DAY-1_Events_HTML5_API_ Hands-On<Rushabh Ramesh Bhusanur>

Problem 1:

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Personal Notes Saver</title>
</head>
<body>

  <h2>Personal Notes Saver</h2>

  <!-- Requirement: A textarea for writing notes -->
  <textarea id="notes" rows="6" cols="50" placeholder="Write your notes here..."></textarea>
  <br><br>

  <!-- Requirement: Save button using inline onclick event -->
  <button onclick="saveNote()">Save</button>

  <!-- Requirement: Clear button -->
  <button onclick="clearNote()">Clear</button>

  <p id="status"></p>

  <script>
    // Requirement: Automatically load note when page refreshes (Page load handling)
    window.onload = function () {
      // Requirement: Use localStorage.getItem() to retrieve stored note
      let savedNote = localStorage.getItem("myNote");  // key-value pair
```

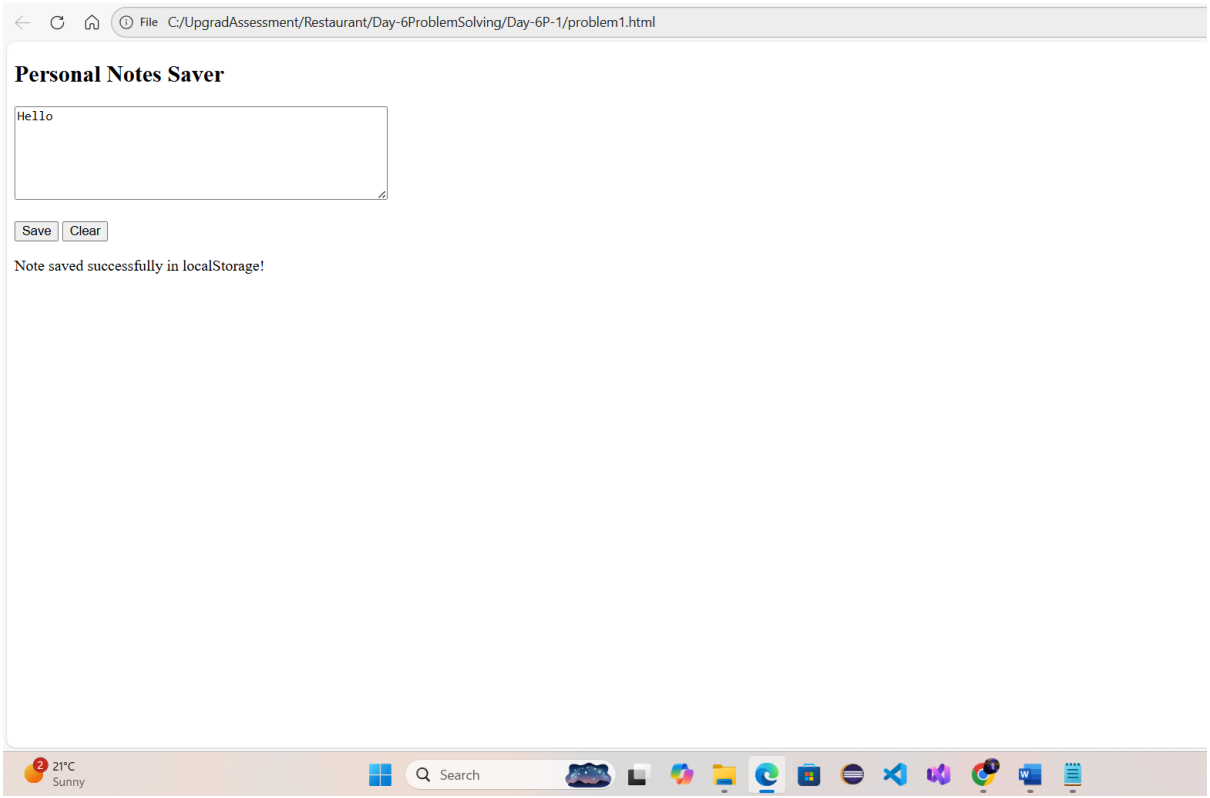
```
    if (savedNote) {
      // Requirement: Display stored note on page load
      document.getElementById("notes").value = savedNote;
      document.getElementById("status").innerText = "Saved note loaded from localStorage.";
    } else {
      document.getElementById("status").innerText = "No saved note found.";
    }
  };

  function saveNote() {
    let noteText = document.getElementById("notes").value;

    // Requirement: Store note in localStorage using setItem (key-value pair)
    localStorage.setItem("myNote", noteText);

    document.getElementById("status").innerText = "Note saved successfully in localStorage!";
  }

  function clearNote() {
    // Requirement: Remove note using localStorage.removeItem()
    localStorage.removeItem("myNote");

    document.getElementById("notes").value = "";
    document.getElementById("status").innerText = "Note cleared from localStorage.";
  }
 </script>

</body>
</html>
```

Code Snapshot:

File Edit View

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Personal Notes Saver</title>
</head>
<body>

  <h2>Personal Notes Saver</h2>

  <!-- Requirement: A textarea for writing notes -->
  <textarea id="notes" rows="6" cols="50" placeholder="Write your notes here..."></textarea>
  <br><br>

  <!-- Requirement: Save button using inline onclick event -->
  <button onclick="saveNote()">Save</button>

  <!-- Requirement: Clear button -->
  <button onclick="clearNote()">Clear</button>

  <p id="status"></p>

  <script>
    // Requirement: Automatically load note when page refreshes (Page load handling)
    window.onload = function () {
      // Requirement: Use localStorage.getItem() to retrieve stored note
      let savedNote = localStorage.getItem("myNote");  // key-value pair

      if (savedNote) {
        // Requirement: Display stored note on page load
        document.getElementById("notes").value = savedNote;
        document.getElementById("status").innerText = "Saved note loaded from localStorage.";
      } else {
        document.getElementById("status").innerText = "No saved note found.";
      }
    };

    function saveNote() {
      let noteText = document.getElementById("notes").value;
```

Output Snapshot:



Code Explanation:

This code creates a simple “Personal Notes Saver” web page where you can write notes in a textarea, save them in the browser using localStorage, and automatically load them again when the page is refreshed. When the page loads, the window.onload function runs and uses localStorage.getItem("myNote") to check if any note was previously saved; if found, it displays that note in the textarea and shows a status message, otherwise it tells the user that no saved note exists. The “Save” button (using an inline onclick event) calls saveNote(), which reads the textarea value and stores it in localStorage as a key–value pair using setItem(). The “Clear” button calls clearNote(), which removes the stored note using removeItem(), clears the textarea, and updates the status message, demonstrating how browser storage can persist data across page reloads.

Problem 2:

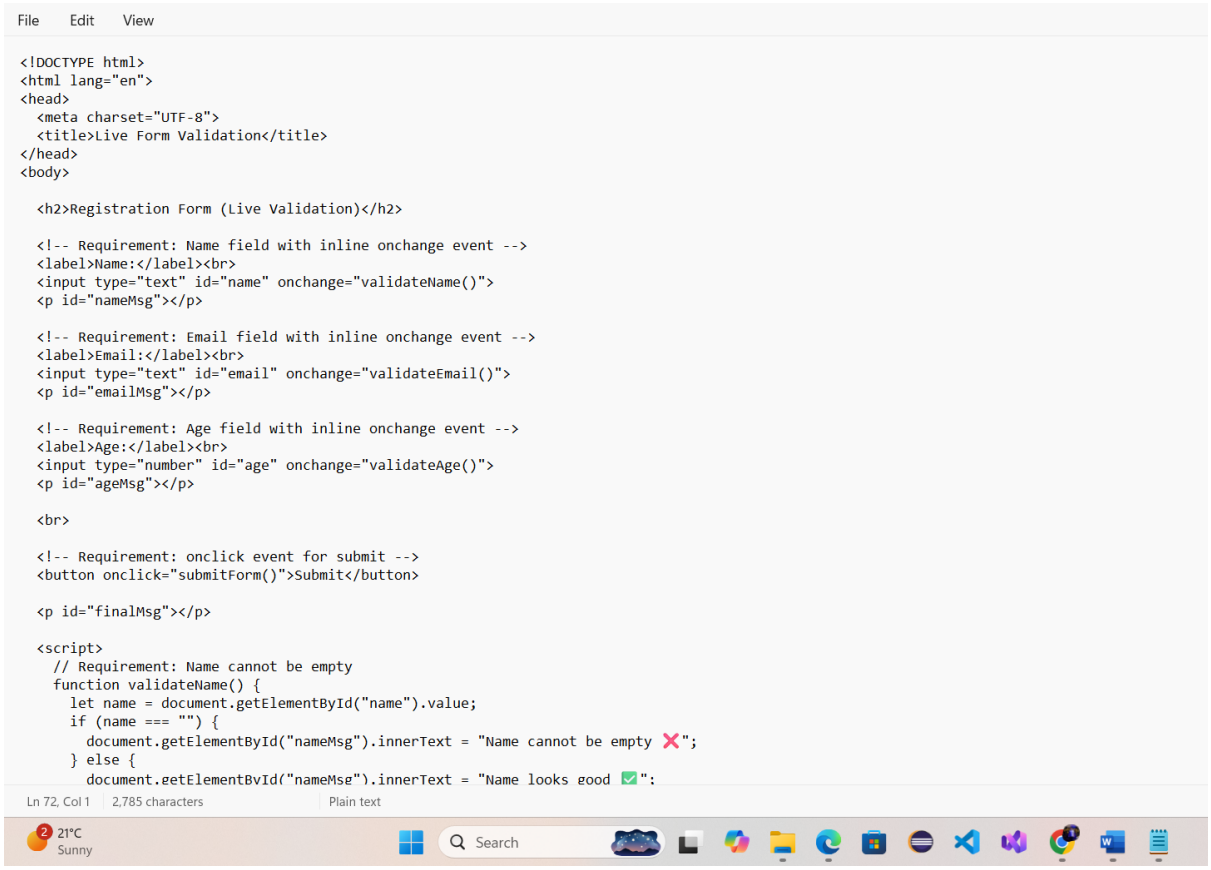
Code :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Live Form Validation</title>
</head>
<body>

  <h2>Registration Form (Live Validation)</h2>

  <!-- Requirement: Name field with inline onchange event -->
  <label>Name:</label><br>
  <input type="text" id="name" onchange="validateName()">
  <p id="nameMsg"></p>

  <!-- Requirement: Email field with inline onchange event -->
  <label>Email:</label><br>
  <input type="text" id="email" onchange="validateEmail()">
  <p id="emailMsg"></p>

  <!-- Requirement: Age field with inline onchange event -->
  <label>Age:</label><br>
  <input type="number" id="age" onchange="validateAge()">
  <p id="ageMsg"></p>

  <br>

  <!-- Requirement: onclick event for submit -->
  <button onclick="submitForm()">Submit</button>
```

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

<script>
 // Requirement: Name cannot be empty
 function validateName() {
  let name = document.getElementById("name").value;
  if (name === "") {
   document.getElementById("nameMsg").innerText = "Name cannot be empty ❌";
  } else {
   document.getElementById("nameMsg").innerText = "Name looks good ✅";
  }
 }

 // Requirement: Email must contain "@"
 function validateEmail() {
  let email = document.getElementById("email").value;
  if (email.includes("@")) {
   document.getElementById("emailMsg").innerText = "Email is valid ✅";
  } else {
   document.getElementById("emailMsg").innerText = "Email must contain @ ❌";
  }
 }

 // Requirement: Age must be greater than 18
 function validateAge() {
  let age = Number(document.getElementById("age").value);
  if (age > 18) {
   document.getElementById("ageMsg").innerText = "Age is valid ✅";
  } else {
   document.getElementById("ageMsg").innerText = "Age must be greater than 18 ❌";
```

```
    }
  }

  // Requirement: onclick submit button + store valid data in sessionStorage
  function submitForm() {
    let name = document.getElementById("name").value;
    let email = document.getElementById("email").value;
    let age = Number(document.getElementById("age").value);

    // Final validation before storing
    if (name !== "" && email.includes("@") && age > 18) {

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

      document.getElementById("finalMsg").innerText =
        "Form submitted successfully! Data stored in sessionStorage ✅";
    } else {
      document.getElementById("finalMsg").innerText =
        "Please correct validation errors before submitting ❌";
    }
  }
 </script>

</body>
</html>
```

Code Snapshot:

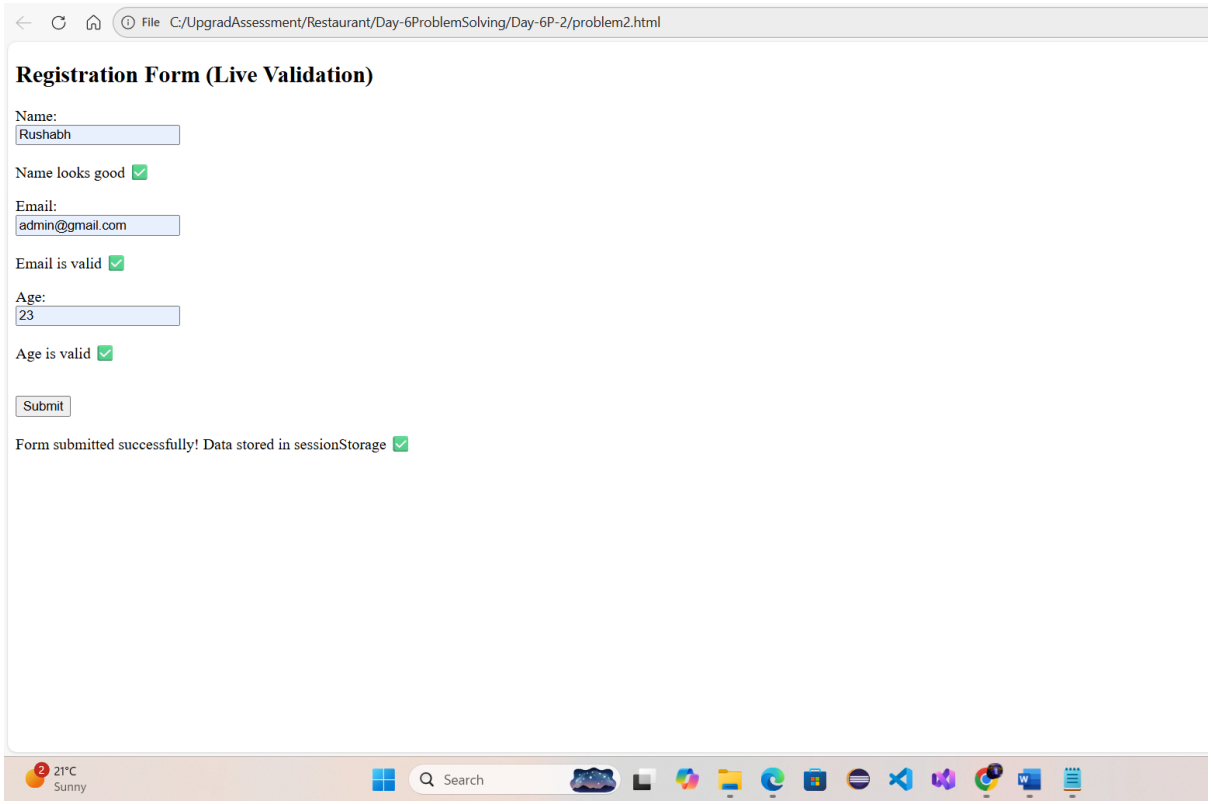


File Edit View

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Live Form Validation</title>
</head>
<body>

  <h2>Registration Form (Live Validation)</h2>

  <!-- Requirement: Name field with inline onchange event -->
  <label>Name:</label><br>
  <input type="text" id="name" onchange="validateName()">
  <p id="nameMsg"></p>

  <!-- Requirement: Email field with inline onchange event -->
  <label>Email:</label><br>
  <input type="text" id="email" onchange="validateEmail()">
  <p id="emailMsg"></p>

  <!-- Requirement: Age field with inline onchange event -->
  <label>Age:</label><br>
  <input type="number" id="age" onchange="validateAge()">
  <p id="ageMsg"></p>

  <br>

  <!-- Requirement: onclick event for submit -->
  <button onclick="submitForm()">Submit</button>

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

  <script>
    // Requirement: Name cannot be empty
    function validateName() {
      let name = document.getElementById("name").value;
      if (name === "") {
        document.getElementById("nameMsg").innerText = "Name cannot be empty ❌";
      } else {
        document.getElementById("nameMsg").innerText = "Name looks good ✅";
```

Ln 72, Col 1 | 2,785 characters | Plain text

Output Snapshot:



Code Explanation:

This code builds a simple registration form with **live validation** using inline onchange and onclick events. As the user fills each field and moves out of it, the corresponding validation function runs: validateName() checks that the name is not empty, validateEmail() ensures the email contains an “@”, and validateAge() verifies the age is greater than 18, showing instant feedback messages below each input. When the user clicks the Submit button, submitForm() performs a final validation check on all fields, and if everything is valid, it stores the user’s name, email, and age in sessionStorage using setItem(), which means the data is available only for the current browser session. If any validation fails, an error message is shown instead, demonstrating how to combine inline event handling, form validation logic, dynamic DOM updates, and session-based web storage in one flow.

Problem 3:

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Location Logger</title>
</head>
<body>

  <h2>Location-Based Logger</h2>

  <!-- Requirement: Button "Get My Location" using inline onclick event -->
  <button onclick="getLocation()">Get My Location</button>

  <h3>Current Location</h3>
  <!-- Requirement: Display Latitude and Longitude -->
  <p id="currentLocation">No location fetched yet.</p>

  <h3>Last 5 Location History</h3>
  <!-- Requirement: Display location history on page load -->
  <ul id="historyList"></ul>

  <p id="status"></p>

  <script>
    // Requirement: Display location history on page load using localStorage + JSON.parse()
    window.onload = function () {
      loadHistory();
    };
```

```
// Requirement: Use navigator.geolocation.getCurrentPosition()
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      showPosition,   // Requirement: Success callback
      showError,      // Requirement: Error callback
      {
        timeout: 10000  // For handling timeout scenario
      }
    );
  } else {
    document.getElementById("status").innerText = "Geolocation is not supported by this browser.";
  }
}

// Requirement: Success callback
function showPosition(position) {
  let lat = position.coords.latitude;
  let lon = position.coords.longitude;

  // Requirement: Display Latitude and Longitude
  document.getElementById("currentLocation").innerText =
    "Latitude: " + lat + " , Longitude: " + lon;

  document.getElementById("status").innerText = "Location fetched successfully ✅";

  // Save location entry with timestamp
  let newEntry = {
    latitude: lat,
    longitude: lon,
    time: new Date().toLocaleString()
```

```
  };

  // Get existing history from localStorage using JSON.parse()
  let history = JSON.parse(localStorage.getItem("locationHistory")) || [];

  // Add new entry to history
  history.unshift(newEntry);

  // Requirement: Save only last 5 location entries
  if (history.length > 5) {
    history = history.slice(0, 5);
  }

  // Requirement: Store data as JSON using JSON.stringify()
  localStorage.setItem("locationHistory", JSON.stringify(history));

  // Update UI
  loadHistory();
}

// Requirement: Error callback handling (permission denied, timeout, unavailable)
function showError(error) {
  let message = "";
  switch (error.code) {
    case error.PERMISSION_DENIED:
      message = "Permission denied by user ❌";
      break;
    case error.POSITION_UNAVAILABLE:
      message = "Location information is unavailable ❌";
      break;
    case error.TIMEOUT:
```

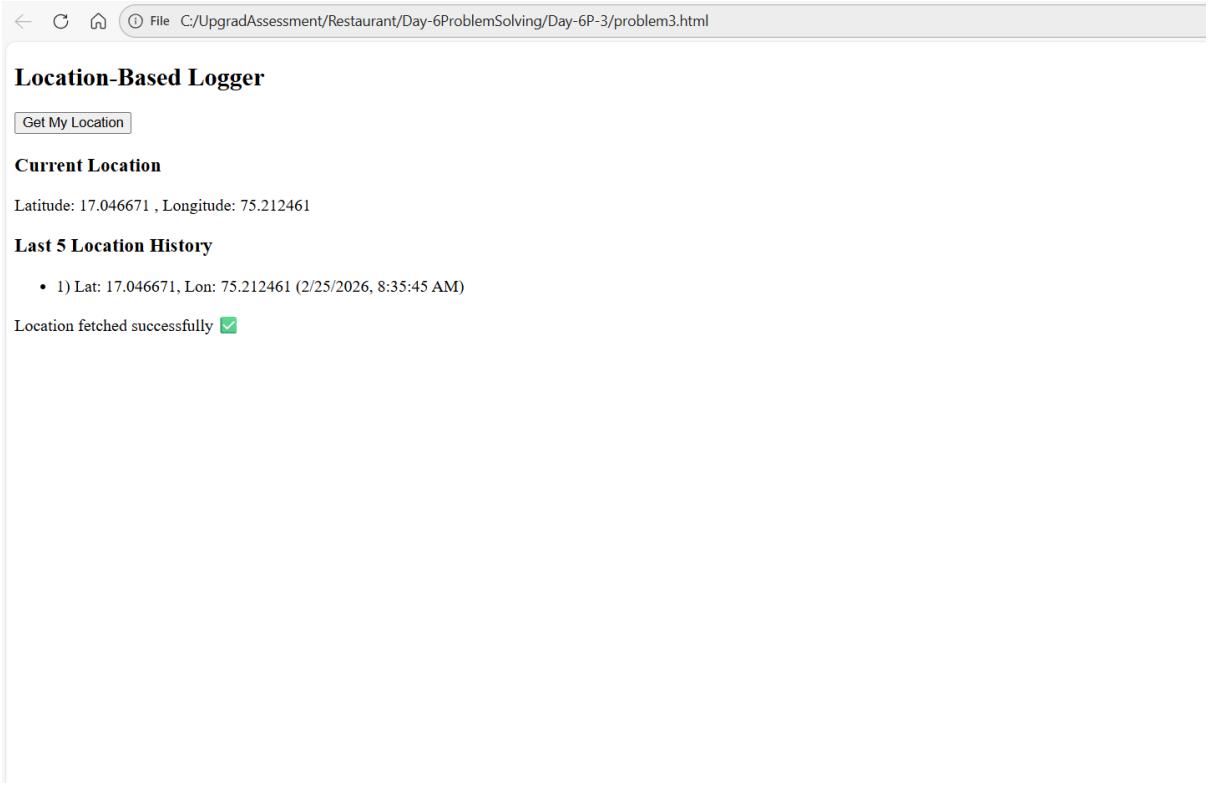
```
        message = "The request to get user location timed out ❌";
        break;
      default:
        message = "An unknown error occurred ❌";
    }
    document.getElementById("status").innerText = message;
  }

  // Load and display location history
  function loadHistory() {
    let history = JSON.parse(localStorage.getItem("locationHistory")) || [];
    let list = document.getElementById("historyList");
    list.innerHTML = "";

    history.forEach(function (item, index) {
      let li = document.createElement("li");
      li.innerText =
        (index + 1) +
        ") Lat: " + item.latitude +
        ", Lon: " + item.longitude +
        " (" + item.time + ")";
      list.appendChild(li);
    });
  }
 </script>

</body>
</html>
```

Code Snapshot:

File Edit View

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Location Logger</title>
</head>
<body>

  <h2>Location-Based Logger</h2>

  <!-- Requirement: Button "Get My Location" using inline onclick event -->
  <button onclick="getLocation()">Get My Location</button>

  <h3>Current Location</h3>
  <!-- Requirement: Display Latitude and Longitude -->
  <p id="currentLocation">No location fetched yet.</p>

  <h3>Last 5 Location History</h3>
  <!-- Requirement: Display location history on page load -->
  <ul id="historyList"></ul>

  <p id="status"></p>

  <script>
    // Requirement: Display location history on page load using localStorage + JSON.parse()
    window.onload = function () {
      loadHistory();
    };

    // Requirement: Use navigator.geolocation.getCurrentPosition()
    function getLocation() {
      if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(
          showPosition,   // Requirement: Success callback
          showError,      // Requirement: Error callback
          {
            timeout: 10000  // For handling timeout scenario
          }
        );
      } else {
```

Output Snapshot:



Code Explanation:

This code creates a Location-Based Logger that uses the browser's Geolocation API to fetch the user's current latitude and longitude when the "Get My Location" button is clicked and then displays it on the page. The getLocation() function calls navigator.geolocation.getCurrentPosition() with separate success (showPosition) and error (showError) callbacks to handle cases like permission denial, timeout, or unavailable location. On success, the coordinates are shown on the screen, and a new location entry with a timestamp is added to a history array stored in localStorage using JSON.stringify(), while older entries are trimmed so that only the last 5 locations are kept. When the page loads, window.onload calls loadHistory(), which retrieves the stored history using JSON.parse() and dynamically renders it as a list, demonstrating real-world usage of browser APIs, callbacks, error handling, persistent storage, and dynamic DOM updates together.

Problem 4:

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Mini Expense Tracker (SQL-like using IndexedDB)</title>
</head>
<body>

  <h2>Mini Expense Tracker</h2>

  <!-- Requirement: Fields -->
  <label>Expense Title:</label><br>
  <input type="text" id="title"><br><br>

  <label>Amount:</label><br>
  <input type="number" id="amount"><br><br>

  <label>Date:</label><br>
  <input type="date" id="date"><br><br>

  <!-- Requirement: Buttons -->
  <button onclick="addExpense()">Add Expense</button>
  <button onclick="viewExpenses()">View Expenses</button>
  <button onclick="deleteExpense()">Delete Expense by ID</button>

  <br><br>
  <label>Delete ID:</label>
  <input type="number" id="deleteId">
```

```
<h3>Expense List</h3>
<ul id="expenseList"></ul>

<p id="status"></p>

<script>
 let db;

 // Open / Create client-side database
 let request = indexedDB.open("ExpenseDB", 1);

 request.onupgradeneeded = function (event) {
  db = event.target.result;

  // SQL-like: CREATE TABLE expenses (...)
  // IndexedDB equivalent: createObjectStore
  let store = db.createObjectStore("expenses", { keyPath: "id", autoIncrement: true });
 };

 request.onsuccess = function (event) {
  db = event.target.result;
  document.getElementById("status").innerText = "Database ready ✅";
 };

 request.onerror = function () {
  document.getElementById("status").innerText = "Database error ❌";
 };

 // SQL-like: INSERT INTO expenses VALUES (...)
 // IndexedDB equivalent: objectStore.add()
 function addExpense() {
```

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

  let transaction = db.transaction(["expenses"], "readwrite");
  let store = transaction.objectStore("expenses");

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

  let req = store.add(expense);

  req.onsuccess = function () {
    document.getElementById("status").innerText = "Expense inserted (INSERT) ✅";
    viewExpenses();
  };

  req.onerror = function () {
    document.getElementById("status").innerText = "Insert failed ❌";
  };
}

// SQL-like: SELECT * FROM expenses
// IndexedDB equivalent: objectStore.getAll()
function viewExpenses() {
  let transaction = db.transaction(["expenses"], "readonly");
  let store = transaction.objectStore("expenses");

  let req = store.getAll();

  req.onsuccess = function () {
    let list = document.getElementById("expenseList");
```

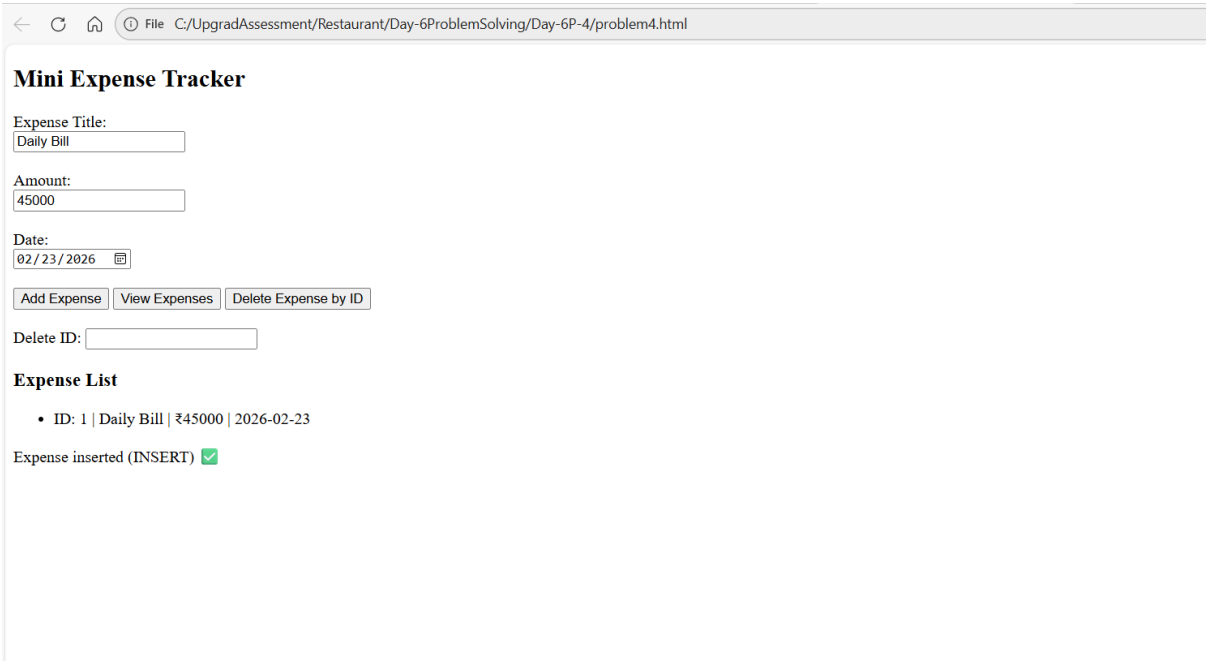
```
  list.innerHTML = "";

  req.result.forEach(function (item) {
    let li = document.createElement("li");
    li.innerText =
      "ID: " + item.id +
      " | " + item.title +
      " | ₹" + item.amount +
      " | " + item.date;
    list.appendChild(li);
  });
 };

 req.onerror = function () {
  document.getElementById("status").innerText = "Select failed ❌";
 };
}

// SQL-like: DELETE FROM expenses WHERE id = ?
// IndexedDB equivalent: objectStore.delete(id)
function deleteExpense() {
 let id = Number(document.getElementById("deleteId").value);

 let transaction = db.transaction(["expenses"], "readwrite");
 let store = transaction.objectStore("expenses");

 let req = store.delete(id);

 req.onsuccess = function () {
  document.getElementById("status").innerText = "Expense deleted (DELETE) ✅";
  viewExpenses();
```

```
    };

    req.onerror = function () {
      document.getElementById("status").innerText = "Delete failed ❌";
    };
  }
</script>

</body>
</html>
```

Code Snapshot:

```
File  Edit  View

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Mini Expense Tracker (SQL-like using IndexedDB)</title>
</head>
<body>

  <h2>Mini Expense Tracker</h2>

  <!-- Requirement: Fields -->
  <label>Expense Title:</label><br>
  <input type="text" id="title"><br><br>

  <label>Amount:</label><br>
  <input type="number" id="amount"><br><br>

  <label>Date:</label><br>
  <input type="date" id="date"><br><br>

  <!-- Requirement: Buttons -->
  <button onclick="addExpense()">Add Expense</button>
  <button onclick="viewExpenses()">View Expenses</button>
  <button onclick="deleteExpense()">Delete Expense by ID</button>

  <br><br>
  <label>Delete ID:</label>
  <input type="number" id="deleteId">

  <h3>Expense List</h3>
  <ul id="expenseList"></ul>

  <p id="status"></p>

  <script>
    let db;

    // Open / Create client-side database
    let request = indexedDB.open("ExpenseDB", 1);
```

Output Snapshot:



Code Explanation:

This code builds a small **Mini Expense Tracker** using the browser's **IndexedDB**, which works like a client-side database and is compared to SQL-style operations. When the page loads, indexedDB.open("ExpenseDB", 1) opens (or creates) a database, and inside onupgradeneeded an object store named expenses is created, which is similar to creating a table in SQL with an auto-incrementing id. The addExpense() function works like an **INSERT** operation by reading the title, amount, and date from the form and adding a new record using objectStore.add(). The viewExpenses() function is similar to **SELECT *** because it fetches all records using getAll() and dynamically displays them in a list on the page. The deleteExpense() function mimics **DELETE FROM ... WHERE id = ?** by removing a specific expense using objectStore.delete(id). Overall, this example demonstrates how SQL-like CRUD operations (CREATE, INSERT, SELECT, DELETE) can be simulated in the browser using IndexedDB with asynchronous callbacks and dynamic DOM updates.