

---

# JavaScript

Session 4

---

---

# 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?



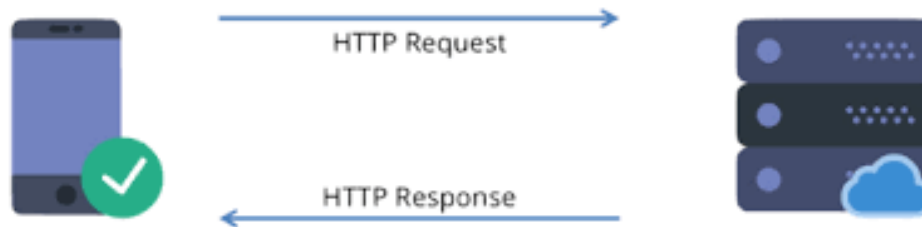
- Data Sharing: Retrieve or send data from/to servers.
- Integration: Connect different services and platforms.
- Automation: Automate repetitive tasks.
- Extend functionality: Use services that offer specific features (e.g., payment gateways, geolocation).



# 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.



+

---

# 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();
```



# 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.



# 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	<ul style="list-style-type: none"><li>- payment gateways</li><li>- identity management</li><li>- CRM solutions</li><li>- financial and telecommunication services</li><li>- legacy system support</li></ul>	<ul style="list-style-type: none"><li>- public APIs</li><li>- simple resource-driven apps</li></ul>	<ul style="list-style-type: none"><li>- mobile APIs</li><li>- complex systems</li><li>- micro-services</li></ul>	<ul style="list-style-type: none"><li>- command and action-oriented APIs</li><li>- high performance communication in massive micro-services systems</li></ul>