Week2_Day1_Hands_On_Koduru_Ushasree

Problem 1: Personal Notes Saver using LocalStorage (Level-1)

Scenario

You are building a simple web page where users can write daily notes and save them in their browser without using a server.

📌 Requirements

- A textarea for writing notes.
- A **Save** button (using onclick).
- A **Clear** button.
- Notes must: Be stored in localStorage

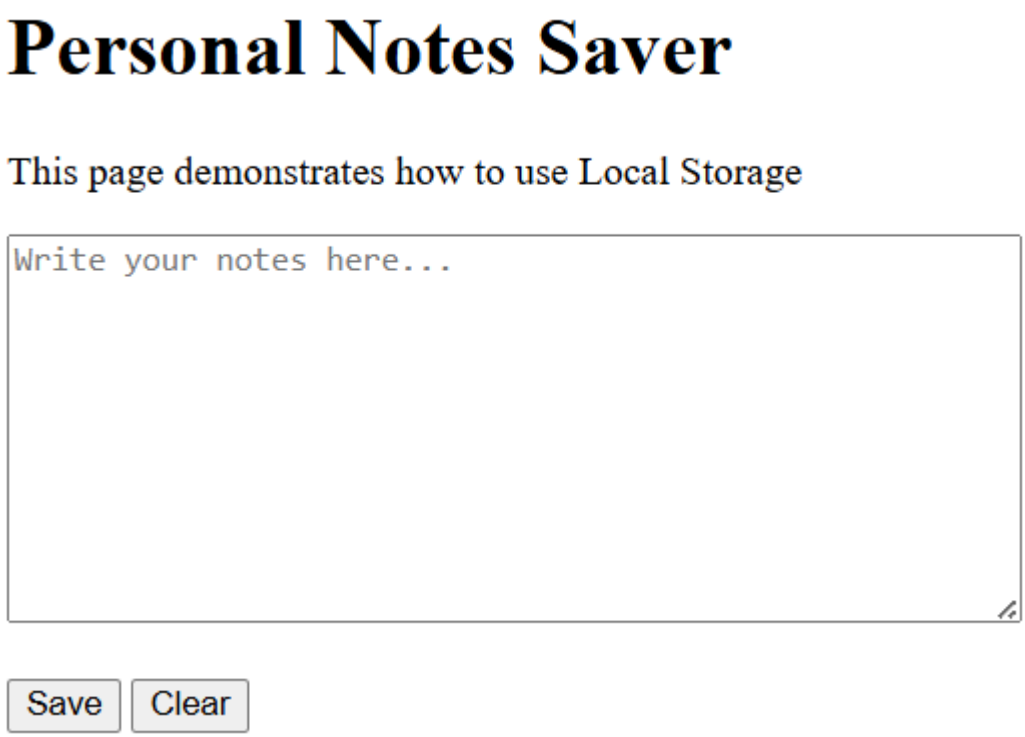
  Automatically load when the page refreshes
- Display stored note on page load.

Code :-

p1.html > html > head

```
<html lang="en">
<body>
    <h1>Personal Notes Saver</h1>
    <p>This page demonstrates how to use Local Storage</p>
    <textarea id="notes" placeholder="Write your notes here..." re></textarea>
    <br><br>
    <button onclick="saveNote()">Save</button>
    <button onclick="clearNote()">Clear</button>

    <script>

        // Save note to localStorage
        function saveNote() {
            var noteText = document.getElementById("notes").value;
            localStorage.setItem("dailyNote", noteText);
            alert("Note Saved Successfully!");
        }

        //load the note stored when page loads
        window.onload = function() {
            var storedNote = localStorage.getItem("dailyNote");
            if (storedNote !== null) {
                document.getElementById("notes").value = storedNote;
            }
        };


        // Clear note from localStorage
        function clearNote() {
            localStorage.removeItem("dailyNote");
            document.getElementById("notes").value = "";
            alert("Note Cleared!");
        }
    </script>
</body>
</html>
```

Output :-



Technical Requirements :-

This application must be built using only HTML and vanilla JavaScript without any backend. It must use inline onclick events for the Save and Clear buttons. Notes should be stored in the browser using localStorage.setItem() as a key-value pair, retrieved using localStorage.getItem() when the page loads, and removed using localStorage.removeItem() when cleared. The stored note must automatically appear after page refresh, which means DOM manipulation and page load handling are required.

Problem 2: Live Form Validation with Events (Level-1)

Scenario

Create a simple registration form that validates user input when fields change.

📌 **Requirements**

- Fields:
    - Name
    - Email
    - Age
- Use:
    - onchange
    - onclick
- Validate:
    - Name cannot be empty
    - Email must contain "@"
    - Age must be greater than 18
- Display validation message dynamically.
- Store valid user data in sessionStorage.

Code :-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .error {
            color: red;
        }
        .success {
            color: green;
        }

        body {
            justify-content: center;
            align-items: center;
        }
    </style>
</head>
<body>
    <h1>Registration Form</h1>
    <!-- Name Field -->
    <label>Name:</label>
    <input type="text" id="name" onchange="validateName()">
    <p id="nameMsg" class="error"></p>

    <br>
    <!-- Email -->
    <label>Email:</label>
    <input type="email" id="email" onchange="validateEmail()">
    <p id="emailMsg" class="error"></p>

    <br>
```

```
<!-- Age Field -->
<label>Age:</label>
<input type="number" id="age" onchange="validateAge()">
<p id="ageMsg" class="error"></p>

<br>

<!-- Submit Button -->
<button onclick="saveData()">Register</button>

<p id="finalMsg"></p>

<script>

// This function runs when the Name field changes
function validateName() {

    // Get the value entered in the name input field
    var name = document.getElementById("name").value;

    // Check if the name is empty (trim removes extra spaces)
    if (name.trim() === "") {

        // Display error message if empty
        document.getElementById("nameMsg").innerHTML = "Name cannot be empty";

    } else {

        // Clear error message if valid
        document.getElementById("nameMsg").innerHTML = "";
    }
}
```

```
// This function runs when the Email field changes
function validateEmail() {

    // Get email value
    var email = document.getElementById("email").value;

    // Check if email contains "@"
    if (!email.includes("@")) {

        // Show error if "@" is missing
        document.getElementById("emailMsg").innerHTML = "Email must contain @";

    } else {

        // Clear error message if valid
        document.getElementById("emailMsg").innerHTML = "";
    }
}
```

```
function validateAge(): void  Age field changes
function validateAge() {

    // Get age value
    var age = document.getElementById("age").value;

    // Check if age is less than or equal to 18 OR empty
    if (age <= 18 || age === "") {

        // Show error message
        document.getElementById("ageMsg").innerHTML = "Age must be greater than 18";

    } else {

        // Clear error message if valid
        document.getElementById("ageMsg").innerHTML = "";
    }
}


// This function runs when Register button is clicked
function saveData() {

    // Run all validations when button is clicked
     validateName();
     validateEmail();
     validateAge();

    // Get all input values
    var name = document.getElementById("name").value;
    var email = document.getElementById("email").value;
    var age = document.getElementById("age").value;
```

```
        // Check if all validations are correct
        if (name.trim() !== "" && email.includes("@") && age > 18) {

            // Store valid data in sessionStorage
            sessionStorage.setItem("userName", name);
            sessionStorage.setItem("userEmail", email);
            sessionStorage.setItem("userAge", age);

            // Show success message
            alert("Registration Successful!");
        }
    }

</script>

</body>
</html>
```

Output :-

Registration Form

Name:

Email:

Age:

Register

Technical Requirements :-

The form must use inline onchange events for input validation and onclick for submission. Validation logic should be written in basic JavaScript to check that the name is not empty, the email contains "@", and the age is greater than 18. If the data is valid, it must be stored using sessionStorage.setItem(). No external libraries are allowed, and validation messages should update dynamically using DOM manipulation.

Problem 3: Location-Based Weather Logger (Level-2)

Scenario
Create a web application that fetches the user's geographic location and stores location history in localStorage.

📌 **Requirements**

- Button: **Get My Location**
- Use:
  navigator.geolocation.getCurrentPosition()
- Display:

  Latitude

Longitude

- Handle:

  Permission denied

  Timeout

  Location unavailable

- Save last 5 location entries in localStorage.
- Display location history on page load.

Code :-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <title>Location Logger</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 40px;
        }

        button {
            padding: 10px 20px;
            background-color: #4CAF50;
            color: white;
            border: none;
            cursor: pointer;
        }

        button:hover {
            background-color: #45a049;
        }

        .error {
            color: red;
        }

        .history {
            margin-top: 20px;
        }
    </style>
```

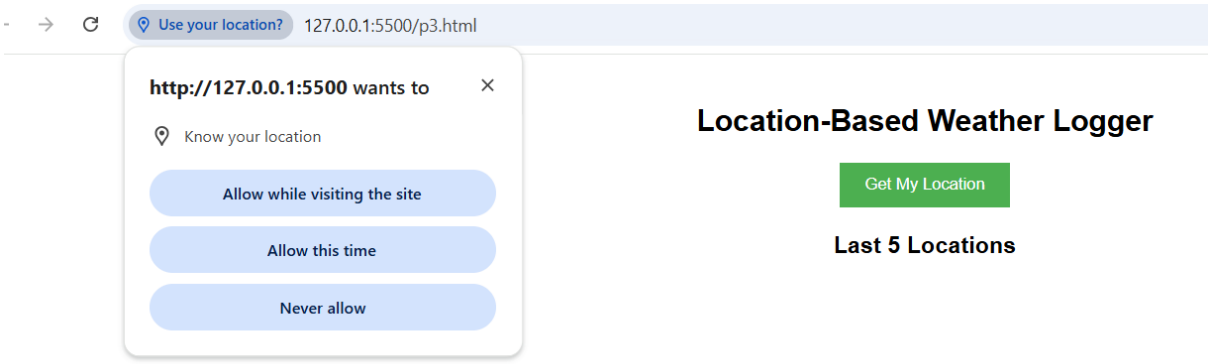
```
<body>
</body>
 <h2>Location-Based Weather Logger</h2>

    <!-- Button with inline onclick -->
    <button onclick="getLocation()">Get My Location</button>

    <p id="output"></p>

    <div class="history">
        <h3>Last 5 Locations</h3>
        <ul id="historyList"></ul>
    </div>

<script>

    // This function runs when button is clicked
    function getLocation() {

        // Check if browser supports Geolocation
        if (navigator.geolocation) {

            // Request current position
            navigator.geolocation.getCurrentPosition(
                showPosition,   // Success callback
                showError,      // Error callback
                { timeout: 10000 } // 10 seconds timeout
            );

        } else {
            document.getElementById("output").innerHTML =
                "Geolocation is not supported by this browser.";
        }
    }
```

```
  // Success callback function
  function showPosition(position) {

      var latitude = position.coords.latitude;
      var longitude = position.coords.longitude;

      // Display latitude and longitude
      document.getElementById("output").innerHTML =
          "Latitude: " + latitude + "<br>Longitude: " + longitude;

      // Create location object with timestamp
      var locationData = {
          lat: latitude,
          long: longitude,
          time: new Date().toLocaleString()
      };
```

```
// Get existing data from localStorage
var history = localStorage.getItem("locations");

// Convert string back to array
history = history ? JSON.parse(history) : [];

// Add new location at beginning
history.unshift(locationData);

// Keep only last 5 entries
if (history.length > 5) {
    history.pop();
}

// Convert array to JSON string and store
localStorage.setItem("locations", JSON.stringify(history));

// Refresh displayed history
displayHistory();
```

```
    // Display history on page load
    function displayHistory() {

        var history = localStorage.getItem("locations");
        history = history ? JSON.parse(history) : [];

        var list = document.getElementById("historyList");
        list.innerHTML = "";

        history.forEach(function(loc) {

            var li = document.createElement("li");
            li.innerHTML =
                "Lat: " + loc.lat +
                ", Long: " + loc.long +
                " (" + loc.time + ")";

            list.appendChild(li);
        });
    }

    // Automatically show stored history when page loads
    window.onload = displayHistory;

</script>
</html>
```

Output :-



Technical Requirements :-

The application must use navigator.geolocation.getCurrentPosition() with both success and error callback functions to handle permission denied, timeout, and location unavailable errors. The button must use inline onclick. Location data (latitude and longitude) should be stored in localStorage as structured data using JSON.stringify() and retrieved using JSON.parse(). Only the last five location records should be maintained, and the location history must be displayed automatically when the page loads.

Problem 4: Mini Expense Tracker using Client-Side Database (Level-2)

Scenario

Develop a client-side expense tracker where users can add, view, and delete expenses using a browser-supported database.

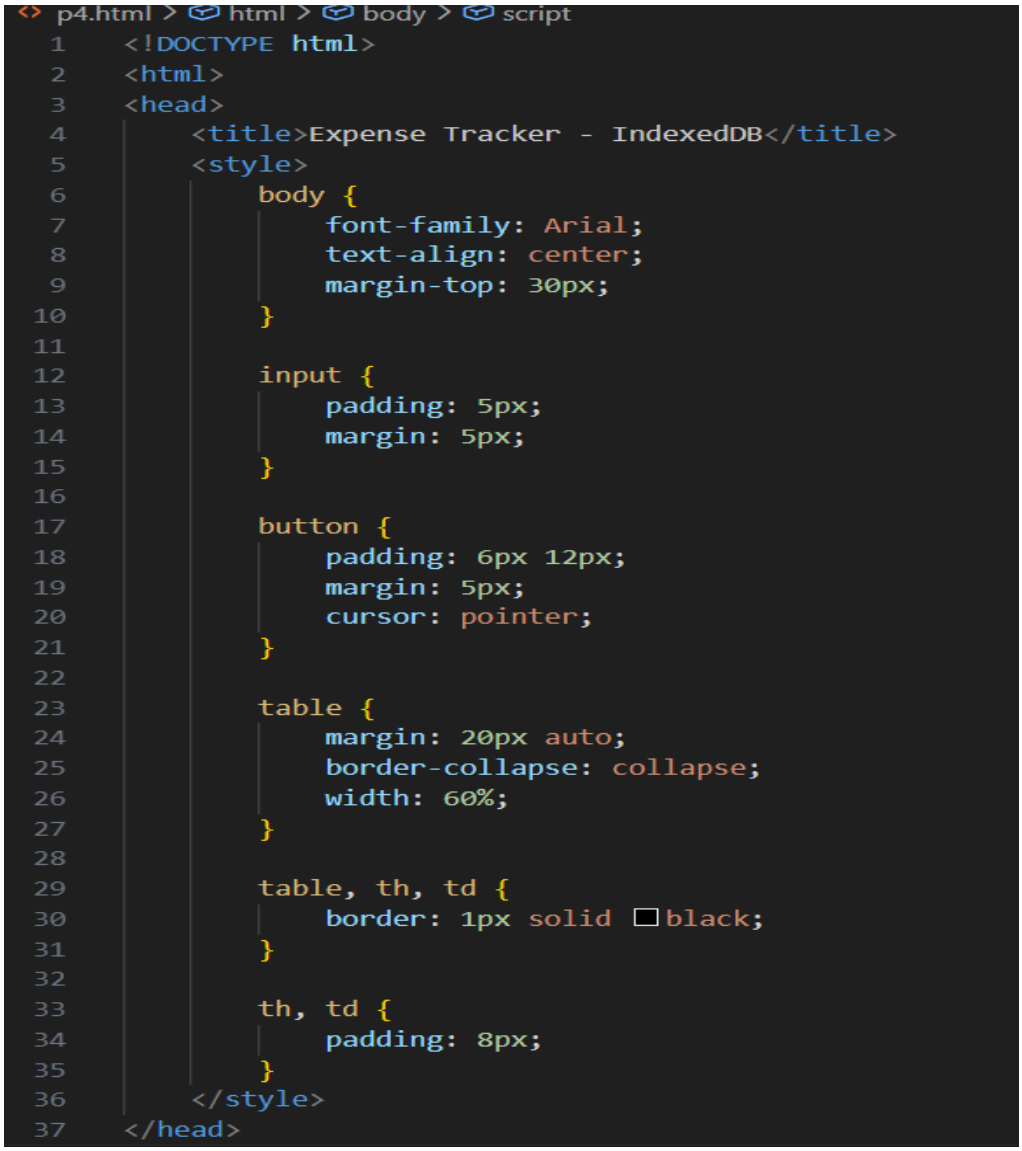
📌 Requirements

1. Fields:
    - Expense Title
    - Amount
    - Date
2. Buttons:
    - Add Expense
    - View Expenses

- Delete Expense

3. Use:
    - Client-side database (Web SQL or IndexedDB)
4. Execute SQL-like queries:
    - CREATE TABLE
    - INSERT
    - SELECT
    - DELETE
5. Maintain transaction handling.
6. Display expense list dynamically.

Code :-



```html
<head>
</head>
<body>

<h2>Mini Expense Tracker (IndexedDB)</h2>

<input type="text" id="title" placeholder="Expense Title">
<input type="number" id="amount" placeholder="Amount">
<input type="date" id="date">

<br>

<button onclick="addExpense()">Add Expense</button>
<button onclick="viewExpenses()">View Expenses</button>
<button onclick="deleteExpense()">Delete Expense</button>

<p id="message"></p>

<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>Title</th>
            <th>Amount</th>
            <th>Date</th>
        </tr>
    </thead>
    <tbody id="expenseList"></tbody>
</table>

<script>

let db;

// Open (or create) Database
let request = indexedDB.open("ExpenseDB", 1);
```

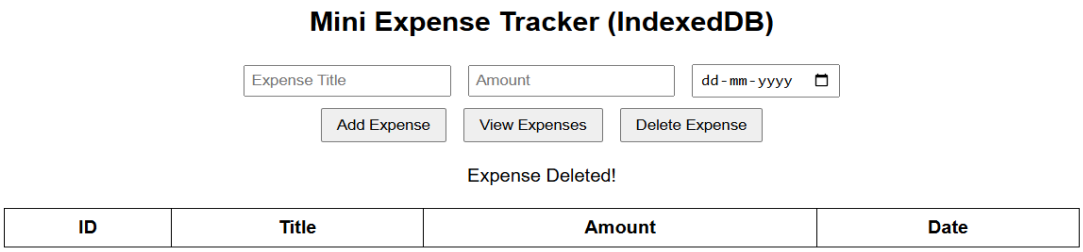
```
// Runs if database is created or version changes
request.onupgradeneeded = function(event) {

    db = event.target.result;

    // Create object store (like table)
    db.createObjectStore("expenses", {
        keyPath: "id",
        autoIncrement: true
    });
};

// Runs when database opens successfully
request.onsuccess = function(event) {
    db = event.target.result;
    viewExpenses(); // Load data automatically
};

// Error opening DB
request.onerror = function() {
    console.log("Database failed to open");
};


//  Add Expense
function addExpense() {

    let title = document.getElementById("title").value;
    let amount = document.getElementById("amount").value;
    let date = document.getElementById("date").value;

    if (title === "" || amount === "" || date === "") {
        document.getElementById("message").innerHTML = "All fields required!";
        return;
```

```
// Start transaction
let transaction = db.transaction(["expenses"], "readwrite");

let store = transaction.objectStore("expenses");

let expense = {
    title: title,
    amount: amount,
    date: date
};

store.add(expense);

transaction.oncomplete = function() {
    document.getElementById("message").innerHTML = "Expense Added!";
    viewExpenses();
};
```

```
    transaction.onerror = function() {
        document.getElementById("message").innerHTML = "Transaction Failed!";
    };
}
// View Expenses
function viewExpenses() {

    let transaction = db.transaction(["expenses"], "readonly");

    let store = transaction.objectStore("expenses");

    let request = store.getAll();

    request.onsuccess = function() {

        let expenses = request.result;
        let output = "";

        expenses.forEach(function(item) {

            output += "<tr>" +
                "<td>" + item.id + "</td>" +
                "<td>" + item.title + "</td>" +
                "<td>" + item.amount + "</td>" +
                "<td>" + item.date + "</td>" +
                "</tr>";
        });

        document.getElementById("expenseList").innerHTML = output;
    };
}
```

```
//  Delete Expense
function deleteExpense() {

    let id = prompt("Enter Expense ID to delete:");
    if (!id) return;
    let transaction = db.transaction(["expenses"], "readwrite");
    let store = transaction.objectStore("expenses");
    store.delete(Number(id));
    transaction.oncomplete = function() {
        document.getElementById("message").innerHTML = "Expense Deleted!";
        viewExpenses();
    };
     transaction.onerror = function() {
        document.getElementById("message").innerHTML = "Delete Failed!";
    };
}
</script>
</body>
</html>
```

Output :-



Technical Requirements :-

This project must use a client-side database such as Web SQL or IndexedDB without any server. It should execute SQL-like commands including CREATE TABLE, INSERT, SELECT, and DELETE within proper database transactions. The application must handle both transaction errors and query errors. All operations must be done using pure HTML and JavaScript, and the expense list should update dynamically on the page after database operations.