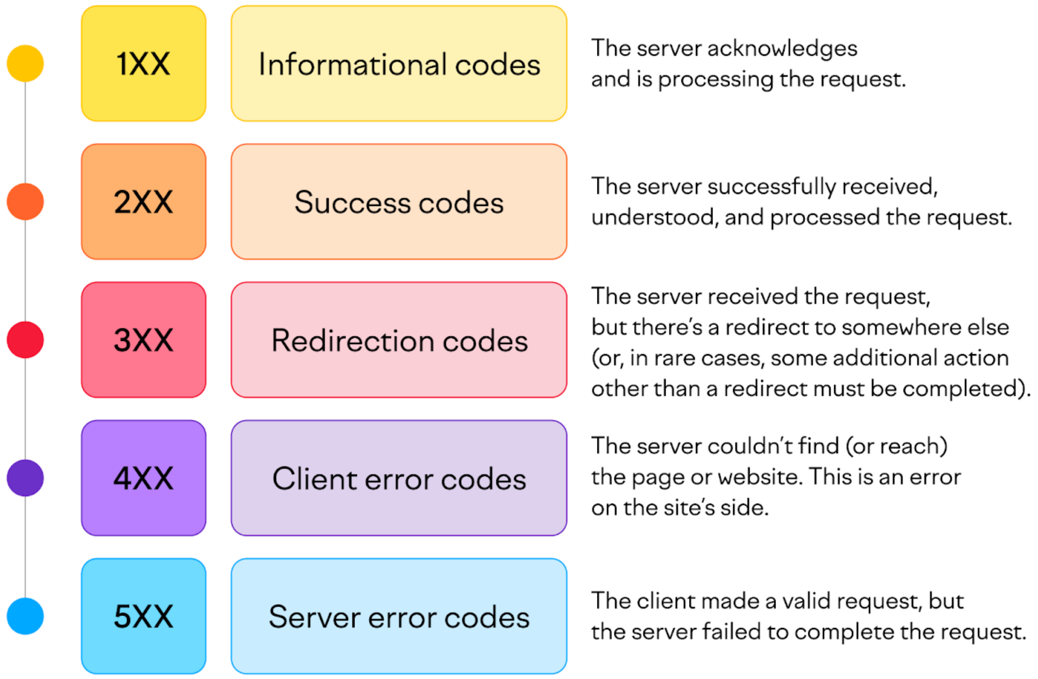
- Data Sharing: Retrieve or send data from/to servers.
- Integration: Connect different services and platforms.
- Automation: Automate repetitive tasks.
- Extend functionality: Use services that offer specific features (e.g., payment gateways, geolocation).

## Fetching Data in JavaScript

- JavaScript can request data from a server after a page has loaded.
- This enables dynamic content updates without refreshing the entire page.



## HTTP Status Codes



| | | |
|---|---|---|
| 1XX | Informational codes | The server acknowledges and is processing the request. |
| 2XX | Success codes | The server successfully received, understood, and processed the request. |
| 3XX | Redirection codes | The server received the request, but there's a redirect to somewhere else (or, in rare cases, some additional action other than a redirect must be completed). |
| 4XX | Client error codes | The server couldn't find (or reach) the page or website. This is an error on the site's side. |
| 5XX | Server error codes | The client made a valid request, but the server failed to complete the request. |

## Traditional Way - XMLHttpRequest

- The original method in web browsers for making HTTP requests.
- Uses **callback functions**.
- Can get complex and hard to manage for bigger tasks.

**XMLHttpRequest (XHR)**: This is the older way of making asynchronous requests in JavaScript before the fetch API came along. It's more complex and less elegant than fetch, but it's still widely used for historical reasons.

## XMLHttpRequest

```
const xhr = new XMLHttpRequest();
xhr.open('GET', `${baseUrl}q=${city}&appid=${apiKey}&units=metric`, true);
xhr.onload = function() {
    if (this.status >= 200 && this.status < 400) {
        const data = JSON.parse(this.response);
                // Handle data here
    } else {
        console.error('Server returned an error');
    }
};
xhr.onerror = function() {
    console.error('Request failed');
};
xhr.send();
```

## Modern Way - Fetch API

- Promise-based.
- More flexible and powerful than XHR.
- Cleaner and more readable syntax.

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

**Promises** are the foundation of **asynchronous programming** in modern JavaScript. A promise is an object returned by an asynchronous function, which represents the current state of the operation.

## Even Better - Async/Await

- Introduced with ES8 (ES2017).
- Allows asynchronous code to look and behave like synchronous code.
- Used in conjunction with promises.

```
async function fetchData(url) {
    try {
        const response = await fetch(url);
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
```

## Promise vs Async/Await

- **Promises**: Uses .then() for success, .catch() for errors. Can become nested.
- **Async/Await**: Uses try/catch for error handling. Code looks more linear and synchronous.

## Other Alternatives

- **jQuery's $.ajax method**: If you're using jQuery, you can use its $.ajax method, which provides a higher-level interface for making AJAX requests.

```
$.ajax({
    url: `${baseUrl}q=${city}&appid=${apiKey}&units=metric`,
    type: 'GET',
    dataType: 'json',
    success: function(data) {
        // Handle data here
    },
    error: function(error) {
        console.error('Failed to fetch weather data:', error);
    }
});
```

## Other Alternatives

- **Third-Party Libraries**: There are numerous third-party libraries like Axios, which offer enhanced features, better error handling, and cleaner syntax compared to the native methods.

```
axios.get(`${baseUrl}q=${city}&appid=${apiKey}&units=metric`)
   .then(response => {
       const data = response.data;
       // Handle data here
   })
   .catch(error => {
       console.error('Failed to fetch weather data:', error);
   });
```

## CRUD Operations in JavaScript

- Methods to interact with data on the web

| | |
|---|---|
| **GET (Read)** | fetch(url); |
| **POST (Create)** | fetch(url, { method: 'POST', body: data }); |
| **PUT (Update)** | fetch(url, { method: 'PUT', body: updatedData }); |
| **Delete (Delete)** | fetch(url, { method: 'DELETE' }); |

# Restful API vs GraphQl vs soap

| | SOAP (Simple Object Access Protocol) | REST (REpresentational State Transfer) | GraphQL | RPC (Remote Procedure Call) |
|---|---|---|---|---|
| Organized in terms of | enveloped message structure | compliance with six architectural constraints | schema & type system | local procedure call |
| Format | XML only | XML, JSON, HTML, plain text | JSON | JSON, XML, Protobuf, Thrift, FlatBuffers |
| Learning curve | Difficult | Easy | Medium | Easy |
| Community | Small | Large | Growing | Large |
| Use cases | - payment gateways<br>- identity management<br>- CRM solutions<br>- financial and telecommunication services<br>- legacy system support | - public APIs<br>- simple resource-driven apps | - mobile APIs<br>- complex systems<br>- micro-services | - command and action-oriented APIs<br>- high performance communication in massive micro-services systems |

## Weather API

- What type of information to expect?
- What is the base URL ?
- What is API KEY?
- Data Format?

```
{
  "location": {
    "name": "London",
    "country": "United Kingdom"
  },
  "current": {
    "temp_c": 15.0,
    "condition": {
      "text": "Partly cloudy"
    }
  }
}
```

$\underbrace{\text{https://api.openweathermap.org/data/2.5/}}\underbrace{\text{weather}}\text{?}\underbrace{\text{q={city name}&appid={API key}}}$

## Fetching Weather Data (READ)

1. Send an HTTP GET request
2. Handle the response (usually in JSON format)

```
fetch(https://api.openweathermap.org/data/2.5/weather?q=London&appid=x897987gd')
  .then(response => response.json())
  .then(data => console.log(data));
```

## CREATE Operation

1. Specify Method, Headers, and Body
2. Response is usually Acknowledgement!

```
fetch('https://api.weatherapi.com/v1/locations', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'API-Key': 'YOUR_API_KEY'
  },
  body: JSON.stringify({
    locationName: 'New Location',
    latitude: 40.7128,
    longitude: -74.0060,
    weatherData: {
      temperature: 25,
      condition: 'Sunny'
    }
  })
})
.then(response => response.json())
.then(data => console.log('Location Added:', data))
.catch(error => console.error('Error:', error));
```

## UPDATE Operation

1. Specify the id that needs to be updated in the url path

```
fetch('https://api.weatherapi.com/v1/locations/LOCATION_ID',
{
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
    'API-Key': 'YOUR_API_KEY'
  },
  body: JSON.stringify({
    weatherData: {
      temperature: 26,
      condition: 'Partly Cloudy'
    }
  })
})
.then(response => response.json())
.then(data => console.log('Weather Data Updated:', data))
.catch(error => console.error('Error:', error));
```

## DELETE Operation

1. No body!
2. Specify the data that needs to be deleted
3. Response (200 OK)

```
fetch('https://api.weatherapi.com/v1/locations/LOCATION_ID',
{
  method: 'DELETE',
  headers: {
    'API-Key': 'YOUR_API_KEY'
  }
})
.then(response => {
  if (response.ok) {
    console.log('Weather Data Deleted Successfully');
  } else {
    console.error('Deletion Failed');
  }
})
.catch(error => console.error('Error:', error));
```