 **Data Management**: It uses JavaScript arrays (projects and skills) to store data about projects and skills, which it dynamically renders into the HTML document. This modular approach facilitates easy updates to the project or skills data without needing to modify the HTML structure.

 **Event Listeners**: The code listens for various events such as load, scroll, click, and mousemove to trigger actions like hiding a loader, revealing a custom cursor, collapsing and expanding a responsive navigation menu, and managing header visibility based on scroll behavior.

 **Dynamic Content Generation**: JavaScript is used to create HTML content dynamically. It loops through the projects and skills arrays to build and append HTML elements (like project cards and skill icons) to the webpage, allowing for flexible and scalable content management.

 **CSS Manipulation and Animation**: It manipulates CSS styles directly via JavaScript (e.g., adjusting the cursor's position, toggling theme colors, and changing header visibility), demonstrating how to enhance user experience with responsive and interactive animations.

 **Form Validation and Interaction**: Implements client-side form validation for a contact form, providing instant feedback on input errors and handling form submission with JavaScript, showcasing how to enhance form usability without server-side processing.

 **Intersection Observer API**: Uses this API to create lazy loading effects for project cards, improving performance and user experience by loading content only as it enters the viewport.

 **Custom Typewriter Effect**: Includes a typewriter effect for hero section text, which cycles through an array of words, simulating typing and deletion animations to engage users immediately upon visiting the site.

**Sir jawad ka code 1 e commerce wala**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Following are some useful links to explore each topics  
HTML (Use any or all of following resources to prepare)

* <https://www.w3schools.com/html/>
* <https://developer.mozilla.org/en-US/docs/Web/HTML>

CSS (Use any or all of following resources to prepare)

* <https://www.w3schools.com/css/default.asp>
* <https://developer.mozilla.org/en-US/docs/Web/CSS>

Javascript  (Use any or all of following resources to prepare)

* <https://www.w3schools.com/js/default.asp>
* <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Browser:

* DOM (<https://www.w3schools.com/js/js_htmldom.asp>)
* Windows - BOM (<https://www.w3schools.com/js/js_window.asp>)

AJAX and Web APIs

* <https://www.w3schools.com/js/js_ajax_intro.asp>
* <https://www.w3schools.com/js/js_json_intro.asp>
* <https://www.w3schools.com/js/js_api_intro.asp>

Other Links:

* <https://www.freecodecamp.org/learn/2022/responsive-web-design/>
* <https://www.freecodecamp.org/news/learn-javascript-for-beginners/>
* <https://github.com/PacktPublishing/50-Projects-In-50-Days---HTML-CSS-JavaScript>
* <https://github.com/PacktPublishing/30-Web-Projects-with-HTML-CSS-and-JavaScript>

Certainly! Below, I'll provide code examples for each of the AJAX topics listed, demonstrating basic to advanced usage:

**1. Introduction to AJAX**

// Basic AJAX request using XMLHttpRequest

function loadText() {

const xhr = new XMLHttpRequest();

xhr.open('GET', 'sample.txt', true);

xhr.onreadystatechange = function() {

if(xhr.readyState === 4 && xhr.status === 200) {

console.log(xhr.responseText);

}

};

xhr.send();

}

loadText();

**2. The XMLHttpRequest Object**

// Creating and sending an XMLHttpRequest

const xhr = new XMLHttpRequest();

xhr.open('GET', 'api/data.json', true);

xhr.onload = function() {

if (this.status === 200) {

console.log(JSON.parse(this.responseText));

}

};

xhr.send();

**3. Working with Data Formats**

// Handling JSON data in AJAX

const xhr = new XMLHttpRequest();

xhr.open('GET', 'data.json', true);

xhr.onload = function() {

if (this.status === 200) {

const users = JSON.parse(this.responseText);

console.log(users);

}

};

xhr.send();

**4. Basic AJAX Operations**

// Making a POST request with AJAX

const xhr = new XMLHttpRequest();

xhr.open('POST', 'submit.php', true);

xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');

xhr.onload = function() {

console.log(this.responseText);

};

xhr.send('name=John&age=30');

**5. Updating the DOM with AJAX**

// Dynamically updating content received from AJAX

const xhr = new XMLHttpRequest();

xhr.open('GET', 'userinfo.php', true);

xhr.onload = function() {

if (this.status === 200) {

document.getElementById('user').innerHTML = this.responseText;

}

};

xhr.send();

**6. Event Handling in AJAX**

// Handling errors in AJAX requests

const xhr = new XMLHttpRequest();

xhr.open('GET', 'content.json', true);

xhr.onerror = function() {

console.error('Request failed.');

};

xhr.onload = function() {

if (this.status === 200) {

console.log('Data:', this.responseText);

} else {

console.error('Error:', this.status);

}

};

xhr.send();

**7. AJAX with jQuery**

// Using jQuery to simplify AJAX requests

$.ajax({

url: 'server.php',

type: 'GET',

success: function(data) {

console.log('Data received:', data);

},

error: function() {

console.error('Error fetching data.');

}

});

**8. AJAX Security**

// Handling CORS in AJAX

const xhr = new XMLHttpRequest();

xhr.open('GET', 'https://api.example.com/data', true);

xhr.withCredentials = true; // To include cookies in the request sent to the API

xhr.onload = function() {

console.log(this.responseText);

};

xhr.send();

**9. Working with APIs using AJAX**

// Fetching data from a RESTful API

const xhr = new XMLHttpRequest();

xhr.open('GET', 'https://api.example.com/users', true);

xhr.onload = function() {

if (this.status === 200) {

const users = JSON.parse(this.responseText);

console.log(users);

}

};

xhr.send();

**10. Optimizing AJAX Requests**

// Debouncing AJAX calls to minimize request frequency

let timeout;

function fetchData(input) {

clearTimeout(timeout);

timeout = setTimeout(() => {

const xhr = new XMLHttpRequest();

xhr.open('GET', `search.php?query=${input}`, true);

xhr.onload = function() {

if (this.status === 200) {

console.log(this.responseText);

}

};

xhr.send();

}, 300); // Delay the AJAX request by 300 ms

}

These examples should give you a practical sense of how to implement AJAX across a range of uses, from basic requests to more complex interactions with APIs, including managing performance and handling errors.