

# LAB TEST-3

## Set E12

Q1:

Scenario: In the Education sector, a company faces a challenge related to web frontend development.

Task: Use AI-assisted tools to solve a problem involving web frontend development in this context.

Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

SOURCE CODE:

INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ed-Tech Quiz</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="app">
        <h1>Simple Education Quiz</h1>
        <div class="quiz-container">
            <h2 id="question-text">Question goes here</h2>
```

```
<div id="answer-buttons" class="btn-grid">
</div>

<button id="next-btn">Next</button>

</div>

<div id="result-container" class="hide">
<h2>Quiz Complete!</h2>
<p id="score-text"></p>
<button id="restart-btn">Restart Quiz</button>
</div>
</div>

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

</body>
</html>
```

## STYLE.CSS

```
/* General Styling */

body {
    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue",
    Arial, sans-serif;
    background-color: #f0f4f8;
    color: #333;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: 20px;
}
```

```
}
```

```
.app {  
  background-color: #ffffff;  
  width: 100%;  
  max-width: 600px;  
  border-radius: 12px;  
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);  
  padding: 30px;  
  box-sizing: border-box; /* Ensures padding doesn't break layout */  
}
```

```
h1 {
```

```
  font-size: 1.5rem;  
  color: #005a9e;  
  text-align: center;  
  margin-bottom: 20px;  
}
```

```
h2 {
```

```
  font-size: 1.25rem;  
  margin-bottom: 15px;  
}
```

```
/* Quiz Container */
```

```
.btn-grid {  
  display: grid;  
  grid-template-columns: 1fr; /* Single column layout */
```

```
gap: 10px;  
margin-top: 20px;  
}  
  
.btn {  
width: 100%;  
padding: 15px;  
font-size: 1rem;  
text-align: left;  
background-color: #f0f4f8;  
border: 2px solid #d1d9e0;  
border-radius: 8px;  
cursor: pointer;  
transition: background-color 0.2s ease, border-color 0.2s ease;  
}  
  
.btn:hover:not(:disabled) {  
background-color: #d1d9e0;  
}  
  
.btn:disabled {  
cursor: not-allowed;  
opacity: 0.7;  
}  
  
/* Feedback Classes */  
.btn.correct {  
background-color: #d4edda;
```

```
border-color: #c3e6cb;  
color: #155724;  
}  
  
.btn.incorrect {  
  
background-color: #f8d7da;  
border-color: #f5c6cb;  
color: #721c24;  
}  
  
/* Controls */  
  
#next-btn, #restart-btn {  
  
display: block;  
width: 100%;  
padding: 15px;  

```

```
}
```

```
/* Utility Class */
```

```
.hide {  
    display: none;  
}
```

## SCRIPT.JS

```
// --- 1. Data: Define the questions ---
```

```
const questions = [  
    {  
        question: "What does HTML stand for?",  
        answers: [  
            { text: "Hyper Trainer Marking Language", correct: false },  
            { text: "Hyper Text Marketing Language", correct: false },  
            { text: "Hyper Text Markup Language", correct: true },  
            { text: "Hyperlink and Text Markup Language", correct: false }  
        ]  
    },  
    {  
        question: "Which CSS property is used to change the text color of an element?",  
        answers: [  
            { text: "font-color", correct: false },  
            { text: "text-color", correct: false },  
            { text: "background-color", correct: false },  
            { text: "color", correct: true }  
        ]  
    }]
```

```
        ],
    },
{
    question: "What is the correct way to include an external JavaScript file?",  

    answers: [
        { text: "<script href='script.js'></script>", correct: false },
        { text: "<script src='script.js'></script>", correct: true },
        { text: "<javascript src='script.js'></javascript>", correct: false },
        { text: "<link rel='javascript' href='script.js'>", correct: false }
    ]
};
```

```
// --- 2. Get references to HTML elements ---  
  
const questionElement = document.getElementById('question-text');  
  
const answerButtonsElement = document.getElementById('answer-buttons');  
  
const nextButton = document.getElementById('next-btn');  
  
const quizContainer = document.querySelector('.quiz-container');  
  
const resultContainer = document.getElementById('result-container');  
  
const scoreText = document.getElementById('score-text');  
  
const restartButton = document.getElementById('restart-btn');
```

```
// --- 3. Quiz State Variables ---  
  
let currentQuestionIndex = 0;  
  
let score = 0;
```

```
// --- 4. Core Functions ---
```

```
/**  
 * Starts or restarts the quiz  
 */  
  
function startQuiz() {  
  
    currentQuestionIndex = 0;  
  
    score = 0;  
  
    nextButton.innerHTML = "Next";  
  
    nextButton.classList.add('hide'); // Hide next button initially  
  
    resultContainer.classList.add('hide');  
  
    quizContainer.classList.remove('hide');  
  
    showQuestion();  
  
}  
  
}
```

```
/**  
 * Displays the current question and answers  
 */  
  
function showQuestion() {  
  
    // 1. Reset the state from the previous question  
  
    resetState();  
  
    // 2. Get the current question object  
  
    let currentQuestion = questions[currentQuestionIndex];  
  
    let questionNo = currentQuestionIndex + 1;  
  
    questionElement.innerHTML = questionNo + ". " + currentQuestion.question;  
  
    // 3. Create and display buttons for each answer  
  
    currentQuestion.answers.forEach(answer => {  
  
        const button = document.createElement('button');
```

```
button.innerHTML = answer.text;
button.classList.add('btn');

// Store the correct answer info on the button itself
if (answer.correct) {
    button.dataset.correct = answer.correct;
}

button.addEventListener('click', selectAnswer);
answerButtonsElement.appendChild(button);
});

}

/***
 * Resets the answer buttons and hides the next button
 */
function resetState() {
    nextButton.classList.add('hide');
    while (answerButtonsElement.firstChild) {
        answerButtonsElement.removeChild(answerButtonsElement.firstChild);
    }
}

/***
 * Handles the click event when a user selects an answer
 */
function selectAnswer(e) {
    const selectedButton = e.target;
```

```
const isCorrect = selectedButton.dataset.correct === "true";  
  
// 1. Update score  
if (isCorrect) {  
    score++;  
}  
  
// 2. Provide visual feedback  
Array.from(answerButtonsElement.children).forEach(button => {  
    setStatusClass(button, button.dataset.correct === "true");  
}  
// 3. Disable all buttons  
button.disabled = true;  
});  
  
// 4. Show the "Next" button  
nextButton.classList.remove('hide');  
}  
  
/**  
 * Applies .correct or .incorrect classes for feedback  
 */  
function setStatusClass(element, isCorrect) {  
    if (isCorrect) {  
        element.classList.add('correct');  
    } else {  
        element.classList.add('incorrect');  
    }  
}
```

```
/**  
 * Handles the click event for the "Next" button  
 */  
  
function handleNextButton() {  
    currentQuestionIndex++;  
  
    if (currentQuestionIndex < questions.length) {  
        showQuestion();  
    } else {  
        showResults();  
    }  
}  
  
/**  
 * Displays the final score  
 */  
  
function showResults() {  
    quizContainer.classList.add('hide');  
  
    resultContainer.classList.remove('hide');  
  
    scoreText.innerHTML = `You scored ${score} out of ${questions.length}!`;  
}  
  
// --- 5. Event Listeners ---  
  
// Handle "Next" button click  
nextButton.addEventListener('click', handleNextButton);  
  
// Handle "Restart" button click
```

```
restartButton.addEventListener('click', startQuiz);

// Start the quiz when the page loads
document.addEventListener('DOMContentLoaded', startQuiz);
```

## CODE EXPLANATION:

### Code Explanation

- **HTML:** The structure is semantic and minimal. We have two main "views": .quiz-container (for active questions) and #result-container (for the final score). We toggle between them using the .hide CSS class.
- **CSS:** The styling is straightforward. The most important parts are the .correct and .incorrect classes, which provide immediate visual feedback. The .hide class (display: none;) is our simple way of switching views without needing a complex framework.
- **JavaScript:**
  - **questions Array:** This array of objects is our "database." It makes the quiz easy to expand by just adding more objects.
  - **startQuiz:** This is the entry point. It resets the state variables (currentQuestionIndex, score) and shows the first question.
  - **showQuestion:** This is the core of the dynamic UI. It clears old answer buttons (resetState) and then loops through the answers for the *current* question, creating a new <button> element for each one.
  - **button.dataset.correct:** This is a key technique. We store the "correctness" data directly on the HTML button element. When the user clicks, we can easily check this data attribute to see if they were right.
  - **selectAnswer:** This function manages the logic after a click. It checks the dataset, updates the score, applies the .correct/.incorrect classes to *all* buttons for feedback, and disables them to prevent changing the answer. Finally, it reveals the "Next" button.
  - **handleNextButton:** This function controls the flow. It increments the currentQuestionIndex and either calls showQuestion (if more questions exist) or showResults (if the quiz is over).

## SAMPLE OUTPUT:

The screenshot shows a CodePen environment with an untitled pen. The code includes HTML, CSS, and JavaScript. The quiz interface displays a question: "1. What does HTML stand for?" with three options: "Hyper Trainer Marketing Language", "Hyper Text Marketing Language", and "Hyper Text Markup Language".

```
HTML
16+     <button id="next-btn">Next</button>
17+   </div>
18+   <div id="result-container" class="hide">
19+     <h2>Quiz Complete!</h2>
20+     <p id="score-text"></p>
21+     <button id="restart-btn">Restart Quiz</button>
22+   </div>
23+   <script src="script.js"></script>
24+ </body>
25+ </html>
```

```
CSS
89 margin-top: 20px;
90 transition: background-color 0.2s ease;
91 }
92
93 #next-btn:hover, #restart-btn:hover {
94 background-color: #0056b3;
95 }
96
97 /* Utility Class */
98 .hide {
99 display: none;
100 }
```

```
JS
154 // Handle "Next" button click
155 nextButton.addEventListener('click', handleNextButton);
156
157 // Handle "Restart" button click
158 restartButton.addEventListener('click', startQuiz);
159
160 // Start the quiz when the page loads
161 document.addEventListener('DOMContentLoaded', startQuiz);
```

The screenshot shows a CodePen environment with an untitled pen. The code includes HTML, CSS, and JavaScript. The quiz interface displays a question: "2. Which CSS property is used to change the text color of an element?" with two options: "font-color" and "text-color".

```
HTML
16+     <button id="next-btn">Next</button>
17+   </div>
18+   <div id="result-container" class="hide">
19+     <h2>Quiz Complete!</h2>
20+     <p id="score-text"></p>
21+     <button id="restart-btn">Restart Quiz</button>
22+   </div>
23+   <script src="script.js"></script>
24+ </body>
25+ </html>
```

```
CSS
89 margin-top: 20px;
90 transition: background-color 0.2s ease;
91 }
92
93 #next-btn:hover, #restart-btn:hover {
94 background-color: #0056b3;
95 }
96
97 /* Utility Class */
98 .hide {
99 display: none;
100 }
```

```
JS
154 // Handle "Next" button click
155 nextButton.addEventListener('click', handleNextButton);
156
157 // Handle "Restart" button click
158 restartButton.addEventListener('click', startQuiz);
159
160 // Start the quiz when the page loads
161 document.addEventListener('DOMContentLoaded', startQuiz);
```

Q2:

Scenario: In the Retail sector, a company faces a challenge related to algorithms with ai assistance.

Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this context.

Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

Source code:

HTML CODE:

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI-Assisted Retail Recommendations</title>
    <!-- Load Tailwind CSS -->
    <script src="https://cdn.tailwindcss.com"></script>

    <!-- Link to your new CSS file -->
    <link rel="stylesheet" href="style.css">

    <!-- Load Inter font -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap" rel="stylesheet">
  </head>
  <body class="bg-gray-100">
```

```
<div class="container mx-auto max-w-4xl p-4 sm:p-8">

  <h1 class="text-3xl font-bold text-center text-gray-800 mb-8">AI "Cold Start"
  Recommendation</h1>

  <!-- 1. The "New Product" Display -->

  <div class="bg-white rounded-lg shadow-lg overflow-hidden md:flex">

    <div class="md:w-1/3">
      <!-- Using a placeholder for the new product image -->
      
    </div>

    <div class="md:w-2/3 p-6 flex flex-col justify-between">
      <div>
        <div class="text-sm text-gray-500 uppercase" id="product-category">Men's
        Footwear</div>

        <h2 class="text-2xl font-bold text-gray-900 mt-2" id="product-name">Trail-Ready
        Hiking Boots</h2>

        <p class="text-gray-700 mt-4" id="product-desc">
          A brand new item. Durable, waterproof hiking boots designed for all-terrain
          comfort.
          Features a breathable mesh lining, high-traction rubber outsole, and reinforced
          toe cap.
        </p>
      </div>
      <!-- The button that triggers the AI algorithm -->
      <button id="recommend-btn" class="mt-6 w-full bg-sky-600 text-white font-
      semibold py-3 px-6 rounded-lg shadow-md hover:bg-sky-700 transition duration-300 ease-
      in-out flex items-center justify-center">
```

```
<svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"  
stroke-width="1.5" stroke="currentColor" class="w-5 h-5 mr-2">  
  
  <path stroke-linecap="round" stroke-linejoin="round" d="M9.813 15.904L9  
18.75L-.813-2.846a4.5 4.5 0 00-3.09-3.09L1.25 12L2.846-.813a4.5 4.5 0 003.09-3.09L9  
5.25L8.13 2.846a4.5 4.5 0 003.09 3.09L15.75 12L2.846.813a4.5 4.5 0 00-3.09 3.09zM18.25  
12L17 15.75L-1.25-3.75-3.75-1.25L17 9.5L1.25 3.75" />  
  
</svg>
```

```
    Get AI Recommendations  
  
  </button>  
  
</div>  
  
</div>
```

```
<!-- 2. AI Recommendations Section -->  
  
<div class="mt-12">  
  <h3 class="text-2xl font-semibold text-gray-800 mb-6">You Might Also Like...</h3>  
  
  <!-- Loading Spinner (initially hidden) -->  
  <div id="loading" class="flex justify-center items-center h-32" style="display: none;">  
    <div class="spinner"></div>  
    <span class="ml-4 text-gray-600">Finding the perfect items...</span>  
  </div>  
  
  <!-- Results Container (where AI recommendations are injected) -->  
  <div id="results-container" class="grid grid-cols-1 md:grid-cols-3 gap-6">  
    <!-- AI-generated recommendations will be injected here by the script -->  
  </div>  
  
</div>  
  
</div>
```

```
<!-- Link to your new JavaScript file -->  
<script src="script.js"></script>  
</body>  
</html>
```

CSS CODE:

```
/* Use the Inter font */  
body {  
    font-family: 'Inter', sans-serif;  
}  
  
/* Simple loading spinner */  
.spinner {  
    border: 4px solid rgba(0, 0, 0, 0.1);  
    width: 36px;  
    height: 36px;  
    border-radius: 50%;  
    border-left-color: #0ea5e9;  
    animation: spin 1s ease infinite;  
}  
  
@keyframes spin {  
    0% { transform: rotate(0deg); }  
    100% { transform: rotate(360deg); }  
}
```

JS CODE:

```
// --- 1. Get Element References ---  
  
const recommendBtn = document.getElementById('recommend-btn');  
  
const loadingSpinner = document.getElementById('loading');  
  
const resultsContainer = document.getElementById('results-container');  
  
  
// --- 2. Add Event Listener ---  
  
recommendBtn.addEventListener('click', getAiRecommendations);  
  
  
// --- 3. The "Algorithm": AI Schema for Structured JSON Output ---  
  
// This schema is the core of our "algorithm." It forces the AI  
// to return data in a predictable, structured format that we  
// can immediately parse and render in our UI.  
  
const RECOMMENDATION_SCHEMA = {  
    type: "OBJECT",  
    properties: {  
        "recommendations": {  
            type: "ARRAY",  
            items: {  
                type: "OBJECT",  
                properties: {  
                    "productName": { "type": "STRING" },  
                    "category": { "type": "STRING" },  
                    "reason": { "type": "STRING" }  
                },  
                required: ["productName", "category", "reason"]  
            }  
        }  
    }  
}
```

```
        }

    },
    required: ["recommendations"]
};

/** 
 * Fetches recommendations from the Gemini API with exponential backoff for resilience.
 */

async function fetchWithRetry(apiUrl, payload, retries = 3, delay = 1000) {
    for (let i = 0; i < retries; i++) {
        try {
            const response = await fetch(apiUrl, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });

            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            return await response.json();
        } catch (error) {
            // Log errors but don't show to user
            console.error(`Attempt ${i + 1} failed: ${error.message}`);
            if (i < retries - 1) {
                // Wait longer before retrying
                await new Promise(resolve => setTimeout(resolve, delay * Math.pow(2, i)));
            }
        }
    }
}
```

```
        } else {
            throw error; // Rethrow last error after all retries fail
        }
    }
}

/**
 * Main function to get and display AI recommendations.
 */
async function getAiRecommendations() {
    // 1. Set UI to loading state
    loadingSpinner.style.display = 'flex';
    resultsContainer.innerHTML = ""; // Clear old results
    recommendBtn.disabled = true;
    recommendBtn.classList.add('opacity-50', 'cursor-not-allowed');

    // 2. Get product data from the page
    const productName = document.getElementById('product-name').innerText;
    const productDesc = document.getElementById('product-desc').innerText;
    const productCat = document.getElementById('product-category').innerText;

    // 3. Define the AI prompts (This is the core of the "algorithm")
    // The "System Prompt" defines the AI's persona, rules, and goals.
    const systemPrompt = `You are an expert e-commerce merchandiser and retail analyst.
Your goal is to increase the average order value (AOV) by providing 3 relevant product
recommendations.`
```

For each recommendation, provide a compelling, one-sentence reason why it complements the user's current item.

Focus on cross-sells (e.g., socks, boot cleaner) and compatible items (e.g., backpack, waterproof pants).

You MUST return your answer in the provided JSON schema.`;

```
// The "User Query" provides the specific data for this single request
```

```
const userQuery = `
```

```
  Main Product: ${productName}
```

```
  Category: ${productCat}
```

```
  Description: ${productDesc}
```

Generate 3 recommendations for this "cold start" product.

```
`;
```

```
// 4. Set up the Gemini API call
```

```
const apiKey = ""; // API key is provided by the environment
```

```
const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-09-2025:generateContent?key=${apiKey}`;
```

```
const payload = {
```

```
  contents: [{ parts: [{ text: userQuery }] }],
```

```
  systemInstruction: {
```

```
    parts: [{ text: systemPrompt }]
```

```
  },
```

```
  // This forces the AI to reply in the structured JSON format we defined
```

```
  generationConfig: {
```

```
    responseMimeType: "application/json",
```

```
    responseSchema: RECOMMENDATION_SCHEMA,
```

```
    temperature: 0.7
  }
};

// 5. Make the API call and handle response

try {
  const result = await fetchWithRetry(apiUrl, payload);

  if (!result.candidates || !result.candidates[0].content.parts[0].text) {
    throw new Error("Invalid response structure from AI.");
  }

  // Extract and parse the JSON data
  const jsonText = result.candidates[0].content.parts[0].text;
  const data = JSON.parse(jsonText);

  // 6. Render the recommendations
  renderRecommendations(data.recommendations);

} catch (error) {
  console.error("Error fetching AI recommendations:", error);
  resultsContainer.innerHTML = `<p class="text-red-600 text-center col-span-full">
    Sorry, we couldn't load recommendations at this time. Please try again later.
  </p>`;
}

} finally {
  // 7. Reset UI from loading state
  loadingSpinner.style.display = 'none';
  recommendBtn.disabled = false;
```

```
recommendBtn.classList.remove('opacity-50', 'cursor-not-allowed');

}

}

/***
 * Renders the recommendation data into HTML cards.
 */
function renderRecommendations(recommendations) {

if (!recommendations || recommendations.length === 0) {

resultsContainer.innerHTML = `<p class="text-gray-600 text-center col-span-full">
    No specific recommendations found.
</p>`;

return;
}

// Create a card for each recommendation
recommendations.forEach(item => {

const card = document.createElement('div');
card.className = 'bg-white rounded-lg shadow-md overflow-hidden animate-fade-in';

// Use a placeholder image, dynamically adding the product name
card.innerHTML = `



<div class="p-5">
    <div class="text-xs text-sky-600 uppercase font-semibold">${item.category}</div>
```

```
<h4 class="text-lg font-bold text-gray-900 mt-1">${item.productName}</h4>
<p class="text-gray-600 text-sm mt-2">${item.reason}</p>
</div>
`;
resultsContainer.appendChild(card);
});

// Add a simple fade-in animation by injecting a style tag
const styleSheet = document.createElement("style");
styleSheet.type = "text/css";
styleSheet.innerHTML =
`@keyframes fade-in {
  from { opacity: 0; transform: translateY(10px); }
  to { opacity: 1; transform: translateY(0); }
}
.animate-fade-in {
  animation: fade-in 0.5s ease-out forwards;
}
`;
document.head.appendChild(styleSheet);
}
```

## EXPLAIANTION OF CODE:

### 1. index.html (The "Skeleton")

This file builds the **structure and content** of your webpage.

- **<head> section:**

- It loads the "Inter" font from Google to make the text look modern.
- It loads **Tailwind CSS**, which is a toolkit that lets us style elements using class names (like bg-white, rounded-lg, etc.).
- It links to your style.css file for custom styles.
- It links to your script.js file at the very end to load the "brains" of the page.

- **<body> section:**

- It creates the main product box, including the product's image (a placeholder), its category (id="product-category"), name (id="product-name"), and description (id="product-desc").
- It creates the blue "Get AI Recommendations" button (id="recommend-btn").
- It creates a hidden loading spinner (id="loading").
- It creates an empty div (id="results-container") that acts as a placeholder. This is where the JavaScript will inject the new AI-generated recommendations.

In short: This file is the static, visible layout of the page *before* any AI magic happens.

---

### 2. style.css (The "Clothes")

This file adds **custom visual styles** that are not covered by the main Tailwind CSS toolkit.

- body { ... }: This tells the entire page to use the "Inter" font that was loaded in the index.html.
- .spinner { ... }: This defines what the loading spinner looks like. It makes a round, light-gray border and then makes the *top* border blue (border-left-color).
- @keyframes spin { ... }: This defines an *animation* called "spin." It simply tells the spinner to rotate 360 degrees over and over. The .spinner class uses this animation.

In short: This file just adds the custom font and the spinning animation for the loading icon.

---

### 3. script.js (The "Brains")

This file contains all the **logic and interactivity**. This is where the AI-assisted algorithm lives. It waits for you to do something and then reacts.

Here is the flow, step-by-step:

1. **Get Elements:** The script starts by "grabbing" the important HTML elements it needs to control:
  - recommendBtn: The button.
  - loadingSpinner: The hidden spinner.
  - resultsContainer: The empty results box.
2. **Wait for a Click:**
  - recommendBtn.addEventListener('click', getAiRecommendations);
  - This line is the main trigger. It tells the browser, "When the user clicks the recommendBtn, run the getAiRecommendations function."
3. **Define the "AI Algorithm" (The Schema):**
  - const RECOMMENDATION\_SCHEMA = ...
  - This is the *most important* part. It's a strict set of rules. It tells the AI: "You MUST reply in a JSON format. The JSON must contain a list called recommendations, and *each* item in that list MUST have a productName, a category, and a reason."
  - This is what makes the AI a reliable tool. It forces the AI's response to be perfectly structured so the code can understand it.
4. **The Main Function (getAiRecommendations):** This runs when the button is clicked.
  - **Step 1:** It shows the loading spinner and disables the button (so you can't click it 100 times).
  - **Step 2:** It reads the product's name and description from the HTML page.
  - **Step 3:** It writes the instructions for the AI (the "prompts"):
    - systemPrompt: The AI's "job" ("You are an expert retail merchandiser. Your goal is to increase sales...").
    - userQuery: The AI's "task" ("Here is the product... give me 3 recommendations.").

- **Step 4:** It bundles these prompts and the RECOMMENDATION\_SCHEMA into a payload object.
- **Step 5:** It sends this payload to the Gemini API using the fetchWithRetry function (a helper that safely retries if the network fails).
- **Step 6:** When the AI responds, it gets the structured JSON data, parses it, and sends the list of recommendations to the renderRecommendations function.
- **Step 7:** Finally, it hides the spinner and re-enables the button.

##### 5. The "Builder" Function (renderRecommendations):

- This function receives the list of 3 recommendations from the AI.
- It loops through the list (e.g., "Wool Socks," "Rain Pants," "Boot Spray").
- For each item, it **builds a new HTML "card"** (a div) using the product's name, category, and the AI-generated reason.
- It then **injects** this new card into the results-container on the page, making it appear to the user.
- It also adds a little CSS animation to make the new cards fade in smoothly.

## SAMPLE OUTPUT:

The screenshot shows a CodePen interface with three panels: HTML, CSS, and JS. The HTML panel contains a grid structure with a button and a script tag. The CSS panel includes a spin animation for the button. The JS panel contains code for getting element references, adding event listeners, and executing the algorithm. Below the editor is a preview window titled "AI 'Cold Start' Recommendation". It features a blue sidebar with "New Product" text and a main content area for "MEN'S FOOTWEAR" showing a product card for "Trail-Ready Hiking Boots".

```
HTML
class="grid grid-cols-1 md:grid-cols-3 gap-6">
    <!-- AI-generated recommendations will be injected here by the script -->
    </div>
</div>
</div>
<!-- Link to your new JavaScript file -->
<script src="script.js"></script>
</body>
</html>
```

```
CSS
border: 4px solid rgba(0, 0, 0, 0.1);
width: 36px;
height: 36px;
border-radius: 50%;
border-left-color: #0ea5e9;
animation: spin 1s ease infinite;
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

```
JS
// --- 1. Get Element References ---
const recommendBtn = document.getElementById('recommend-btn');
const loadingSpinner = document.getElementById('loading');
const resultsContainer = document.getElementById('results-container');
// --- 2. Add Event Listener ...
recommendBtn.addEventListener('click', getAiRecommendations);
// --- 3. The "Algorithm": AI Schema for Structured JSON Output ...
```

This screenshot shows the same CodePen setup as the previous one, but the preview window has been modified. It now includes a blue button at the bottom labeled "Get AI Recommendations".

```
HTML
class="grid grid-cols-1 md:grid-cols-3 gap-6">
    <!-- AI-generated recommendations will be injected here by the script -->
    </div>
</div>
</div>
<!-- Link to your new JavaScript file -->
<script src="script.js"></script>
</body>
</html>
```

```
CSS
border: 4px solid rgba(0, 0, 0, 0.1);
width: 36px;
height: 36px;
border-radius: 50%;
border-left-color: #0ea5e9;
animation: spin 1s ease infinite;
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

```
JS
// --- 1. Get Element References ---
const recommendBtn = document.getElementById('recommend-btn');
const loadingSpinner = document.getElementById('loading');
const resultsContainer = document.getElementById('results-container');
// --- 2. Add Event Listener ...
recommendBtn.addEventListener('click', getAiRecommendations);
// --- 3. The "Algorithm": AI Schema for Structured JSON Output ...
```

