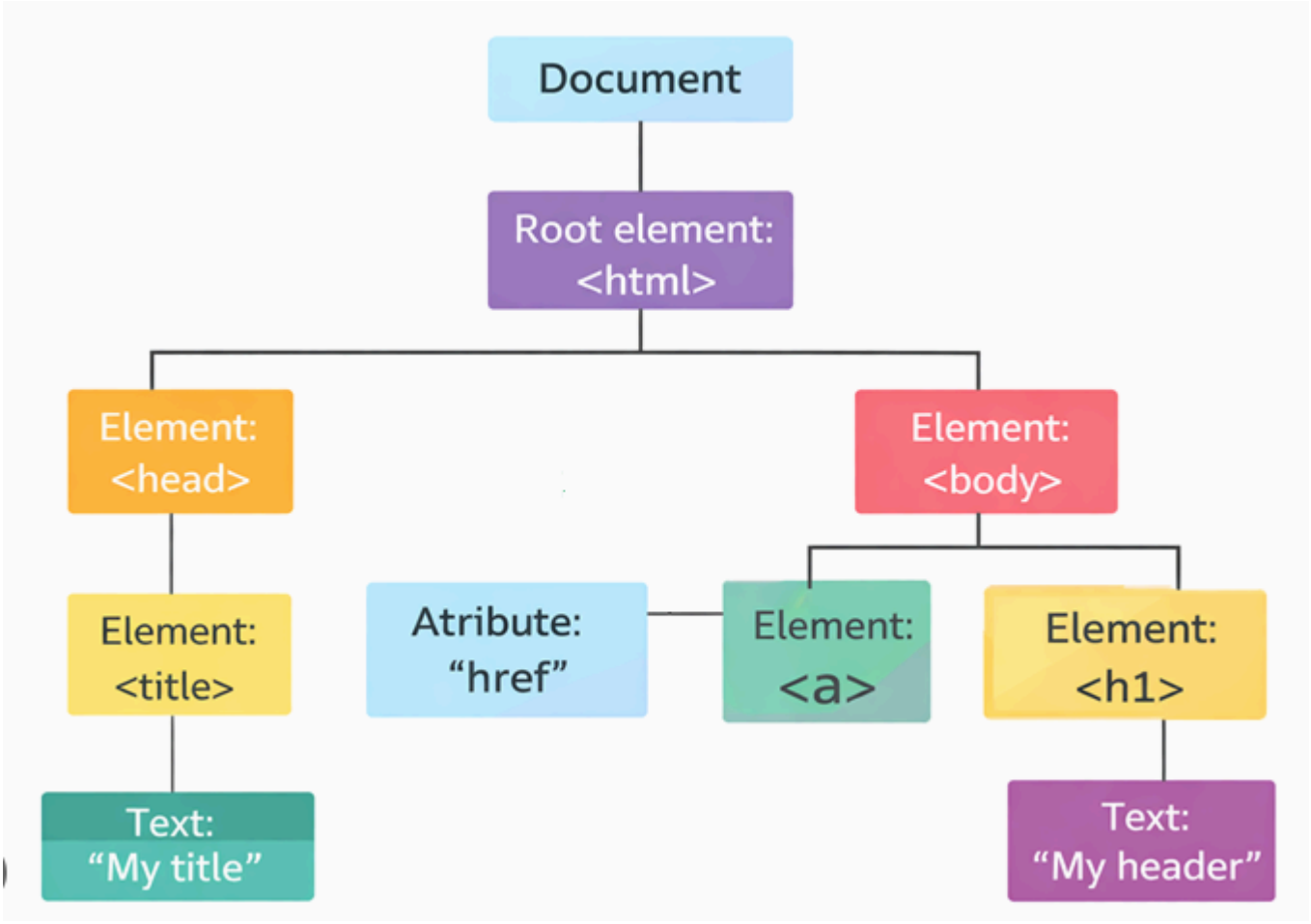
Week 4 Day 21: Mastering Advance Js Minutes of sessions

DOM Manipulation: DOM Objects, Accessing elements in the DOM, Modifying HTML content, Adding and removing elements Closures, Promises and Async Programming, Closures and Scoping,
Functional Programming, Error Handling client side scripting
JSON, JSON - Objects, arrays etc, JSON-Ajax Advanced DOM Manipulation,
jQuery Plugins, Event Delegation, Ajax and Deferred Objects, Optimization and Performance Diff between JQuery and Bootstrap ? AJAX - Asynchronous javascript and XML



DOM (tree like structure):

- It is like a tree representation of HTML
- Here Every HTML element becomes a javascript object.
- So that it can be accessed , modified, added or removed dynamically.

Why DOM :

- Enables Dynamic UI Updated
- Allows interaction without page reload.(partial page upload)_ AJAX
- Foundation of REACT, Angular, Vue.

DOM Objects

DOM Object	Description
document	Entry point to the DOM
window	Browser window object
Element	Any HTML tag
Node	Generic DOM unit
Event	User actions (click, input, submit)

Different ways of accessing element in DOM:

1. document.getElementById("username");
2. document.getElementsByClassName("card");
3. document.getElementsByTagName("p");
4. document.querySelector(".card");
5. document.querySelectorAll(".card");

Adding & Removing Elements:

Create Element const div = document.createElement("div"); div.innerText = "New Element";	**Add to DOM** document.body.appendChild(div);
Remove Element div.remove();	**Replace Element** parent.replaceChild(newElement, oldElement);

Case Study : Dynamic Event Registration Dashboard (Client-Side Only)

A society clubhouse is hosting multiple events. The admin wants a **single-page web interface** where users can:

- View available events
- Register / unregister dynamically
- See live participant counts
  All interactions must happen using **DOM manipulation (no backend, no frameworks).**

Objective	DOM Concept Applied
Access and manage UI elements	DOM Objects
Read and update UI data	Modifying HTML
Add new elements dynamically	Creating & Appending Elements
Remove elements based on user action	Removing Elements

User Story ID	User Story	DOM Skill Used
US-01	As a user, I want to view all available events when the page loads	DOM Objects
US-02	As a user, I want to click on an event and see its details	Accessing DOM
US-03	As a user, I want to register for an event and see confirmation instantly	Modify Content
US-04	As a user, I want to see the participant count increase in real time	Modify Text
US-05	As a user, I want to unregister from an event	Remove Elements
US-06	As an admin, I want to add a new event dynamically	Add Elements

Step 1: Creating a basic HTML page with following :

- Container div
- Crating headings
- Event list section
- event details section
- creating participant count
- creating buttons

Step 2: Access DOM Object (js)

- Accessing event list
- Accessing Details section
- Access buttons
- Access count span

Step 3: Display event list dynamically

- Looping through data
- Create a list item
- Add event name
- Append it to list

Defining the scope of variable in JS/ES6 we have : Var, Const and let
What is a scope ?
It defines where a variable is accessible.

Scope Type	Description
Global	**Accessible everywhere**
Function	**Accessible inside function**
Block (let, const)	**Accessible inside {}**

Closure
A closure is when a function remembers variables from its outer scope, even after the outer function has finished execution.

Why Closure matter :

1. **Data privacy**
2. **State management (since HTTP is stateless)**
3. **Callbacks & event handlers**
4. **Used majorly in frameworks.**

```
> function counter() {
    let count = 0;
    return function () {
      count++;
      return count;
    };
  }

  const increment = counter();
  increment(); // 1
  increment(); // 2
< 2
> increment();
< 3
> increment();
< 4
>
```

Promises: A Promise represent future completion or failure

call back hell : to handle it we have following ways :-

1. To handle it via promises(Older methods)

2. To implement Asynchronous programming using Async/Await() - (modern approach)
3. In case of APi we have Fetch API()

```
doA(() => {
  doB(() => {
    doC(() => {
      doD();
    });
  });
});
```

States of promises :

1. **Pending**
2. **Fulfilled**
3. **Rejected**

Steps for implementing promises :

Step 1 : creating a promise :	**Step 2 : calling/consuming a promise :**
const promise = new Promise((resolve, reject) => { resolve("Success"); });	**promise .then(result => console.log(result)) .catch(error => console.log(error));**

Case Study: Online Order Status Checker (Using Promises)

Context

A user places an online order. The system needs to:

- **Check order status**
- **Confirm delivery**
- **Handle errors properly**
  All operations are simulated using JavaScript Promises (no API, no backend).

User Story ID	User Story	Promise Concept
US-01	As a user, I want to check my order status so that I know whether my order is confirmed	Promise creation
US-02	As a user, I want to see delivery details once the order is confirmed	Promise chaining (then)
US-03	As a user, I want to see an error message if my order fails	Promise rejection (catch)

Step 1: Create a Promise to Check Order Status	**Step 2: Chain promise to get delivery details**

function checkOrderStatus(orderId) { return new Promise((resolve, reject) => { if (orderId) { resolve("Order Confirmed"); } else { reject("Invalid Order ID"); } }); }	function getDeliveryDetails(status) { return new Promise((resolve) => { resolve(`${status} - Delivery in 3 days`); }); }
Step 3: Consume promises(then & catch) checkOrderStatus(101) .then(status => getDeliveryDetails(status)) .then(result => console.log(result)) .catch(error => console.error(error));	

Using async Await(modern way)

- **Cleaner Code**
- **Easier debugging**
- **No callback hell**
-

```
async function fetchData() {
  try {
    const response = await fetch("https://api.example.com/data");
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}
```

Functional Programming In JS/ES6 refers to the programming paradigm where functions are considered as **first class citizens _ all the importance is given to them hence immutable data and pure function is of top priority.**

Principle	Description
Pure Functions	Same input → same output
Immutability	Data is not changed directly
First-class Functions	Functions can be passed as values
No Side Effects	Avoid modifying external state

Method	Purpose
map()	Transform array
filter()	Select data
reduce()	Aggregate values
forEach()	Iterate

Benefits **1. cleaner code** **2. Easier testing** **3. Better scalability** **4. Widely used React and redux.**	

Error Handling in Js:

1. **It prevents application crashes**
2. **Improves user experience**
3. **helps debugging**

Global Error Handling

Ex.

```
window.onerror = function(message, source, line) {
  console.log(message);
};
```

try–catch Block	Handling Errors in Promises
try { JSON.parse("{invalid}"); } catch (error) { console.error(error.message); }	**fetch(url)** ** .then(res => res.json())** ** .catch(err => console.error(err));**

JSON: Javascript object notation
Light weight data interchange format used for exchange between client and server.

JSON Data Types

Type	Example
Object	**{ "name": "John" }**
Array	**[1, 2, 3]**
String	**"Hello"**
Number	**100**
Boolean	**true**
Null	**null**

Overall flow with JSON data : - (JSON - AJAX FLOW)

1. **Client sends request**
2. **Server returns JSON**
3. **Javascript parses JSON**
4. **UI updates dynamically**

Parsing JSON const data = JSON.parse(response);	Converting JS to JSON const json = JSON.stringify(obj);

AJAX - Synchronous javascript and XML - Partial page update

1. **AJAX allows asynchronous data loading without page reload.**
2. **Makes page rendering faster and smooth.**
3. **UI/UX looks dynamic.**

Case Study

Smart Product Catalog Web Application (Client-Side Only)

Business Context

A retail company wants a dynamic product catalog page that loads products asynchronously, displays them efficiently, and provides a smooth user experience using modern JavaScript, AJAX, JSON, jQuery, and performance best practices.

Case Study Objectives

Area	Covered
Functional Programming	**Data transformation**
Error Handling	**Client-side stability**
JSON & AJAX	**Server communication**
Advanced DOM Manipulation	**Dynamic UI**
jQuery, Event Delegation	**Efficient event handling**
Async/Await & Deferred	**Async flow**
Performance Optimization	**UI responsiveness**

User Stories (5 Only)

User Story 1 – Load Product Data Asynchronously

Item	Description
User Story ID	US-01
As a	Customer
I want	Products to load without page refresh
So that	The website feels fast and responsive
Concepts Applied	AJAX, JSON, Async/Await

Acceptance Criteria

- Products are fetched using AJAX
- JSON response is parsed
- UI updates dynamically
- Page does not reload

User Story 2 – Display Products Using Functional Programming

Item	Description
User Story ID	US-02
As a	Customer
I want	Products filtered and formatted cleanly
So that	I can easily browse items

Concepts Applied	**Functional Programming (map, filter), JSON Arrays**

Acceptance Criteria

- **Products are filtered by category**
- **Prices are formatted using map()**
- **No original data mutation**

User Story 3 – Handle Errors Gracefully on Data Failure

Item	Description
User Story ID	**US-03**
As a	**User**
I want	**Clear error messages if data fails to load**
So that	**I understand what went wrong**
Concepts Applied	**Error Handling, try–catch, Promise rejection**

Acceptance Criteria

- **Network/API errors are caught**
- **User-friendly message is displayed**
- **App does not crash**

User Story 4 – Interact with Products Efficiently

Item	Description

User Story ID	**US-04**
As a	**Customer**
I want	**To add products to cart dynamically**
So that	**My experience is seamless**
Concepts Applied	**Advanced DOM Manipulation, Event Delegation**

Acceptance Criteria

- **Products are added dynamically to DOM**
- **Single event listener handles all buttons**
- **DOM updates are optimized**

User Story 5 – Enhance UI Using jQuery & Performance Techniques

Item	Description
User Story ID	**US-05**
As a	**Admin**
I want	**A smooth UI with reusable components**
So that	**The page performs well on all devices**
Concepts Applied	**jQuery Plugins, Deferred Objects, Optimization**

Acceptance Criteria

- **jQuery plugin used for UI enhancement**
- **Deferred objects manage async flows**
- **DOM updates are minimized**

- **Performance best practices followed**