Day_2_JavaScript_session-7_hands_On _Uday

1) Problem Statement : developing a small utility for a teacher to analyze student marks stored in an array.

Index.html:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Student Marks Analyzer</title>

  <!-- Linking external CSS -->
  <link rel="stylesheet" href="style.css">
</head>

<body>

  <div class="card">
    <h1>Student Marks Analyzer</h1>
    <div id="result"></div>
  </div>

  <script>

    const marks = [35, 96, 66, 73, 59];

    const getTotal = arr => arr.reduce((sum, mark) => sum + mark, 0);
    const getAverage = arr => getTotal(arr) / arr.length;
    const getResult = avg => avg >= 35 ? "PASS" : "FAIL";

    const total = getTotal(marks);
    const average = getAverage(marks);
    const result = getResult(average);

    const marksList = marks.map((mark, index) =>
      `<div class="subject">Subject ${index + 1}: ${mark}</div>`
    ).join("");

    document.getElementById("result").innerHTML = `
      ${marksList}

      <div class="total">Total Marks : ${total}</div>
      <div class="average">Average Marks : ${average.toFixed(2)}</div>
      <div class="${result === "PASS" ? "pass" : "fail"}">
        Final Result : ${result}
      </div>
    `;

  </script>

</body>
</html>
```

Day_2_JavaScript_session-7_hands_On _Uday

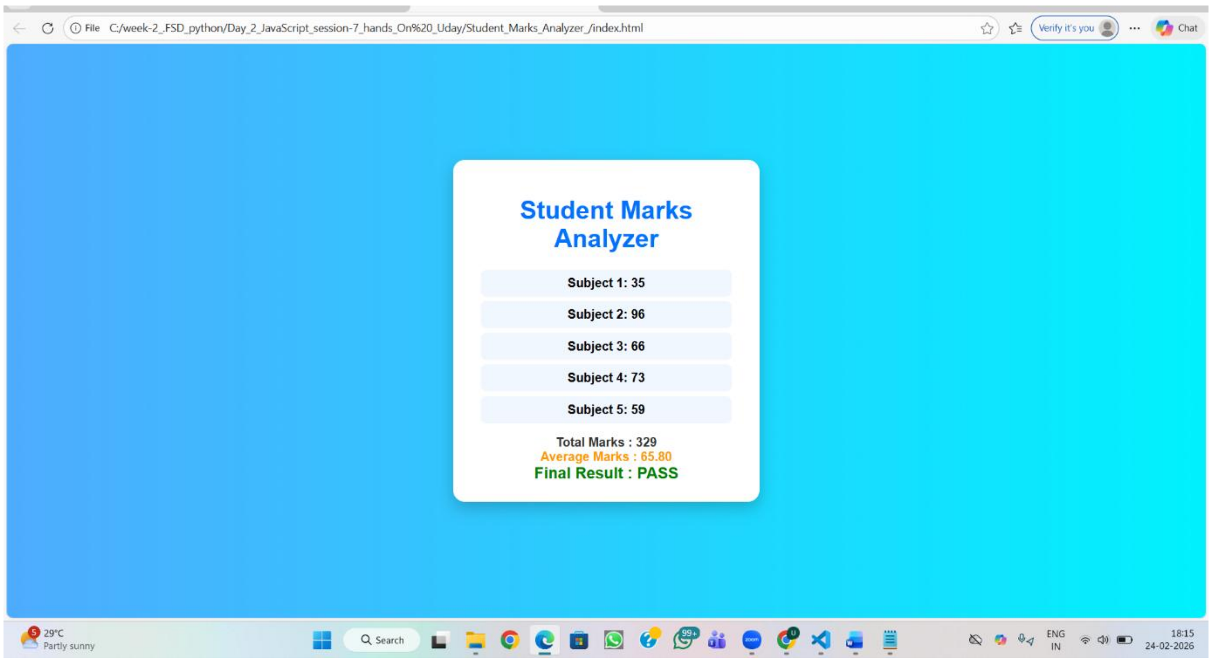
style.css :

```
body {
   font-family: Arial, sans-serif;
   background: linear-gradient(to right, #4facfe, #00f2fe);
   display: flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
   margin: 0;
}

/* Card layout */
.card {
   background: #ffffff;
   padding: 25px 35px;
   border-radius: 15px;
   box-shadow: 0 8px 20px rgba(0, 0, 0, 0.2);
   width: 320px;
   text-align: center;
}

h1 {
   color: #0072ff;
}

/* Subject box */
.subject {
   background: #f1f7ff;
   margin: 6px 0;
   padding: 8px;
   border-radius: 8px;
   font-weight: bold;
}

/* Result styles */
.total {
   margin-top: 15px;
   font-weight: bold;
   color: #333;
}

.average {
   color: #ff9800;
   font-weight: bold;
}

.pass {
   color: green;
   font-size: 20px;
   font-weight: bold;
}

.fail {
   color: red;
   font-size: 20px;
   font-weight: bold;
```

Day_2_JavaScript_session-7_hands_On _Uday

}**Technical Explanation :** I developed a Student Marks Analyzer that stores student marks in an array and processes them using ES6 array methods. I used the reduce() method inside an arrow function to calculate the total marks and derived the average by dividing the total by the number of subjects. Based on the average, I implemented pass/fail logic using a ternary operator. I used the map() method to generate subject-wise output dynamically and displayed the final result (total, average, and status) using template literals and DOM manipulation. The application follows ES6+ standards with const for fixed data, reusable modular functions, and no external libraries.

- **OUTPUT:**



Student Marks Analyzer

Subject 1: 35
Subject 2: 96
Subject 3: 66
Subject 4: 73
Subject 5: 59

Total Marks : 329
Average Marks : 65.80
Final Result : PASS

2) Problem Statement : Build a simple shopping cart summary system.

- **Index.html :**

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Cart Summary</title>
</head>

<body>

  <h2>Shopping Cart</h2>
  <p>Could You please open console to view the invoice</p>
  <script type="module" src="./main.js"></script>

</body>
</html>
```

Day_2_JavaScript_session-7_hands_On _Uday

- **main.js:**

```
import { getCartTotal } from './cart.js';

const cart = [
  { name: "Bag", price: 1500, quantity: 1 },
  { name: "Book", price: 60, quantity: 2 },
  { name: "Pen", price: 10, quantity: 1 }
];

// create bill lines
const bill = cart.map(item => {
  const itemTotal = item.price * item.quantity;

  return `
Item      : ${item.name}
Price     : ₹${item.price}
Quantity  : ${item.quantity}
Subtotal  : ₹${itemTotal}
________________________________`;
}).join("");

// get final total from module
const finalAmount = getCartTotal(cart);

// full invoice
const output = `
SHOPPING CART SUMMARY
_______________________________
${bill}
Total Amount : ₹${finalAmount}
_______________________________
`;

console.log(output);
```

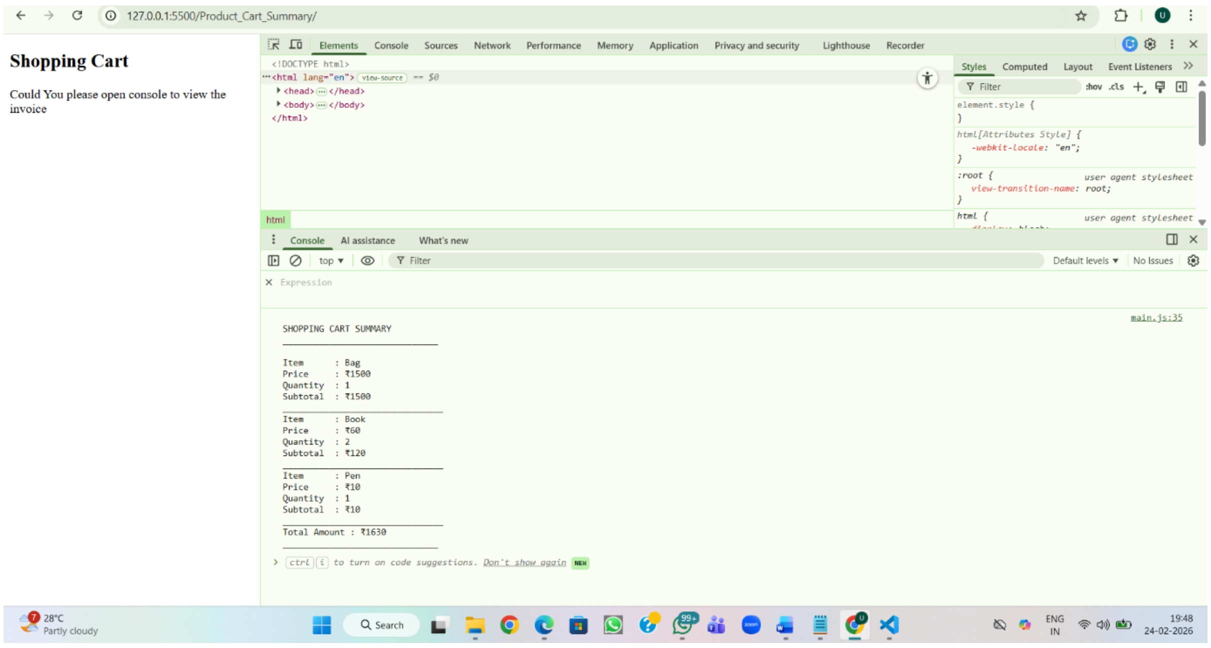
Day_2_JavaScript_session-7_hands_On _Uday

- **cart.js:**

```
export const getCartTotal = (items) =>
 items.reduce((sum, product) =>
  sum + product.price * product.quantity, 0);
```

Technical Explanation : I developed a shopping cart summary system using **ES6 modules**. Product items are stored as objects in an array with name, price, and quantity. I used the **map() method** to generate formatted bill lines for each product and **template literals** to create a clean invoice layout. The **total cart value** is calculated using a **reusable arrow function** exported from a separate module (cart.js) with **reduce()** to sum up the subtotals. Finally, the full invoice, including itemized details and the total amount, is displayed in the console. The project follows ES6+ standards with modular code, arrow functions, and no DOM manipulation.

OUTPUT:



```
SHOPPING CART SUMMARY
______________________________

Item      : Bag
Price     : ₹1500
Quantity  : 1
Subtotal  : ₹1500

_______________________________
Item      : Book
Price     : ₹60
Quantity  : 2
Subtotal  : ₹120

_______________________________
Item      : Pen
Price     : ₹10
Quantity  : 1
Subtotal  : ₹10

_______________________________
Total Amount : ₹1630

______________________________
```

Day_2_JavaScript_session-7_hands_On _Uday

3) Problem Statement Create an application that fetches weather data asynchronously.

HTML CODE :

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Weather App</title>
  <link rel="stylesheet" href="style.css">
</head>

<body>

  <div class="container">

    <h1>Weather Hunter</h1>

    <div class="search-box">
      <input type="text" id="city" placeholder="Enter city name">
      <button onclick="getWeather()">Search</button>
    </div>

    <div class="card" id="weatherCard">
      <p class="msg">Enter a city to get weather</p>
    </div>

  </div>

  <script src="weather.js"></script>
</body>

</html>
```

style.css:

```
body {
  margin: 0;
  font-family: Arial, sans-serif;
  background: linear-gradient(to right, #4facfe, #00f2fe);
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
}

/* main box */
.container {
  text-align: center;
  background: white;
  padding: 30px;
  border-radius: 15px;
  width: 320px;
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.2);
}

/* heading */
h1 {
  margin-bottom: 20px;
}

/* input + button */
.search-box {
  display: flex;
  gap: 10px;
  margin-bottom: 20px;
}

input {
  flex: 1;
  padding: 10px;
  border-radius: 8px;
  border: 1px solid gray;
}

button {
  padding: 10px 15px;
  border: none;
  background: #4facfe;
  color: white;
  border-radius: 8px;
  cursor: pointer;
}

button:hover {
  background: #0077ff;
}

/* weather card */
.card {
  background: #f2f2f2;
  padding: 15px;
  border-radius: 10px;
  text-align: left;
}

.msg {
  text-align: center;
  color: gray;
}
```

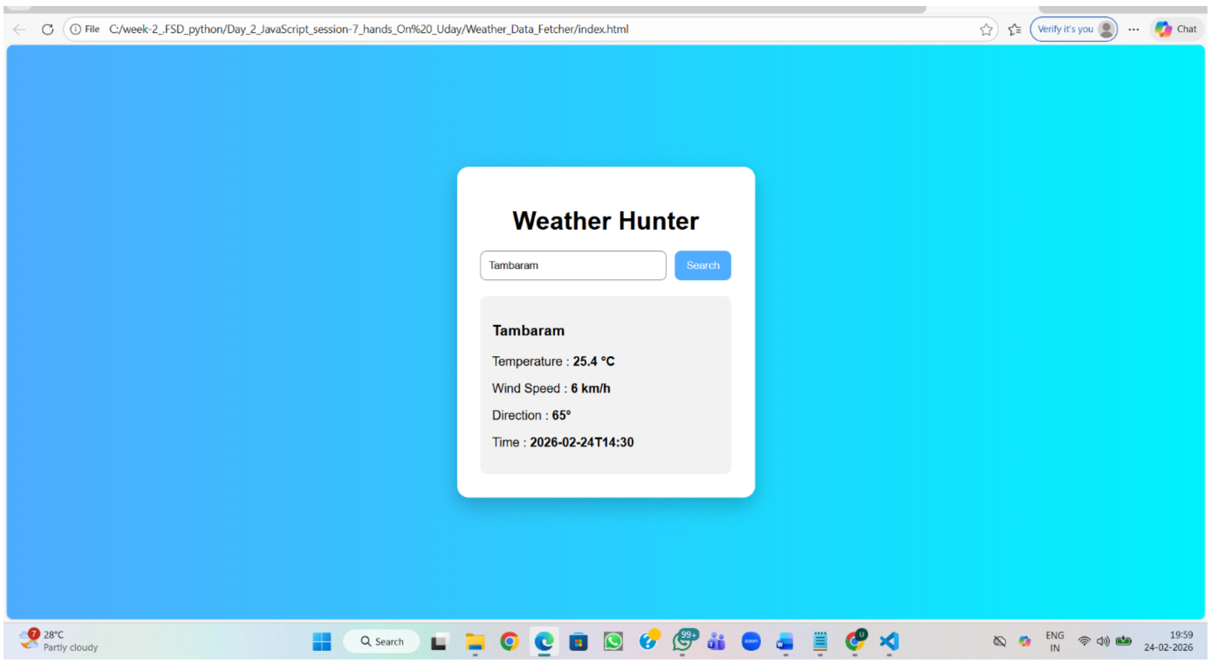
Weather.js :

```
const card = document.getElementById("weatherCard");

// display function
const showWeather = (city, data) => {

  card.innerHTML = `
```

```
    <h3> ${city}</h3>
    <p> Temperature : <b>${data.temperature} °C</b></p>
    <p> Wind Speed : <b>${data.windspeed} km/h</b></p>
    <p> Direction : <b>${data.winddirection}°</b></p>
    <p> Time : <b>${data.time}</b></p>
  `;
};


// get coordinates (Promise)
const getCoordinates = (city) => {

  const geoURL =
    `https://geocoding-api.open-meteo.com/v1/search?name=${city}&count=1`;

  return fetch(geoURL)
    .then(res => {
      if (!res.ok) throw new Error("City not found");
      return res.json();
    })
    .then(data => {

      if (!data.results) throw new Error("Invalid city");

      return {
        lat: data.results[0].latitude,
        lon: data.results[0].longitude,
        name: data.results[0].name
      };
    });
};



// fetch weather (async/await)
const getWeatherData = async (lat, lon) => {

  const url =
    `https://api.open-meteo.com/v1/forecast?latitude=${lat}&longitude=${lon}&current_weather=true`;

  const res = await fetch(url);

  if (!res.ok) throw new Error("Weather fetch failed");

  const data = await res.json();

  return data.current_weather;
};



// main function
const getWeather = async () => {

  const city = document.getElementById("city").value;

  if (!city) {
    card.innerHTML = `<p class="msg"> Enter a city name</p>`;
    return;
  }

  card.innerHTML = `<p class="msg">Loading...</p>`;

  try {

    const location = await getCoordinates(city);
    const weather = await getWeatherData(location.lat, location.lon);

    showWeather(location.name, weather);

  }
  catch (err) {
    card.innerHTML = `<p class="msg"> ${err.message}</p>`;
  }

};
```

- **Technical Explanation :** I developed a weather data fetcher that retrieves information from a public API asynchronously. I implemented two versions: one using **Promises** and another using **async/await** for cleaner asynchronous handling. Data fetching and processing are done using **arrow functions** and the output is formatted with **template literals**. Proper **error handling** is implemented using try/catch blocks to manage network or API errors gracefully. The application displays a structured weather report in the console or UI, following modern ES6+ standards and best practices for asynchronous JavaScript.

- **OUTPUT:**



File C:/week-2_FSD_python/Day_2_JavaScript_session-7_hands_On%20_Uday/Weather_Data_Fetcher/index.html

Weather Hunter

Tambaram | Search

Tambaram

Temperature : **25.4 °C**

Wind Speed : **6 km/h**

Direction : **65°**

Time : **2026-02-24T14:30**

Day_2_JavaScript_session-7_hands_On _Uday

4) Problem Statement : Develop a task manager where tasks are saved and retrieved asynchronously

HTML CODE :

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Async Task Manager</title>
  <link rel="stylesheet" href="style.css">
</head>

<body>

  <div class="box">
    <h2> Task Manager</h2>

    <input type="text" id="taskInput" placeholder="Enter a task">

    <div class="btns">
      <button onclick="addNewTask()">Add</button>
      <button onclick="deleteExistingTask()">Delete</button>
      <button onclick="showTasks()">List</button>
    </div>

    <ul id="taskList"></ul>
  </div>

  <script type="module" src="main.js"></script>
</body>

</html>
```

style.css :

```
body {
  font-family: Arial;
  background: linear-gradient(to right, #667eea, #764ba2);
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  color: white;
}

.box {
  background: white;
  color: black;
  padding: 20px;
  border-radius: 15px;
  width: 300px;
```

```
    text-align: center;
}

input {
    padding: 8px;
    width: 90%;
    margin-bottom: 10px;
}

button {
    padding: 8px 12px;
    margin: 5px;
    border: none;
    background: #667eea;
    color: white;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background: #764ba2;
}

li {
    text-align: left;
    margin-top: 5px;
}
```

main.js :

```
import { addTask, deleteTask, listTasks } from './storage.js';

const input = document.getElementById("taskInput");
const list = document.getElementById("taskList");

window.addNewTask = async () => {
    const task = input.value;

    if (task === "") return alert("Enter a task");

    const msg = await addTask(task);
    alert(msg);

    input.value = "";
};

window.deleteExistingTask = async () => {
    const task = input.value;

    const msg = await deleteTask(task);
    alert(msg);

    input.value = "";
};

window.showTasks = async () => {
    const tasks = await listTasks();
```

Day_2_JavaScript_session-7_hands_On _Uday

```
            list.innerHTML = tasks.map(t => `<li> ${t}</li>`).join("");
        };
```

storage.js :

```
let tasks = [];

/* ---------------- CALLBACK VERSION ---------------- */
export const addTaskCallback = (task, callback) => {
  setTimeout(() => {
    tasks.push(task);
    callback(` Task added: ${task}`);
  }, 500);
};

/* ---------------- PROMISE VERSION ---------------- */
export const addTaskPromise = (task) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      tasks.push(task);
      resolve(`  Task added: ${task}`);
    }, 500);
  });
};

/* ---------------- ASYNC/AWAIT FUNCTIONS ---------------- */
export const addTask = async (task) => {
  const msg = await addTaskPromise(task);
  return msg;
};

export const deleteTask = async (task) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      tasks = tasks.filter(t => t !== task);
      resolve(` Task deleted: ${task}`);
    }, 500);
  });
};

export const listTasks = async () => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(tasks);
    }, 500);
  });
};
```

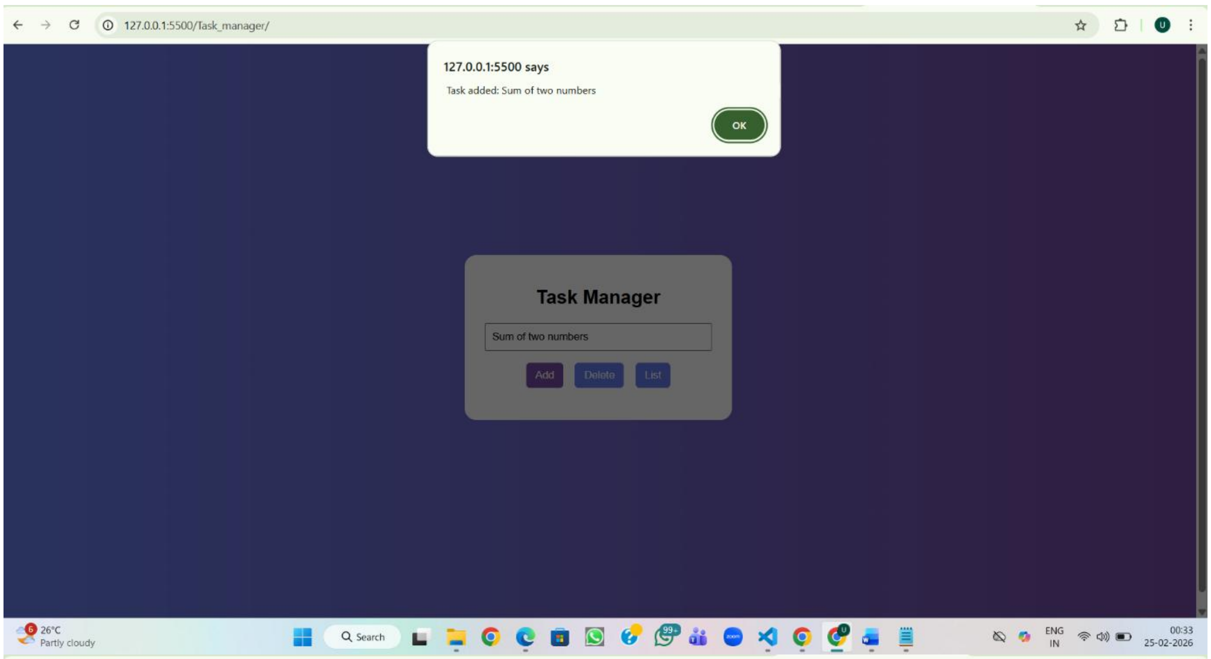
- ## Technical Explanation :

- I developed a task manager that stores tasks in an array and performs **add, delete, and list operations asynchronously**. Initially, I simulated async storage using **callbacks with**

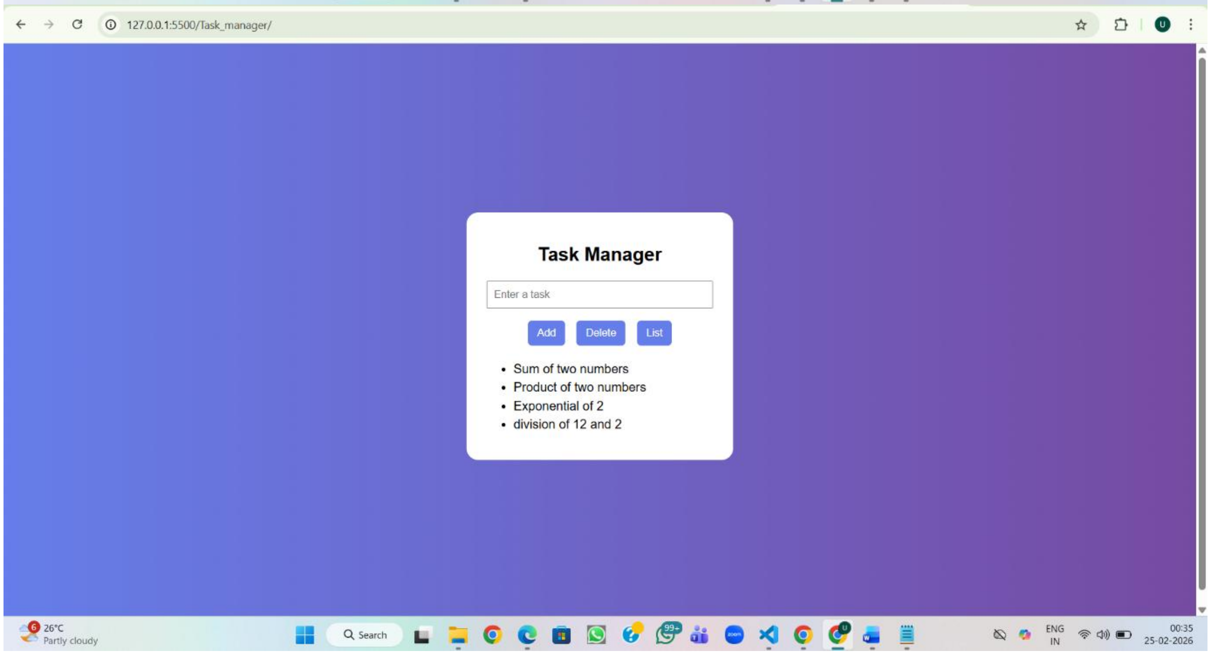
Day_2_JavaScript_session-7_hands_On _Uday

setTimeout, then converted the logic to **Promises**, and finally refactored it using **async/await** for cleaner, modern asynchronous handling. All functions are implemented as **arrow functions**, variables use let/const appropriately, and task output is displayed using **template literals**. The project demonstrates the evolution from **callback → promise → async/await** while following ES6+ modular coding standards.

- **OUTPUT:**


127.0.0.1:5500 says

Task added: Sum of two numbers


Task Manager

- Sum of two numbers
- Product of two numbers
- Exponential of 2
- division of 12 and 2