Week2_Day2_Hands_On_Koduru_Ushasree

PROBLEM 1: Student Marks Analyzer (LEVEL-1)

Scenario:

You are developing a small utility for a teacher to analyze student marks stored in an array.

Requirements

- Store student marks in an array.
- Calculate total and average marks using array methods.
- Display pass/fail result based on average.
- Use let/const appropriately.
- Use arrow functions for calculations.
- Display output using template literals.

Technical Constraints

- Must use array methods like reduce() and map().
- Use only ES6+ syntax.
- No external libraries.
- All logic must be inside modular JavaScript file.

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>
    <h1>Student Marks Analyzer</h1>
    <div id="output"></div>
    <script>
        const students = [75,80,95,46,39];
        // Calculate total using reduce()
        const calculateTotal = (arr) => {
            return arr.reduce((total,mark) => total+mark,0);
        };
        // Calculate Average
        const calculateAverage = (arr) => {
            const total = calculateTotal(arr);
            return total/arr.length;
        };
```

```
        //Determine pass or fail
        const getRe (parameter) avg: any
            return avg>= 50 ? "Pass" : "Fail";

        };
        //perform calculations
        const totalMarks = calculateTotal(students);
        const averageMarks = calculateAverage(students);
        const result = getResult(averageMarks);

        //Display output using template literals
        const outputDiv = document.getElementById("output");
        outputDiv.innerHTML =
        `<p><strong>Total Marks:</strong> ${totalMarks}</p>
            <p><strong>Average Marks:</strong> ${averageMarks.toFixed(2)}</p>
            <p><strong>Result:</strong> ${result}</p>`;
    </script>
</body>
</html>
```

Output :-

Student Marks Analyzer

Total Marks: 335

Average Marks: 67.00

Result: Pass

Technical Requirements :-

This program must be written using only ES6+ JavaScript syntax and organized inside a modular JavaScript file. Student marks should be stored in an array, and calculations such as total and average must be done using array methods like reduce() and map() with arrow functions. Variables should be declared using let and const appropriately, and the output must be displayed using template literals. No external libraries are allowed.

PROBLEM 2: Product Cart Summary (LEVEL-1)

Scenario:

Build a simple shopping cart summary system.

Requirements

- Store product objects (name, price, quantity) in an array.
- Calculate total cart value.
- Display formatted invoice using template literals.
- Use arrow functions.
- Export calculation function from a module.

Technical Constraints

- Use map() and reduce().
- Use ES6 modules (export/import).
- No DOM manipulation required.

Code :-

Cart.js

```
2 > JS cart.js > ...
1
2   // Arrow function to calculate total cart value
3   export const calculateTotal = (products) => {
4       return products.reduce((total,product) => {
5           return total + (product.price * product.quantity);
6       },0);
7   };
8
9   // Arrow function to generate invoice lines
10  export const generateInvoice = (products) => {
11      return  products.map(product => {
12        return `
13   Product: ${product.name}
14
15      Price: ₹${product.price}
16      Quantity: ${product.quantity}
17      Subtotal: ₹${product.price * product.quantity}`
18      }).join("\n");
19  };
```

Main.js

```js
import { calculateTotal, generateInvoice } from './cart.js';

// Store product objects in an array
const cart = [
    { name: "Laptop", price: 50000, quantity: 1 },
    { name: "Mouse", price: 500, quantity: 2 },
    { name: "Keyboard", price: 1500, quantity: 1 }
];

// Generate invoice text
const invoice = generateInvoice(cart);

// Calculate total cart value
const totalAmount = calculateTotal(cart);

// Get output element
const outputElement = document.getElementById("output");

//Display formatted invoice using template literals
outputElement.textContent = `
========== INVOICE ==========
${invoice}

Total Cart Value: ₹${totalAmount}
============================`;
```

Index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Product Cart Summary</title>
</head>
<body>

    <!-- Load main.js as module -->
    <h2>Product Cart Summary</h2>
    <pre id="output"></pre>
    <script type="module" src="./main.js"></script>

</body>
</html>
```

Output :-

Product Cart Summary

```
========== INVOICE ==========

 Product: Laptop

    Price: ₹50000
    Quantity: 1
    Subtotal: ₹50000

 Product: Mouse

    Price: ₹500
    Quantity: 2
    Subtotal: ₹1000

 Product: Keyboard

    Price: ₹1500
    Quantity: 1
    Subtotal: ₹1500

Total Cart Value: ₹52500
============================
```

Technical Requirements :-

This application must store product objects (name, price, quantity) inside an array and calculate the total cart value using map() and reduce() with arrow functions. The calculation function must be exported from an ES6 module and imported where needed. The invoice output should be formatted using template literals. No DOM manipulation or external libraries should be used, and the entire logic must follow ES6 module structure.

PROBLEM 3: Weather Data Fetcher (LEVEL-2)

Scenario:

Create an application that fetches weather data asynchronously.

Requirements

- Fetch data from a public API using fetch().

- Implement version using Promises.

- Implement version using async/await.

- Handle errors properly.

- Display formatted weather report.

Technical Constraints

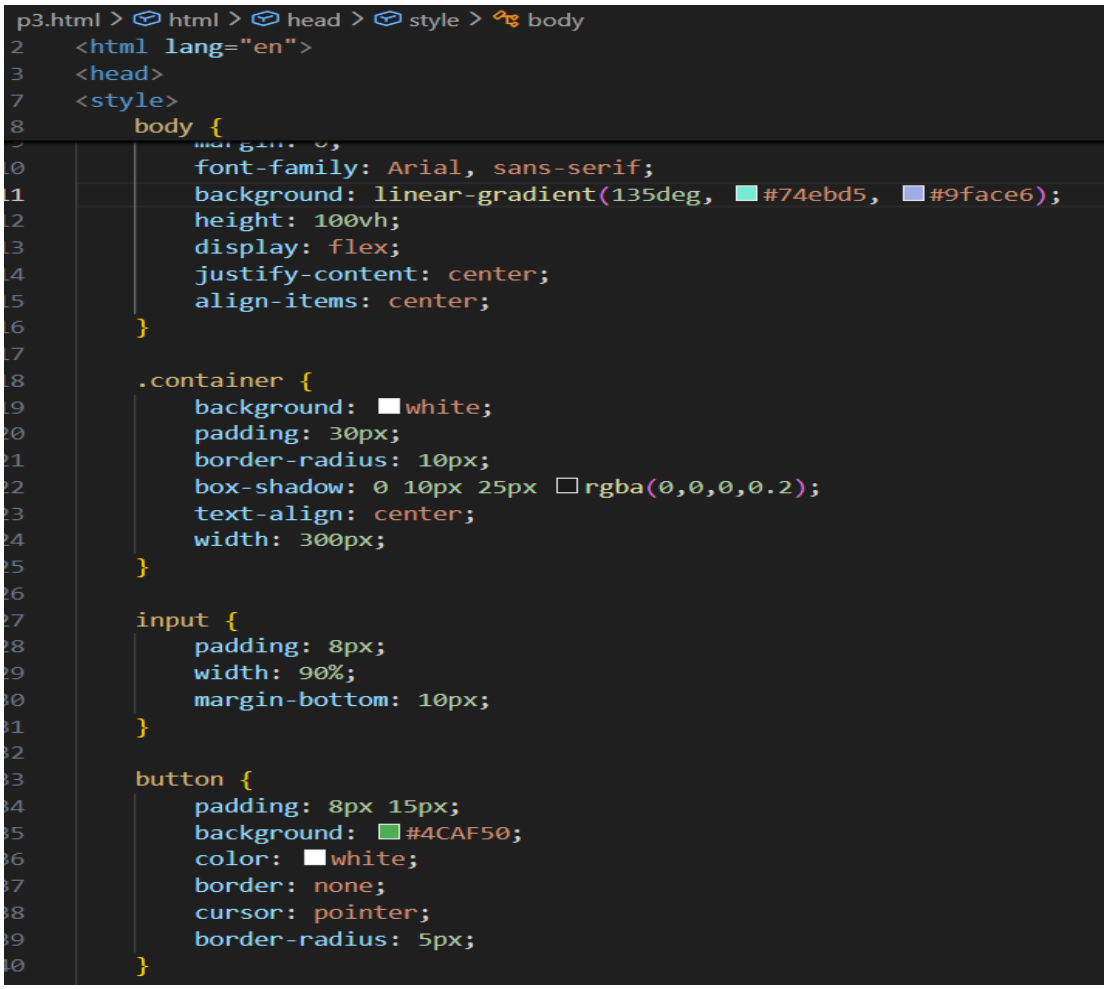
- Use async/await.

- Use template literals.

- Use arrow functions.

- Must include proper error handling using try/catch.

Code :-

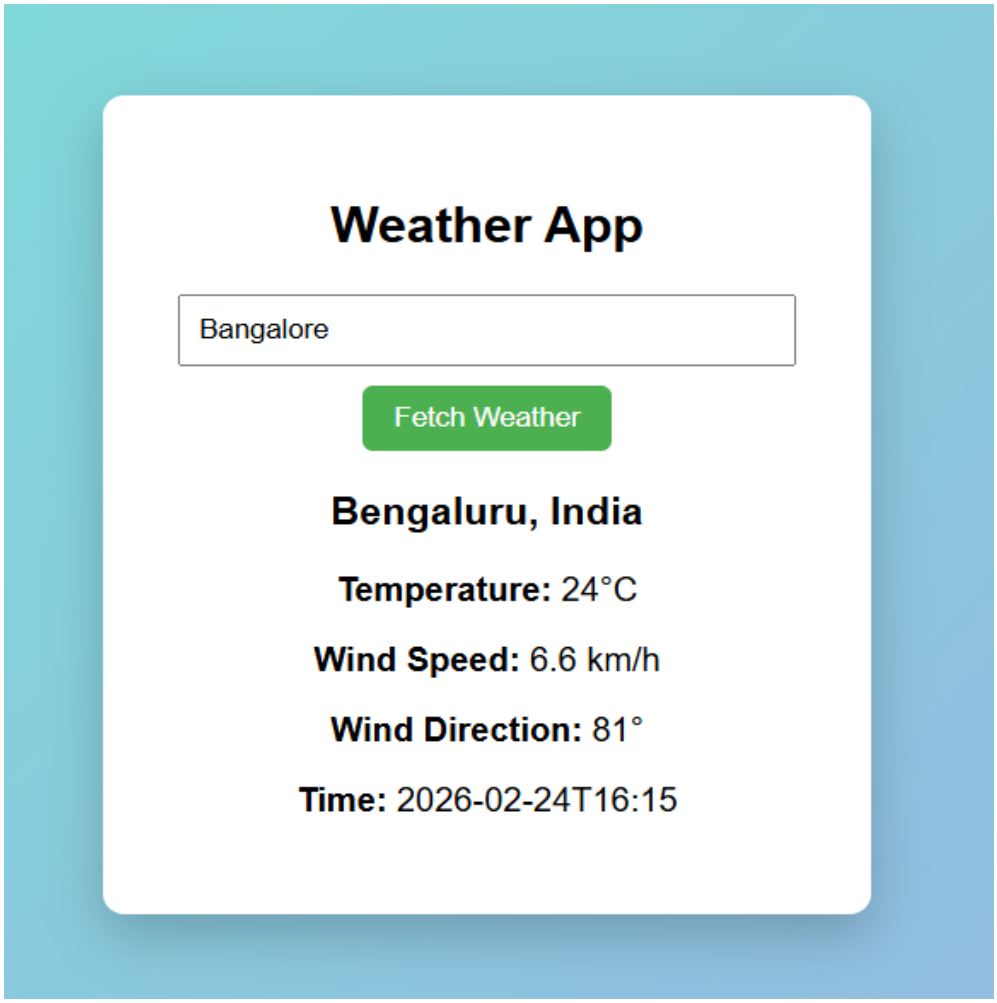


```
<body>
<div class="container">
    <h2>Weather App</h2>

    <input type="text" id="cityInput" placeholder="Enter City (e.g. Bangalore)">
    <button onclick="fetchWeather()">Fetch Weather</button>

    <div id="output"></div>
</div>
<script>
const outputDiv = document.getElementById("output");
// Convert city name to coordinates using Open-Meteo Geocoding API
const fetchWeather = async () => {
    const city = document.getElementById("cityInput").value;
    if (!city) {
        displayError("Please enter a city name");
        return;
    }
    try {

        //  Get coordinates
        const geoResponse = await fetch(`https://geocoding-api.open-meteo.com/v1/search?name=${city}&count=1`);
        if (!geoResponse.ok) {
            throw new Error("City not found");
        }
        const geoData = await geoResponse.json();
        if (!geoData.results) {
            throw new Error("City not found");
        }
        const { latitude, longitude, name, country } = geoData.results[0];
```

```
        //Fetch weather using coordinates
        const weatherResponse = await fetch(`https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitude}&current_weather=tru

        if (!weatherResponse.ok) {
            throw new Error("Failed to fetch weather");
        }

        const weatherData = await weatherResponse.json();

        displayWeather(weatherData.current_weather, name, country);

    } catch (error) {
        displayError(error.message);
    }
};
// Display weather
const displayWeather = (weather, city, country) => {
    outputDiv.innerHTML = `
        <h3>${city}, ${country}</h3>
        <p><strong>Temperature:</strong> ${weather.temperature}°C</p>
        <p><strong>Wind Speed:</strong> ${weather.windspeed} km/h</p>
        <p><strong>Wind Direction:</strong> ${weather.winddirection}°</p>
        <p><strong>Time:</strong> ${weather.time}</p>
    `;
};
```

```
    // Display error
    const displayError = (message) => {
        outputDiv.innerHTML = `
            <p style="color:red;">
                <strong>Error:</strong> ${message}
            </p>
        `;
    };

    </script>

    </body>
    </html>
```

Output :-



Technical Requirements :-

The application must fetch weather data from a public API using the fetch() method. It should include both Promise-based and async/await implementations. Proper error handling must be implemented using try/catch blocks. Arrow functions and template literals should be used throughout the code. The solution must strictly follow modern ES6+ syntax and handle asynchronous operations correctly.

PROBLEM 4: Task Manager with Async Storage (LEVEL-2)

Scenario:

Develop a task manager where tasks are saved and retrieved asynchronously.

Requirements

- Store tasks in an array.
- Create addTask(), deleteTask(), and listTasks() functions.
- Simulate async storage using setTimeout with callbacks.
- Convert callback-based logic into Promises.
- Refactor into async/await version.
- Use ES6 modules.

Technical Constraints

- Must demonstrate callback → promise → async/await evolution.
- Use let/const properly.
- Use arrow functions.
- Use template literals for display

Code :-

p4 > JS taskManager.js > ...

```
// Global task storage
let tasks = [];
//callback
export const addTaskCallback = (task, callback) => {
    setTimeout(() => {
        tasks.push(task);
        callback(`(Callback) Task "${task}" added.`);
    }, 1000);
};
//Promise
export const addTaskPromise = (task) => {
    return new Promise((resolve) => {
        setTimeout(() => {
            tasks.push(task);
            resolve(`(Promise) Task "${task}" added.`);
        }, 1000);
    });
};
//Async/Await
export const addTaskAsync = async (task) => {
    return new Promise((resolve) => {
        setTimeout(() => {
            tasks.push(task);
            resolve(`(Async/Await) Task "${task}" added.`);
        }, 1000);
    });
};

export const deleteTaskAsync = async (task) => {
    return new Promise((resolve) => {
        setTimeout(() => {
            tasks = tasks.filter(t => t !== task);
            resolve(`Task "${task}" deleted.`);
        }, 1000);
    });
};

```

```
export const listTasksAsync = async () => {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(`Current Tasks: ${tasks.join(", ") || "No tasks"}`);
        }, 500);
    });
};
```

Main.js

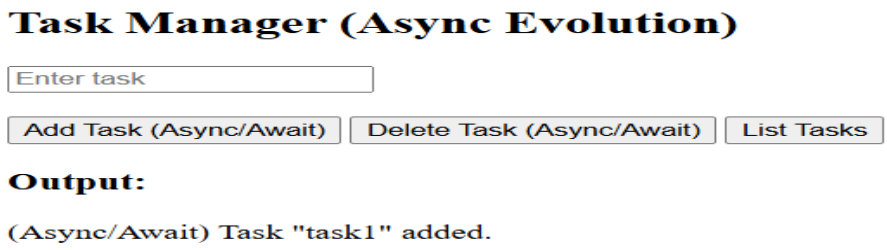
p4 > JS main.js > ...

```
import {
    addTaskAsync,
    deleteTaskAsync,
    listTasksAsync
} from './taskManager.js';

const input = document.getElementById("taskInput");
const output = document.getElementById("output");
// Add Task (Async/Await)
document.getElementById("addBtn").addEventListener("click", async () => {
    const task = input.value.trim();
    if (!task) return;

    const message = await addTaskAsync(task);
    output.innerHTML = `<p>${message}</p>`;
    input.value = "";
});
// Delete Task
document.getElementById("deleteBtn").addEventListener("click", async () => {
    const task = input.value.trim();
    if (!task) return;

    const message = await deleteTaskAsync(task);
    output.innerHTML = `<p>${message}</p>`;
    input.value = "";
});
// List Tasks
document.getElementById("listBtn").addEventListener("click", async () => {
    const message = await listTasksAsync();
    output.innerHTML = `<p>${message}</p>`;
});
```

Index.html

```
04 > index.html > html
  1  <!DOCTYPE html>
  2  <html>
  3  <head>
  4      <title>Async Task Manager</title>
  5  </head>
  6  <body>
  7
  8      <h2>Task Manager (Async Evolution)</h2>
  9
 10      <input type="text" id="taskInput" placeholder="Enter task">
 11
 12      <br><br>
 13
 14      <button id="addBtn">Add Task (Async/Await)</button>
 15      <button id="deleteBtn">Delete Task (Async/Await)</button>
 16      <button id="listBtn">List Tasks</button>
 17
 18      <h3>Output:</h3>
 19      <div id="output"></div>
 20
 21      <!-- Import JS as Module -->
 22      <script type="module" src="main.js"></script>
 23
 24  </body>
 25  </html>
```

Output:



Technical Requirements :-

This project must demonstrate asynchronous JavaScript evolution by first implementing task operations using callbacks with setTimeout, then converting them into Promises, and finally refactoring them using async/await.