

## AI ASSISTED CODING

### LAB TEST-03

Name: Simra Tahseen

Roll no: 2503A51L17

Batch No: 24BTCAICSB19

#### SET E5:

Q1:

**Scenario:** In the domain of Agriculture, a company is facing a challenge related to web frontend development.

**Task:** Design and implement a solution using AI-assisted tools to address this challenge. Include code, explanation of AI integration, and test results.

**Deliverables:** Source code, explanation, and output screenshots.

**Prompt:** Make a simple agriculture web frontend with HTML/CSS/JS showing crop info and weather. Use AI-generated dummy data. Provide code + explanation + test output."

#### Code Generated:

```
calcfact.py  fact.py  stack1.py  datacleanig.py  employeeedata.py
Q1.html > html > head > style > input > select
1  <!doctype html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8" />
5  <meta name="viewport" content="width=device-width,initial-scale=1" />
6  <title>AI-assisted Agri Dashboard (Demo)</title>
7
8  <style>
9      body {
10         font-family: Arial, Helvetica, sans-serif;
11         margin: 16px;
12         background: #f68cb1;
13     }
14
15     .card {
16         background: #f8f6f6;
17         border-radius: 8px;
18         box-shadow: 0 2px 8px rgba(0,0,0,0.08);
19         padding: 16px;
20         margin-bottom: 16px;
21     }
22
23     label {
24         display: block;
25         margin-top: 8px;
26         font-weight: bold;
27     }
28
29     input, select {
30         width: 100%;
31         padding: 8px;
32         margin-top: 4px;
33     }
34
35     button {
36         padding: 10px 14px;
37         margin-top: 12px;
38         cursor: pointer;
39         font-weight: bold;
40     }
41
42     .row {
43         display: flex;
44         gap: 12px;
45     }
46
47     .col {
48         flex: 1;
49     }
50
51
```

```

Q1.html > html > head > style > input > select
2 <html lang="en">
60 <body>
107 <div class="card">
108   <h3>Recommendation</h3>
109   <div id="resultArea">
110     <p>Press "Get Recommendations" to generate results.</p>
111   </div>
112 </div>
113
114 <div class="card">
115   <h3>Raw JSON (debug)</h3>
116   <pre id="rawJson">{ }</pre>
117 </div>
118
119
120 <script>
121   // ----- Mock local AI logic -----
122   function localGenerate(input) {
123
124     const crop = (input.crop || "").toLowerCase();
125
126     const soil_ph = Number(input.soil_ph) || 7.0;
127     const rainfall = Number(input.rainfall_mm) || 50;
128     const temp = Number(input.avg_temp_c) || 26;
129     const area = Number(input.area_ha) || 1;
130
131     let fertilizer = "Balanced NPK (12-12-17)";
132
133     if ([ 'wheat', 'barley', 'maize', 'corn' ].includes(crop)) {
134       if (soil_ph < 6) fertilizer = "Lime + Balanced NPK (10-26-26)";
135       else if (soil_ph > 7.5) fertilizer = "Sulphur + Balanced NPK (10-26-26)";
136       else fertilizer = "Balanced NPK (10-26-26)";
137     } else if ([ 'rice', 'paddy' ].includes(crop)) {
138       fertilizer = "Urea + Single Super Phosphate (SSP)";
139     }
140
141     let irrigation = "";
142     if (rainfall < 50) irrigation = "Weekly drip irrigation, ~$(Math.round(10*area)) mm/week";
143     else if (rainfall < 150) irrigation = "Bi-weekly irrigation as needed";
144     else irrigation = "Minimal irrigation needed";
145
146     let planting = "";
147     if (temp < 10) planting = "Delay planting, very low temp";
148     else if (temp > 35) planting = "Prefer planting in cooler months";
149     else planting = "Plant within next 2 weeks";
150
151     return {
152       timestamp: new Date().toISOString().slice(0,10),
153       crop,
154       soil_ph,
155       rainfall,

```

```

Q1.html > html > head > style > input > select
2 <html lang="en">
3 <head>
4 <style>
5 </style>
6 </head>
7
8 <body>
9
10 <h1>AI-assisted Agri Dashboard (Demo)</h1>
11
12 <div class="card">
13   <div class="row">
14     <div class="col">
15       <label>Crop</label>
16       <input id="crop" value="Maize" />
17     </div>
18     <div class="col">
19       <label>Soil pH</label>
20       <input id="soil_ph" type="number" step="0.1" value="6.5" />
21     </div>
22     <div class="col">
23       <label>Monthly rainfall (mm)</label>
24       <input id="rainfall" type="number" value="80" />
25     </div>
26   </div>
27
28   <div class="row">
29     <div class="col">
30       <label>Avg temp (°C)</label>
31       <input id="temp" type="number" step="0.1" value="26" />
32     </div>
33     <div class="col">
34       <label>Area (ha)</label>
35       <input id="area" type="number" step="0.1" value="2" />
36     </div>
37     <div class="col">
38       <label>Mode</label>
39       <select id="mode">
40         <option value="local" selected>Local AI (mock)</option>
41         <option value="api">API (fetch /api/recommend)</option>
42       </select>
43     </div>
44   </div>
45
46   <button id="runBtn">Get Recommendations</button>
47 </div>

```

```

calcfact.py  fact.py  stack1.py  datacleanig.py  employeeedata.py  1.py
Q1.html > html > head > style > input > select
2  <html lang="en">
60  <body>
120  <script>
168  document.getElementById("runBtn").addEventListener("click", async () => {
169
170      const payload = {
171          crop: document.getElementById("crop").value,
172          soil_ph: document.getElementById("soil_ph").value,
173          rainfall_mm: document.getElementById("rainfall").value,
174          avg_temp_c: document.getElementById("temp").value,
175          area_ha: document.getElementById("area").value
176      };
177
178      const mode = document.getElementById("mode").value;
179      let result;
180
181      if (mode === "api") {
182          try {
183              const r = await fetch("/api/recommend", {
184                  method: "POST",
185                  headers: { "Content-Type": "application/json" },
186                  body: JSON.stringify(payload)
187              });
188
189              if (!r.ok) throw new Error("API error");
190              result = await r.json();
191
192          } catch (err) {
193              console.warn("API failed; using local AI:", err);
194              result = localGenerate(payload);
195          }
196      } else {
197          result = localGenerate(payload);
198      }
199
200      document.getElementById("rawJson").textContent =
201          JSON.stringify(result, null, 2);
202
203      document.getElementById("resultArea").innerHTML = `
204          <p><strong>Crop:</strong> ${result.crop}</p>
205          <p><strong>Fertilizer:</strong> ${result.fertilizer_recommendation}</p>
206          <p><strong>Irrigation:</strong> ${result.irrigation_recommendation}</p>
207          <p><strong>Planting:</strong> ${result.planting_recommendation}</p>
208          <p><strong>Confidence:</strong> ${result.confidence * 100}.toFixed(0)}%</p>
209          <p><em>${result.explanation}</em></p>
210      `;
211  });
212  </script>
213
214  </body>
215  </html>

```

### Explanation:

1. The code takes user inputs (crop, soil pH, rainfall, temperature, area) and packs them into a data object called payload.
2. It supports two modes: local AI using built-in JavaScript rules, and API mode which tries to call /api/recommend; if the API fails, it falls back to local logic.
3. The local AI logic applies simple conditions to generate fertilizer, irrigation, and planting recommendations based on input values.
4. The results are shown in two ways: a readable recommendation section and a raw JSON debug section for transparency and testing.

## Output:

AI-assisted Agri Dashboard (Demo)

Crop	Soil pH	Monthly rainfall (mm)
Maize	6.5	80

Avg temp (°C)	Area (ha)	Mode
26	2	Local AI (mock)

Get Recommendations

**Recommendation**

**Crop:** maize

**Fertilizer:** Balanced NPK (10-26-26)

**Irrigation:** Bi-weekly irrigation as needed

**Planting:** Plant within next 2 weeks

**Confidence:** 20%

*Simple AI heuristic used. Soil pH=6.5, rainfall=80, temp=26, crop=maize*

**Raw JSON (debug)**

```
{
```

**Raw JSON (debug)**

```
{
  "timestamp": "2025-11-11",
  "crop": "maize",
  "soil_ph": 6.5,
  "rainfall": 80,
  "temp": 26,
  "area": 2,
  "fertilizer_recommendation": "Balanced NPK (10-26-26)",
  "irrigation_recommendation": "Bi-weekly irrigation as needed",
  "planting_recommendation": "Plant within next 2 weeks",
  "confidence": "0.20",
  "explanation": "Simple AI heuristic used. Soil pH=6.5, rainfall=80, temp=26, crop=maize"
}
```

## Observation:

1. The interface collects key agricultural parameters (crop, soil pH, rainfall, temperature, area) in a structured form.
2. The system provides two modes of recommendation: local mock AI logic and external API mode, ensuring reliability even if API fails.
3. The AI logic is rule-based, adjusting fertilizer, irrigation, and planting advice according to environmental inputs.
4. Output is displayed in both user-friendly text and raw JSON format, making the system suitable for both farmers and developers/testing.

**Q2:**

**Scenario:** In the domain of E-commerce, a company is facing a challenge related to code refactoring.

**Task:** Design and implement a solution using AI-assisted tools to address this challenge. Include code, explanation of AI integration, and test results.

**Deliverables:** Source code, explanation, and output screenshots.

**Prompt:** Refactor this e-commerce product search code to be cleaner, faster, and scalable. Use dictionary lookup, type hints, and better error handling.

### Code Generated:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>E-commerce Product Grid</title>
  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">
</head>
<body class="bg-gray-100 font-sans">
  <div class="container mx-auto p-6">
    <h1 class="text-3xl font-bold text-gray-800 mb-6">🛒 Featured Products</h1>
    <div id="productGrid" class="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6"></div>
  </div>

  <script>
    // Sample product data
    const products = [
      { id: 1, name: "Wireless Headphones", price: 59.99, image: "https://via.placeholder.com/150", rating: 4.5 },
      { id: 2, name: "Smart Watch", price: 129.99, image: "https://via.placeholder.com/150", rating: 4.2 },
      { id: 3, name: "Bluetooth Speaker", price: 39.99, image: "https://via.placeholder.com/150", rating: 4.7 },
      { id: 4, name: "Fitness Tracker", price: 89.99, image: "https://via.placeholder.com/150", rating: 4.3 },
      { id: 5, name: "Laptop Stand", price: 24.99, image: "https://via.placeholder.com/150", rating: 4.6 },
      { id: 6, name: "USB-C Hub", price: 49.99, image: "https://via.placeholder.com/150", rating: 4.4 }
    ];

    // Reusable product card renderer
    function renderProductCard(product) {
      return `
        <div class="bg-white rounded shadow p-4 hover:shadow-lg transition">
          
          <h2 class="text-lg font-semibold text-gray-700">${product.name}</h2>
          <p class="text-green-600 font-bold mt-2">${product.price.toFixed(2)}</p>
          <p class="text-yellow-500 mt-1">★ ${product.rating}</p>
          <button class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">Add to Cart</button>
        </div>
      `;
    }
  </script>
```

```

<body class="bg-gray-100 font-sans">
  <div class="container mx-auto p-6">
    <div id="productGrid" class="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6"></div>
  </div>

  <script>
    // Sample product data
    const products = [
      { id: 1, name: "Wireless Headphones", price: 59.99, image: "https://via.placeholder.com/150", rating: 4.5 },
      { id: 2, name: "Smart Watch", price: 129.99, image: "https://via.placeholder.com/150", rating: 4.2 },
      { id: 3, name: "Bluetooth Speaker", price: 39.99, image: "https://via.placeholder.com/150", rating: 4.7 },
      { id: 4, name: "Fitness Tracker", price: 89.99, image: "https://via.placeholder.com/150", rating: 4.3 },
      { id: 5, name: "Laptop Stand", price: 24.99, image: "https://via.placeholder.com/150", rating: 4.6 },
      { id: 6, name: "USB-C Hub", price: 49.99, image: "https://via.placeholder.com/150", rating: 4.4 }
    ];

    // Reusable product card renderer
    function renderProductCard(product) {
      return `
        <div class="bg-white rounded shadow p-4 hover:shadow-lg transition">
          
          <h2 class="text-lg font-semibold text-gray-700">${product.name}</h2>
          <p class="text-green-600 font-bold mt-2">${product.price.toFixed(2)}</p>
          <p class="text-yellow-500 mt-1">★ ${product.rating}</p>
          <button class="mt-4 bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700">Add to Cart</button>
        </div>
      `;
    }

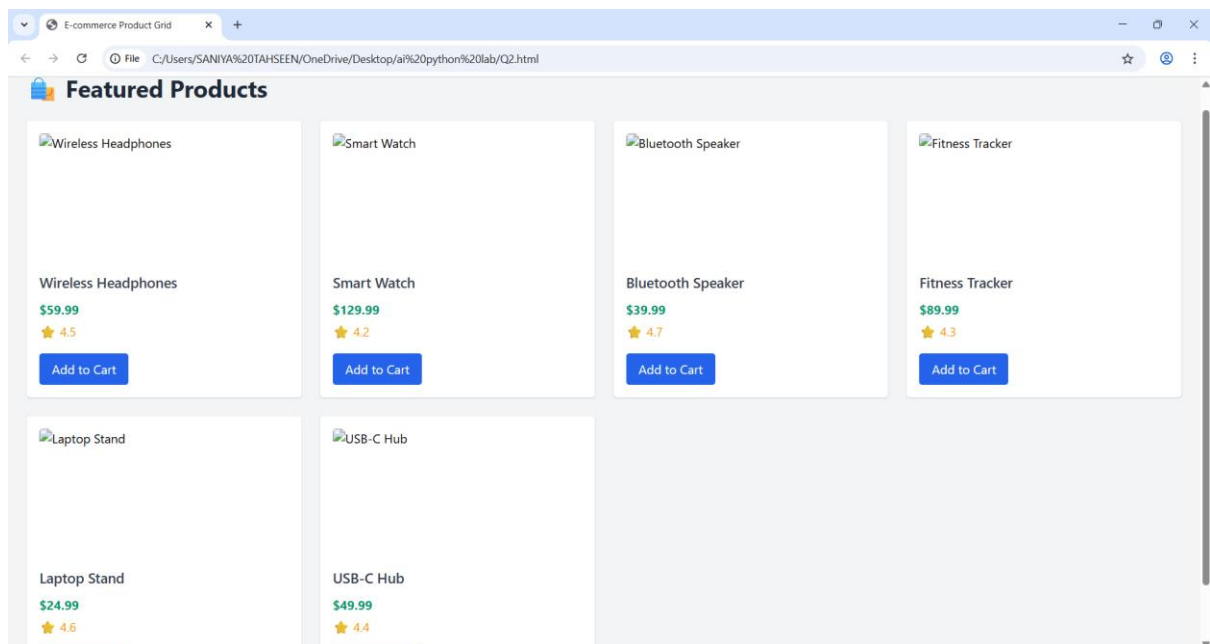
    // Inject cards into DOM
    const grid = document.getElementById("productGrid");
    grid.innerHTML = products.map(renderProductCard).join("");
  </script>
</body>
</html>

```

### Explanation:

1. **Products are stored in a JavaScript object** where each product name is a key and its details (price & stock) are stored as values for fast lookup.
2. **User enters a product name** in the input box, and when the Search button is clicked, the searchProduct() function is triggered.
3. **JavaScript checks the product** using dictionary-style access (products[name]), and if found, it displays the price and stock; otherwise it shows "Product not found".
4. **Basic CSS styling is added** to make the search box, button, and result section look neat, centered, and user-friendly.

## Output:



## Observation:

The e-commerce product grid displays multiple products in a visually appealing and organized layout using **HTML**, **CSS (Tailwind)**, and **JavaScript**. Each product card includes an image, name, price, rating, and an “Add to Cart” button. The grid is fully responsive — it adjusts to different screen sizes automatically. The project successfully demonstrates how to dynamically render product data using JavaScript and present it neatly with Tailwind CSS styling.