Web Essentials

From Code to Career

The **Web Essentials** Workshop is a hands-on, beginner-friendly program designed to introduce you to the core technologies of modern web development.  
 In this workshop, you’ll learn how websites are built from the ground up — from designing responsive user interfaces to connecting them with dynamic server-side systems.

You’ll gain both **frontend** and **backend** skills, learning how data flows between browsers, servers, and databases. By the end of the workshop, you’ll understand how real-world web applications work and how to create one yourself.

Technologies Covered

* **Frontend:** HTML5, CSS3, JavaScript
* **Backend:** JSP (Java Server Pages), MySQL Database
* **Tools:** Apache Tomcat, VS Code, MySQL Workbench

Key Takeaways

* Master the building blocks of web pages — HTML, CSS, and JavaScript
* Learn to design interactive and responsive user interfaces
* Understand how to create dynamic web content and handle user input
* Learn how servers work and how JSP connects front-end and back-end
* Get introduced to database connectivity using MySQL
* Experience the complete workflow — from coding to deployment
* Build your own mini web application project

Workshop Outcomes

By the end of this workshop, students will be able to:

* Develop complete **frontend interfaces** using HTML, CSS, and JavaScript
* Create **interactive web pages** that respond to user actions
* Connect their frontend with backend using **JSP and JDBC**
* Manage data storage and retrieval using **MySQL**
* Understand **how a full-stack web application works** end-to-end
* Confidently start building and hosting their own projects

CONTENTS

|  |  |  |
| --- | --- | --- |
| 1 | What is the Internet? |  |
| 2 | How a website is accessed |  |
| 3 | The Client-Server Model |  |
| 4 | The Flow of a Web Application |  |
| 5 | Frontend + Backend = Full Stack |  |
| 6 | Tools & Software Required |  |
| 7 | HTML — Hyper Text Markup Language |  |
| 8 | CSS – Cascading Style Sheets |  |
| 9 | JavaScript (JS) – The Brain of the Web |  |
| 10 | Summary – HTML, CSS, and JavaScript |  |
| 11 | Mini Project 1: Personal Portfolio Web Page |  |
| 12 | Mini Project 2: Login Page (Frontend Only) |  |
| 13 | Mini Project 3: Registration Page |  |
| 14 | Mini Project 4: Dashboard Page (CRUD Ready) |  |

Understanding How the Internet and Web Applications Work

Before you start building web pages, it’s important to understand **how the internet actually works** — how your browser, web servers, and databases communicate to make a website function.

### **1. W**hat is the Internet?

The **Internet** is a global network of connected computers that communicate using a common language called **HTTP (Hypertext Transfer Protocol)**. Every website you visit lives on a **server**, and your **browser (like Chrome or Edge)** is the tool that requests and displays those web pages.

2. How a website is accessed

When you type a website URL (like <www.google.com>):

1. **DNS (Domain Name System)** converts that name into an IP address — something computers understand.
2. The **browser sends a request** to the web server hosting the website.
3. The **web server processes the request** — sometimes fetching data from a database.
4. The **server sends back a response** (HTML, CSS, JS) to the browser.
5. The **browser renders** the page for you to see and interact with.

3. The Client-Server Model

Web applications work on a **Client Server Architecture**.

|  |  |
| --- | --- |
| **Component** | **Description** |
| **Client (Frontend)** | This is the user’s browser — where HTML, CSS, and JavaScript run. It displays content and handles user actions like clicks, forms, etc. |
| **Server (Backend)** | This is where the logic happens. JSP files and Java code process requests, validate data, and decide what information to send back. |
| **Database** | Stores application data (like user details, messages, products, etc.) — usually managed with MySQL. |

4. The Flow of a Web Application

Here’s the typical request-response cycle:

1. **User Action** → A user clicks a button or submits a form on the website.
2. **Request Sent** → The browser sends an HTTP request to the server (e.g., /login.jsp).
3. **Server Processes** → The server (using JSP + Java) reads the request, connects to MySQL, and gets or stores data.
4. **Response Returned** → The server sends back HTML, JSON, or a redirect to another page.
5. **Browser Updates** → The browser updates what the user sees — possibly without reloading (using AJAX or Fetch API).

5. Frontend + Backend = Full Stack

1. A **Frontend Developer** builds what users see and interact with.
2. A **Backend Developer** builds what happens behind the scenes.
3. A **Full-Stack Developer** understands both — connecting the UI with databases and servers.

In this workshop, you’ll explore both sides —

**Frontend:** HTML, CSS, JavaScript  
**Backend:** JSP, MySQL

Example

When you register on a website:

1. You fill out a form (HTML + CSS).
2. JavaScript validates your input.
3. The form is sent to a JSP page.
4. JSP connects to MySQL and stores your data.
5. The server sends a success message back to your browser.

# **6. Tools & Software Required**

To build and run your complete web application, you’ll need the following tools installed on your system. Each tool has a specific role in developing, testing, and running frontend and backend applications.

1. Web Browser

**Tool:** Google Chrome  
 **Purpose:** For testing and viewing your web pages.  
🔗 [Download Chrome](https://www.google.com/chrome/)

2. Code Editor

**Tool:** Visual Studio Code (VS Code)  
 **Purpose:** To write and edit your HTML, CSS, JavaScript, and JSP code efficiently.  
🔗 [Download VS Code](https://code.visualstudio.com/Download)

3. Java Development Kit (JDK)

**Version:** Java 17 or above  
**Purpose:** Provides the compiler and runtime environment required for JSP and Tomcat server.  
🔗 [Download Java](https://www.oracle.com/java/technologies/downloads/?er=221886#java17-windows)

4. Apache Tomcat Server

**Version:** Tomcat 11  
**Purpose:** A Java-based web server to run JSP pages and serve dynamic content.  
🔗 [Download Tomcat](https://tomcat.apache.org/download-11.cgi)

5. Integrated Development Environment (IDE)

**Tool:** Eclipse IDE for Enterprise Java Developers  
**Purpose:** Used to build, debug, and deploy JSP and Java web applications.  
🔗 [Download Eclipse](https://www.eclipse.org/downloads/)

6. Database Server

**Tool:** MySQL Community Server  
**Purpose:** Stores and manages your web application’s data (users, forms, etc.).  
🔗 [Download MySQL Community Server](https://dev.mysql.com/downloads/mysql/)

7. Database Client

**Tool:** MySQL Workbench  
**Purpose:** GUI tool to create, manage, and query databases easily.  
🔗 [Download MySQL Workbench](https://dev.mysql.com/downloads/workbench/)

8. Runtime Support

**Tool:** Visual C++ Redistributable (latest version)  
**Purpose:** Required for MySQL and other software to run properly on Windows.  
🔗 [Download Visual C++ Redistributable](https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170)

Setup Order

1. Install **Chrome** and **VS Code**
2. Install **Java JDK 17+**
3. Set up **Apache Tomcat**
4. Install **Eclipse IDE** and configure Tomcat
5. Install **MySQL Server** and **Workbench**
6. Install **Visual C++ Redistributable** (if prompted by MySQL installer)

# **7. HTML — Hyper Text Markup Language**

1. What is HTML?

**HTML → Hyper Text Markup Language**

* **Hyper Text:** means *text that links to other text*.  
   It allows navigation between web pages using hyperlinks (<a> tag).
* **Markup Language:** means *a language that uses tags to define structure and meaning* in a document.

Markup doesn’t perform actions — it tells the browser **what to display and how it’s organized**.

**In short:** HTML is the *skeleton* of a web page.  
It defines how text, images, links, and elements are placed and structured.

2. How HTML Works (Processing Flow)

When you open an HTML file in a browser (like Chrome):

1. The **browser reads** the HTML file from top to bottom.
2. It **interprets the tags** and builds a **DOM (Document Object Model)** — a tree-like structure.
3. The **DOM** is then styled using **CSS** and made interactive using **JavaScript**.
4. The browser finally **renders** (displays) the web page visually.

**Example:**

HTML file → Browser reads tags → Creates DOM → CSS styles it → JS makes it interactive

3. Structure of an HTML Document

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>My First Web Page</title>*

*</head>*

*<body>*

*<h1>Welcome to HTML Basics</h1>*

*<p>This is a paragraph inside the body tag. </p>*

*</body>*

*</html>*

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <!DOCTYPE html> | Defines the document type as HTML5 |
| <html> | Root of the page |
| <head> | Contains metadata (title, description, links) |
| <body> | Contains visible page content |

4. Basic HTML Tags and Examples

Headings

Used to create titles and subtitles.

*<h1>Main Title</h1>*

*<h2>Subheading</h2>*

*<h3>Section Heading</h3>*

Paragraph

*<p>This is a simple paragraph describing HTML basics.</p>*

Links

*<a href="https://www.google.com" target="\_blank">Visit Google</a>*

Image

*<img src="image.jpg" alt="Example Image" width="300">*

Lists

|  |  |
| --- | --- |
| *<ul>*  *<li>HTML</li>*  *<li>CSS</li>*  *<li>JavaScript</li>*  *</ul>* | *<ol>*  *<li>Learn HTML</li>*  *<li>Practice CSS</li>*  *<li>Master JS</li>*  *</ol>* |

Division & Span

*<div style="background-color: lightgray; padding: 10px;">*

*<h3>Div Example</h3>*

*<p>This is inside a div block.</p>*

*</div>*

*<p>This is <span style="color: red;">important</span> text.</p>*

5. Semantic HTML5 Tags

HTML5 introduced **semantic elements** that describe the meaning of content rather than just its appearance.

This helps search engines and assistive technologies understand your page better.

|  |  |  |
| --- | --- | --- |
| **Tag** | **Purpose** | **Example** |
| *<header>* | Top section or site header | *<header><h1>Website Title</h1></header>* |
| *<nav>* | Navigation links | *`Home* |
| *<section>* | Thematic grouping of content | *<section><h2>About Us</h2></section>* |
| *<article>* | Self-contained content | *<article><h3>Blog Post</h3><p>Content...</p></article>* |
| *<aside>* | Side info or related links | *<aside>Related Posts</aside>* |
| *<footer>* | Footer or copyright info | *<footer>© 2025 Web Essentials</footer>* |
| *<main>* | Main page content | *<main><p>Central content goes here</p></main>* |
| *<figure> & <figcaption>* | For images with captions | *<figure><img src="pic.jpg"><figcaption>Workshop Image</figcaption></figure>* |

6. Table Tags

*<table border="1">*

*<tr>*

*<th>Name</th>*

*<th>Course</th>*

*</tr>*

*<tr>*

*<td>John</td>*

*<td>Web Development</td>*

*</tr>*

*</table>*

7. Form Tags

Forms collect user input and send data to a backend page (like JSP).

*<form action="submit.jsp" method="post">*

*<label for="user">Username:</label>*

*<input type="text" id="user" name="username" required><br>*

*<label for="email">Email:</label>*

*<input type="email" id="email" name="email"><br>*

*<input type="submit" value="Submit">*

*</form>*

8. Multimedia Tags

*<video controls width="300">*

*<source src="intro.mp4" type="video/mp4">*

*Your browser does not support the video tag.*

*</video>*

*<audio controls>*

*<source src="music.mp3" type="audio/mpeg">*

*Your browser does not support the audio element.*

*</audio>*

9. Comments and Meta Tags

*<!-- This is a comment -->*

*<meta charset="UTF-8">*

*<meta name="description" content="Web Essentials Workshop">*

10. Complete Example

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Workshop Demo Page</title>*

*</head>*

*<body>*

*<header>*

*<h1>Welcome to Web Essentials Workshop</h1>*

*<nav>*

*<a href="#">Home</a> | <a href="#">About</a> | <a href="#">Contact</a>*

*</nav>*

*</header>*

*<main>*

*<section>*

*<h2>Introduction to HTML</h2>*

*<p>HTML defines the structure of every webpage you see on the internet.</p>*

*</section>*

*<article>*

*<h3>Workshop Goals</h3>*

*<ul>*

*<li>Learn HTML5 and semantic elements</li>*

*<li>Understand CSS and styling</li>*

*<li>Build your first full web project</li>*

*</ul>*

*</article>*

*</main>*

*<footer>*

*<p>© 2025 Web Essentials Workshop</p>*

*</footer>*

*</body>*

*</html>*

# **8. CSS – Cascading Style Sheets**

**CSS** stands for **Cascading Style Sheets**.

1. What is CSS?

CSS is a **style sheet language** used to describe the **presentation** (look and feel) of a webpage written in HTML.

It defines:

* How elements should be displayed (color, size, position, etc.)
* Layouts (grid, flexbox)
* Responsiveness (different screen sizes)
* Animations and visual effects

2. Meaning of “Cascading” and “Style Sheets”

**Cascading** → Means **priority order** is followed when multiple styles are applied to the same element.

The order of priority is:

* Inline styles (highest priority)
* Internal styles (in <style> tag)
* External styles (in .css file)
* Browser default styles (lowest priority)

**Style Sheets** → A set of rules that define how elements are presented.

3. How CSS Works

When a browser loads a webpage:

1. The **HTML** provides the **structure and content**.
2. The **CSS** provides the **style and layout**.
3. The browser applies CSS styles to HTML elements and renders the styled page.

4. Ways to Add CSS to HTML

1. Inline CSS

CSS applied directly inside an HTML tag using the style attribute.

*<p style="color: blue; font-size: 18px;">This is inline CSS.</p>*

2. Internal CSS

CSS written inside a <style> tag within the <head> section of the HTML.

*<head>*

*<style>*

*p {*

*color: green;*

*font-size: 18px;*

*}*

*</style>*

*</head>*

3. External CSS

CSS written in a separate .css file and linked to the HTML.

*<!-- index.html -->*

*<link rel="stylesheet" href="styles.css">*

*<!-- styles.css -->*

*p {*

*color: red;*

*font-size: 18px;*

*}*

5. Basic Syntax of CSS

*selector {*

*property: value;*

*}*

**Example:**

*h1 {*

*color: darkblue;*

*text-align: center;*

*}*

* **Selector** → Targets HTML elements (e.g., h1, .class, #id)
* **Property** → Defines what you want to style (e.g., color, margin)
* **Value** → Specifies the style setting (e.g., blue, 20px)

6. Types of CSS Selectors

|  |  |  |
| --- | --- | --- |
| **Selector Type** | **Example** | **Description** |
| Element Selector | *p {}* | Selects all <p> tags |
| Class Selector | *.title {}* | Selects all elements with class “title” |
| ID Selector | *#main {}* | Selects the element with ID “main” |
| Universal Selector | *\* {}* | Selects all elements |
| Attribute Selector | *input[type="text"] {}* | Selects specific input type |
| Descendant Selector | *div p {}* | Selects <p> inside <div> |

7. CSS Properties and Examples

Text Formatting

*h1 {*

*color: purple;*

*text-align: center;*

*text-transform: uppercase;*

*}*

Background

*body {*

*background-color: lightyellow;*

*background-image: url("bg.jpg");*

*background-size: cover;*

*}*

Box Model (Margin, Border, Padding)

*div {*

*margin: 20px;*

*padding: 15px;*

*border: 2px solid black;*

*}*

Fonts

*p {*

*font-family: 'Arial', sans-serif;*

*font-size: 16px;*

*font-weight: bold;*

*}*

Links & Hover Effect

*a {*

*color: blue;*

*text-decoration: none;*

*}*

*a:hover {*

*color: red;*

*text-decoration: underline;*

*}*

8. CSS Box Model Overview

Each HTML element is a **box** consisting of:

1. **Content**
2. **Padding** (space inside border)
3. **Border**
4. **Margin** (space outside border)

*div {*

*width: 200px;*

*padding: 10px;*

*border: 2px solid black;*

*margin: 15px;*

*}*

9. Positioning and Layout Techniques

Position Property

*div {*

*position: absolute;*

*top: 20px;*

*left: 50px;*

*}*

Flexbox

Used for aligning and distributing space.

*.container {*

*display: flex;*

*justify-content: center;*

*align-items: center;*

*}*

Grid Layout

Used for complex layouts.

*.grid {*

*display: grid;*

*grid-template-columns: repeat(3, 1fr);*

*gap: 10px;*

*}*

10. Responsive Design

CSS allows pages to adapt to different screen sizes using **media queries**.

*@media (max-width: 600px) {*

*body {*

*background-color: lightblue;*

*}*

*}*

11. CSS3 Features

* Rounded corners (border-radius)
* Shadows (box-shadow, text-shadow)
* Transitions & Animations
* Gradient backgrounds
* Flexbox & Grid

Example – Button Animation

*button {*

*background-color: green;*

*color: white;*

*padding: 10px 20px;*

*transition: background-color 0.3s;*

*}*

*button:hover {*

*background-color: darkgreen;*

*}*

# **9. JavaScript (JS) – The Brain of the Web**

1. Definition

**JavaScript** = **“Just-in-time compiled scripting language”**

It is used to **make web pages dynamic and interactive**.

HTML gives structure, CSS gives style, and **JavaScript gives behavior**.

2. Why JavaScript?

* Makes web pages **interactive** (buttons, forms, sliders, menus).
* Used for **client-side validation** before sending data to the backend.
* Can **fetch and send data** to the server (AJAX / Fetch API).
* Can **manipulate HTML and CSS** in real time using the **DOM**.
* Runs on the **browser (client-side)** but can also be used on **servers** (Node.js).

3. How JavaScript Works

When a webpage loads:

1. The browser loads the **HTML**.
2. Then **CSS** is applied.
3. Then **JavaScript** runs line-by-line inside the **browser engine (V8 for Chrome)**.
4. JS interacts with the **DOM** (Document Object Model) to modify elements dynamically.

4. Where to Write JavaScript

(a) Inline Script

*<button onclick="alert('Hello!')">Click Me</button>*

(b) Internal Script

*<script>*

*document.write("Hello from JS!");*

*</script>*

(c) External Script

*<script src="script.js"></script>*

Best practice: Keep JS in a separate .js file for reusability and cleaner code.

5. Basic Syntax and Example

*// This is a comment*

*let name = "Chetan";*

*console.log("Hello " + name);*

**Key Points:**

* let → block-scoped variable
* const → constant (cannot be changed)
* var → old way (function-scoped)
* console.log() → used for debugging/output in browser console

6. Data Types in JavaScript

|  |  |  |
| --- | --- | --- |
| **Type** | **Example** | **Description** |
| String | "Hello" | Text |
| Number | 42 | Numeric value |
| Boolean | true / false | Logical value |
| Object | { name: "John", age: 25 } | Key-value data |
| Array | [1, 2, 3] | List of items |
| Null | null | Empty value |
| Undefined | let x; | Not assigned yet |

7. Operators

|  |  |  |
| --- | --- | --- |
| **Category** | **Example** | **Description** |
| Arithmetic | + - \* / % | Math operations |
| Comparison | == === != > < | Compare values |
| Logical | `&& |  |
| Assignment | = += -= \*= | Assign or modify values |

8. Conditional Statements

*let marks = 85;*

*if (marks > 90) {*

*console.log("Grade A");*

*} else if (marks > 75) {*

*console.log("Grade B");*

*} else {*

*console.log("Grade C");*

*}*

9. Loops

For Loop

*for (let i = 1; i <= 5; i++) {*

*console.log("Number: " + i);*

*}*

While Loop

*let i = 1;*

*while (i <= 5) {*

*console.log(i);*

*i++;*

*}*

10. Functions

*function greet(name) {*

*return "Hello " + name;*

*}*

*console.log(greet("Students"));*

Functions allow code **reusability** and **logic separation**.

11. Objects and Arrays

Object Example

*let student = {*

*name: "Ravi",*

*age: 21,*

*course: "Web Development"*

*};*

*console.log(student.name);*

Array Example

*let fruits = ["Apple", "Banana", "Mango"];*

*console.log(fruits[1]);*

12. DOM Manipulation (Dynamic HTML)

**DOM (Document Object Model)** = Tree-like structure of HTML that JS can access and modify.

Example – Change HTML Content:

*<p id="demo">Old Text</p>*

*<button onclick="changeText()">Click Me</button>*

*<script>*

*function changeText() {*

*document.getElementById("demo").innerHTML = "Text Changed!";*

*}*

*</script>*

Example – Change Style:

*document.getElementById("demo").style.color = "red";*

13. Event Handling

JS can respond to **events** (click, mouseover, keypress, etc.)

*<button id="btn">Click Me</button>*

*<script>*

*document.getElementById("btn").addEventListener("click", function() {*

*alert("Button was clicked!");*

*});*

*</script>*

14. Form Validation (Client-Side)

Before sending data to JSP or backend, validate it using JS.

*<form onsubmit="return validateForm()">*

*<input type="text" id="name" placeholder="Enter name">*

*<input type="submit">*

*</form>*

*<script>*

*function validateForm() {*

*let name = document.getElementById("name").value;*

*if (name === "") {*

*alert("Name cannot be empty!");*

*return false;*

*}*

*return true;*

*}*

*</script>*

15. Working with JSON (JavaScript Object Notation)

JSON is used to **send data** between frontend and backend.

*let student = {*

*name: "Priya",*

*marks: 90*

*};*

*// Convert JS Object to JSON String*

*let jsonData = JSON.stringify(student);*

*// Convert JSON String to JS Object*

*let obj = JSON.parse(jsonData);*

*console.log(obj.name);*

16. Connecting Frontend to Backend (AJAX / Fetch API)

Example: Sending Data to JSP

*<button onclick="sendData()">Send Data</button>*

*<script>*

*function sendData() {*

*fetch("backend.jsp", {*

*method: "POST",*

*headers: { "Content-Type": "application/json" },*

*body: JSON.stringify({ name: "Rahul", age: 22 })*

*})*

*.then(res => res.text())*

*.then(data => console.log("Response from JSP:", data))*

*.catch(err => console.error(err));*

*}*

*</script>*

Here, fetch() sends data to backend.jsp, and JSP can process it using server-side Java.

17. Local Storage – Save Data on Browser

*localStorage.setItem("username", "Chetan");*

*let user = localStorage.getItem("username");*

*console.log(user);*

Used to store small amounts of data **locally** (e.g., user preferences).

18. Modern JavaScript (ES6+) Key Features

|  |  |  |
| --- | --- | --- |
| **Feature** | **Example** | **Use** |
| Template Literals | `Hello ${name}` | Easier string formatting |
| Arrow Functions | const add = (a,b)=>a+b; | Shorter functions |
| Destructuring | const {name, age} = person; | Extract object values easily |
| Spread Operator | [...arr1, ...arr2] | Merge arrays |
| Promise & Async | async/await | Handle async code cleanly |

19. Practical Examples

Change Background Color on Click

*<button onclick="document.body.style.backgroundColor='lightblue'">*

*Change Background*

*</button>*

Add List Items Dynamically

*<ul id="list"></ul>*

*<button onclick="addItem()">Add Item</button>*

*<script>*

*let count = 1;*

*function addItem() {*

*let li = document.createElement("li");*

*li.innerText = "Item " + count++;*

*document.getElementById("list").appendChild(li);*

*}*

*</script>*

# **10. Summary – HTML, CSS, and JavaScript**

1. HTML – The Structure

HTML (**Hyper Text Markup Language**) defines the **content and structure** of a webpage.

Key Takeaways:

* Defines **layout** and **semantic meaning** using tags.
* Common tags: <html>, <head>, <body>, <div>, <header>, <footer>, <form>, <a>, <img>.
* Uses **attributes** (id, class, href, etc.) to add extra meaning.
* HTML5 introduced semantic tags like <section>, <article>, <nav>, <main>, and multimedia tags like <video>, <audio>.

2. CSS – The Design

CSS (**Cascading Style Sheets**) controls the **presentation** and **layout** of web pages.

Key Takeaways:

* Three ways to apply CSS: **Inline**, **Internal**, **External** (recommended).
* CSS controls:  
   → **Color, background, borders, fonts, spacing, animations** → **Layout systems** – Flexbox & Grid  
   → **Responsive Design** using @media queries.
* The **Box Model** (margin, border, padding, content) defines element spacing.

3. JavaScript – The Behavior

JavaScript adds **interactivity**, **logic**, and **dynamic updates**.

Key Takeaways:

* Variables: let, const, var
* Conditional Logic: if, else, switch
* Loops: for, while
* Functions: Reusable code blocks
* **DOM Manipulation**: Dynamically change HTML & CSS
* **Event Handling**: Respond to clicks, keypresses, and form submissions
* **Client-side Validation** before sending data to the server
* **Fetch API** & **JSON** used to interact with backend (JSP/MySQL)

4. How They Work Together

|  |  |  |
| --- | --- | --- |
| **Layer** | **Language** | **Role** |
| Structure | HTML | Defines content |
| Style | CSS | Beautifies and layouts |
| Logic | JavaScript | Makes it interactive |

Flow:

1. HTML creates elements
2. CSS styles them
3. JS controls their behavior and connects them to backend (JSP/MySQL)

# **11. Mini Project 1: Personal Portfolio Web Page**

Goal: Create a single-page personal portfolio using HTML, CSS, and JavaScript.

Key Features:

* Header with name and navigation links (Home, Skills, Projects, Contact)
* About section with profile image and intro
* Projects section using cards or grid layout
* Contact form with JS validation

Structure Example:

Folder: portfolio/  
 Files:

* index.html
* style.css
* script.js

index.html

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>My Portfolio</title>*

*<link rel="stylesheet" href="style.css">*

*</head>*

*<body>*

*<header>*

*<h1>Welcome to My Portfolio</h1>*

*<nav>*

*<a href="#about">About</a>*

*<a href="#projects">Projects</a>*

*<a href="#contact">Contact</a>*

*</nav>*

*</header>*

*<section id="about">*

*<h2>About Me</h2>*

*<p>Hi! I'm a web developer passionate about building modern web applications.</p>*

*</section>*

*<section id="projects">*

*<h2>My Projects</h2>*

*<div class="project-card">*

*<h3>Portfolio Website</h3>*

*<p>A simple responsive personal website built with HTML, CSS, and JS.</p>*

*</div>*

*<div class="project-card">*

*<h3>Todo App</h3>*

*<p>A dynamic task management app using JavaScript.</p>*

*</div>*

*</section>*

*<section id="contact">*

*<h2>Contact Me</h2>*

*<form onsubmit="return validateForm()">*

*<input type="text" id="name" placeholder="Your Name">*

*<input type="email" id="email" placeholder="Your Email">*

*<textarea id="message" placeholder="Your Message"></textarea>*

*<button type="submit">Send</button>*

*</form>*

*</section>*

*<footer>*

*<p>© 2025 My Portfolio | All Rights Reserved</p>*

*</footer>*

*<script src="script.js"></script>*

*</body>*

*</html>*

style.css

*body {*

*font-family: Arial, sans-serif;*

*margin: 0;*

*background: #f4f4f9;*

*}*

*header {*

*background: #333;*

*color: white;*

*padding: 20px;*

*text-align: center;*

*}*

*nav a {*

*color: white;*

*margin: 0 10px;*

*text-decoration: none;*

*}*

*section {*

*padding: 40px;*

*text-align: center;*

*}*

*.project-card {*

*background: white;*

*padding: 15px;*

*margin: 10px auto;*

*width: 60%;*

*box-shadow: 0 2px 5px rgba(0,0,0,0.2);*

*}*

*form {*

*display: flex;*

*flex-direction: column;*

*align-items: center;*

*}*

*input, textarea {*

*margin: 5px;*

*padding: 10px;*

*width: 60%;*

*border-radius: 5px;*

*border: 1px solid #ccc;*

*}*

*button {*

*background: #333;*

*color: white;*

*padding: 10px 20px;*

*border: none;*

*border-radius: 5px;*

*}*

*button:hover {*

*background: #555;*

*}*

script.js

*function validateForm() {*

*const name = document.getElementById("name").value.trim();*

*const email = document.getElementById("email").value.trim();*

*const message = document.getElementById("message").value.trim();*

*if (!name || !email || !message) {*

*alert("Please fill in all fields!");*

*return false;*

*}*

*alert("Message sent successfully!");*

*return true;*

*}*

# **12. Mini Project 2: Login Page (Frontend Only)**

Goal: Create a simple login page that validates user input using JS.

Steps:

1. Create a form with username and password.
2. Use CSS to center it with a modern UI (box-shadow, border-radius).
3. Add **JavaScript validation** (fields cannot be empty).
4. On success, show “Login successful” message (for now).

Later Integration: This will connect to login.jsp for real authentication.

Folder: login/

index.html

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Login Page</title>*

*<link rel="stylesheet" href="style.css">*

*</head>*

*<body>*

*<div class="login-container">*

*<h2>Login</h2>*

*<form onsubmit="return validateLogin()">*

*<input type="text" id="username" placeholder="Username">*

*<input type="password" id="password" placeholder="Password">*

*<button type="submit">Login</button>*

*</form>*

*</div>*

*<script src="script.js"></script>*

*</body>*

*</html>*

style.css

*body {*

*background: linear-gradient(135deg, #74ABE2, #5563DE);*

*font-family: Arial;*

*display: flex;*

*height: 100vh;*

*align-items: center;*

*justify-content: center;*

*}*

*.login-container {*

*background: white;*

*padding: 30px;*

*border-radius: 10px;*

*box-shadow: 0 3px 8px rgba(0,0,0,0.3);*

*width: 300px;*

*text-align: center;*

*}*

*input {*

*display: block;*

*width: 90%;*

*margin: 10px auto;*

*padding: 10px;*

*}*

*button {*

*background: #5563DE;*

*color: white;*

*border: none;*

*padding: 10px;*

*width: 95%;*

*border-radius: 5px;*

*cursor: pointer;*

*}*

script.js

function validateLogin() {

let user = document.getElementById("username").value;

let pass = document.getElementById("password").value;

if (user === "" || pass === "") {

alert("Please fill in all fields!");

return false;

}

alert("Login successful (dummy)!");

return true;

}

# **13. Mini Project 3: Registration Page**

Goal: Create a registration form that collects user details and performs validation.

Fields:

* Name, Email, Password, Confirm Password
* Gender (radio buttons), Interests (checkboxes)

JS Tasks:

* Validate that password and confirm password match.
* Check that no field is empty.
* Store form data temporarily using localStorage.

Later Integration: This form will connect to register.jsp to insert user data into MySQL.

Folder: register/

index.html

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Register</title>*

*<link rel="stylesheet" href="style.css">*

*</head>*

*<body>*

*<div class="register-box">*

*<h2>Create Account</h2>*

*<form onsubmit="return registerUser()">*

*<input type="text" id="name" placeholder="Full Name">*

*<input type="email" id="email" placeholder="Email">*

*<input type="password" id="password" placeholder="Password">*

*<input type="password" id="confirm" placeholder="Confirm Password">*

*<button type="submit">Register</button>*

*</form>*

*</div>*

*<script src="script.js"></script>*

*</body>*

*</html>*

style.css

*body {*

*font-family: Arial;*

*background: #f4f4f9;*

*display: flex;*

*align-items: center;*

*justify-content: center;*

*height: 100vh;*

*}*

*.register-box {*

*background: white;*

*padding: 30px;*

*border-radius: 8px;*

*box-shadow: 0 2px 10px rgba(0,0,0,0.2);*

*width: 300px;*

*text-align: center;*

*}*

*input {*

*display: block;*

*margin: 10px auto;*

*padding: 10px;*

*width: 90%;*

*}*

*button {*

*background: #333;*

*color: white;*

*padding: 10px;*

*border: none;*

*border-radius: 5px;*

*width: 95%;*

*}*

script.js

*function registerUser() {*

*let name = document.getElementById("name").value;*

*let email = document.getElementById("email").value;*

*let pass = document.getElementById("password").value;*

*let confirm = document.getElementById("confirm").value;*

*if (!name || !email || !pass || !confirm) {*

*alert("Please fill all fields!");*

*return false;*

*}*

*if (pass !== confirm) {*

*alert("Passwords do not match!");*

*return false;*

*}*

*localStorage.setItem("user", JSON.stringify({ name, email }));*

*alert("Registration Successful!");*

*return true;*

*}*

# **14. Mini Project 4: Dashboard Page (CRUD Ready)**

Goal: Create a dashboard where users can view and manage data (like tasks, notes, or contacts).

Features:

* Display data in a table.
* Add, Edit, and Delete rows using JS.
* Store temporary data in browser’s localStorage.

Folder: dashboard/  
 Files:

* index.html
* style.css
* script.js

index.html

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Dashboard - CRUD Operations</title>*

*<link rel="stylesheet" href="style.css">*

*</head>*

*<body>*

*<h1>User Dashboard (CRUD)</h1>*

*<div class="form-box">*

*<input type="text" id="username" placeholder="Enter username">*

*<button id="addBtn" onclick="addUser()">Add User</button>*

*<button id="updateBtn" onclick="updateUser()" style="display:none;">Update User</button>*

*</div>*

*<table>*

*<thead>*

*<tr>*

*<th>#</th>*

*<th>Username</th>*

*<th>Actions</th>*

*</tr>*

*</thead>*

*<tbody id="userTable"></tbody>*

*</table>*

*<script src="script.js"></script>*

*</body>*

*</html>*

style.css

*body {*

*font-family: Arial, sans-serif;*

*background: #f4f4f9;*

*text-align: center;*

*padding: 30px;*

*}*

*h1 {*

*color: #333;*

*}*

*.form-box {*

*margin-bottom: 20px;*

*}*

*input {*

*padding: 10px;*

*width: 200px;*

*margin-right: 5px;*

*border: 1px solid #ccc;*

*border-radius: 5px;*

*}*

*button {*

*padding: 10px 15px;*

*border: none;*

*background-color: #007BFF;*

*color: white;*

*border-radius: 5px;*

*cursor: pointer;*

*}*

*button:hover {*

*background-color: #0056b3;*

*}*

*table {*

*margin: 20px auto;*

*border-collapse: collapse;*

*width: 70%;*

*background: white;*

*box-shadow: 0 2px 8px rgba(0,0,0,0.1);*

*}*

*th, td {*

*border: 1px solid #ddd;*

*padding: 10px;*

*}*

*th {*

*background: #007BFF;*

*color: white;*

*}*

*.action-btn {*

*padding: 5px 10px;*

*margin: 0 2px;*

*border: none;*

*border-radius: 3px;*

*cursor: pointer;*

*}*

*.edit {*

*background-color: #28a745;*

*color: white;*

*}*

*.delete {*

*background-color: #dc3545;*

*color: white;*

*}*

*.edit:hover {*

*background-color: #218838;*

*}*

*.delete:hover {*

*background-color: #c82333;*

*}*

script.js

*let users = JSON.parse(localStorage.getItem("users")) || [];*

*let editIndex = null;*

*// CREATE*

*function addUser() {*

*const username = document.getElementById("username").value.trim();*

*if (!username) {*

*alert("Please enter a username!");*

*return;*

*}*

*users.push(username);*

*localStorage.setItem("users", JSON.stringify(users));*

*document.getElementById("username").value = "";*

*displayUsers();*

*}*

*// READ*

*function displayUsers() {*

*const table = document.getElementById("userTable");*

*table.innerHTML = "";*

*users.forEach((user, index) => {*

*table.innerHTML += `*

*<tr>*

*<td>${index + 1}</td>*

*<td>${user}</td>*

*<td>*

*<button class="action-btn edit" onclick="editUser(${index})">Edit</button>*

*<button class="action-btn delete" onclick="deleteUser(${index})">Delete</button>*

*</td>*

*</tr>*

*`;*

*});*

*}*

*// UPDATE*

*function editUser(index) {*

*document.getElementById("username").value = users[index];*

*document.getElementById("addBtn").style.display = "none";*

*document.getElementById("updateBtn").style.display = "inline-block";*

*editIndex = index;*

*}*

*function updateUser() {*

*const updatedName = document.getElementById("username").value.trim();*

*if (!updatedName) {*

*alert("Username cannot be empty!");*

*return;*

*}*

*users[editIndex] = updatedName;*

*localStorage.setItem("users", JSON.stringify(users));*

*document.getElementById("username").value = "";*

*document.getElementById("addBtn").style.display = "inline-block";*

*document.getElementById("updateBtn").style.display = "none";*

*editIndex = null;*

*displayUsers();*

*}*

*// DELETE*

*function deleteUser(index) {*

*if (confirm("Are you sure you want to delete this user?")) {*

*users.splice(index, 1);*

*localStorage.setItem("users", JSON.stringify(users));*

*displayUsers();*

*}*

*}*

*displayUsers();*

How This Works

|  |  |
| --- | --- |
| **Operation** | **Description** |
| **Create** | Adds a new username into localStorage |
| **Read** | Displays all users in a dynamic table |
| **Update** | Loads selected user into input → updates name |
| **Delete** | Removes selected user from localStorage |

Future JSP Integration Plan

|  |  |  |
| --- | --- | --- |
| **Frontend (Current)** | **JSP Backend (Future)** | **Function** |
| addUser() | addUser.jsp | Inserts new user into DB |
| displayUsers() | fetchUsers.jsp | Retrieves all users |
| updateUser() | updateUser.jsp | Updates user details |
| deleteUser() | deleteUser.jsp | Deletes a user record |