Grade Calculator Example in JavaScript

JavaScript + HTML Code

Understand MVC Architecture

The Model (M), View (V), Controller (C) matches well with the web applications.

- Model: HTML
- View: CSS
- Controller: JavaScript

Another interpretation of MVC

- Model: Data Model
- View: User Interface
- Controller: Application Logic (Also called Business Logic)

Data Model, UI, and Application Logic

Also, when we design software we need to identify the "Data Model", "User Interface", and "Application Logic".

- Data Model: The data structure that all of the application components should share
- User Interface: The components that users use to interact with the application
- Application Logic: The code that processes data and updates the UI

Divid and Conquer

We need to isolate these elements as much as possible:

- We should be able to replace UI anytime necessary (we call this View).
- We need to isolate Data Model so that it can be easily extended and maintained (We can support API to access the core data structure).
- We need to isolate application logic so that we can use any different alogirthms whenever necessary (we call this Strategy pattern).

View (HTML)

- Three inputs and one button

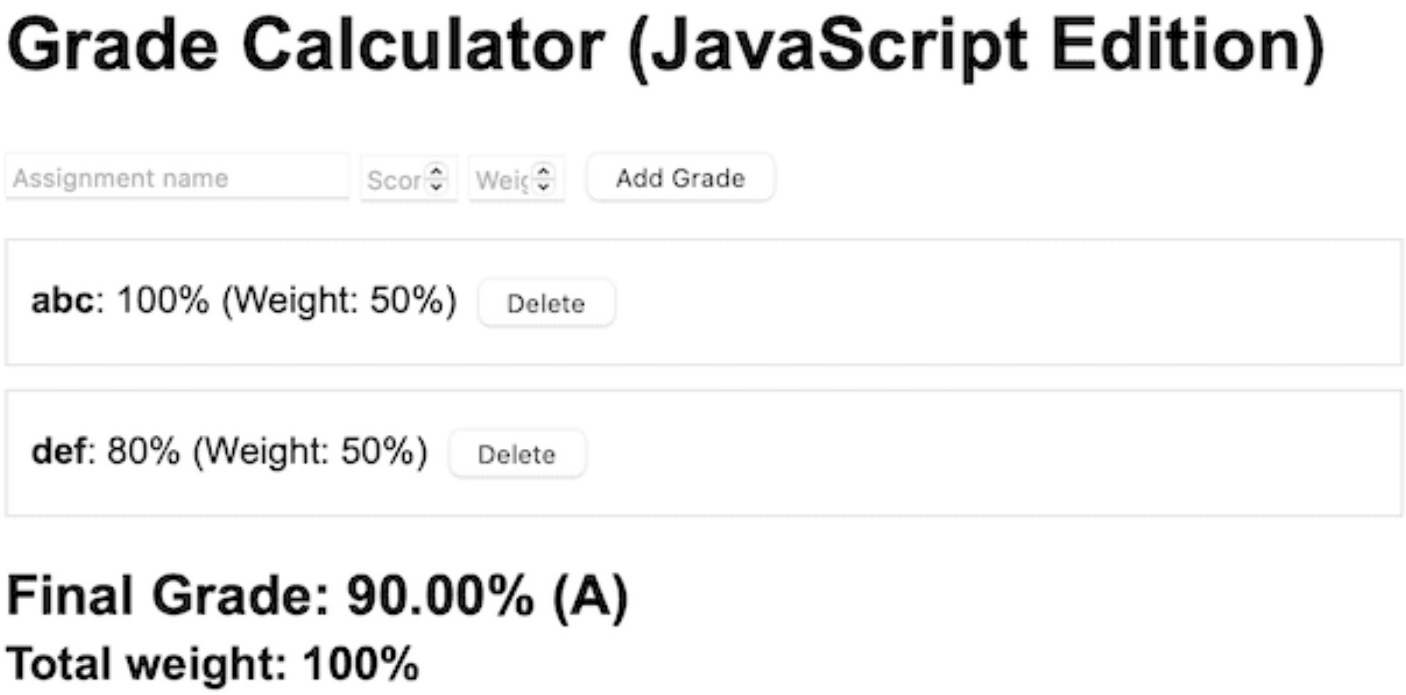
```
<h1>Grade Calculator (JavaScript Edition)</h1>

<div id="grade-form">
  <input type="text" id="assignment-name" placeholder="Assignment name" />
  <input type="number" id="score" placeholder="Score" min="0" max="100" />
  <input type="number" id="weight" placeholder="Weight %" min="0" max="100" />
  <button onclick="addGrade()">Add Grade</button>
</div>

<div id="grades-list"></div>
<div id="result" class="result"></div>

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

With two grades, 100 (50%) and 80 (50%), the final value is calculated and displayed: 100 x 0.5 + 80 x 0.5 = 90 (A)



Data Model (Data Structure)

```
// Grade Calculator Application
let grades = [];
```

Application Logic (Business Logic)

```
// Add a new grade
function addGrade() {
  // Get input values
  const nameInput = document.getElementById('assignment-name');
  const scoreInput = document.getElementById('score');
  const weightInput = document.getElementById('weight');

  const name = nameInput.value.trim();
  const score = parseFloat(scoreInput.value);
  const weight = parseFloat(weightInput.value);

  // Validation
  if (!name || isNaN(score) || isNaN(weight)) {
    alert('Please fill in all fields correctly');
    return;
  }
```

```
  // Create grade object
  const grade = {
    id: Date.now(),
    name: name,
    score: score,
    weight: weight
  };

  // Add to array
  grades.push(grade);

  // Clear inputs
  nameInput.value = '';
  scoreInput.value = '';
  weightInput.value = '';

  // Update display
  displayGrades();
  calculateFinalGrade();
}
```

```
// Delete a grade
function deleteGrade(id) {
  grades = grades.filter((grade) => grade.id !== id);
  displayGrades();
  calculateFinalGrade();
}
```

```
// Convert percentage to letter grade
function getLetterGrade(percentage) {
  if (percentage >= 90) return "A";
  if (percentage >= 80) return "B";
  if (percentage >= 70) return "C";
  if (percentage >= 60) return "D";
  return "F";
}
```

High-Quality Software Product

To build high-quality software product, we need to build software that is easy to extend and fix bugs.

- However, using vanilla JavaScript, it is sometime hard to accomplish this goal.
- This is where frameworks and libraries come in to help.

However, in most cases, novice software engineers just use HTML/JavaScript to write logic and UI at the same time.

UI Logic + Application Logic

Displaying information logic is intermingled with Application Logic.

1. Access the information from HTML using ID.
2. Store the information back to the HTML.

```
// Display all grades
function displayGrades() {
  const gradesList = document.getElementById("grades-list");

  gradesList.innerHTML = grades
    .map(
      (grade) => `<div class="grade-item">
            <strong>${grade.name}</strong>:
            ${grade.score}% (Weight: ${grade.weight}%)
            <button onclick="deleteGrade(${grade.id})">Delete</button>
        </div>
    `
    )
    .join("");
}
```

Calculate final grade: Application logic + UI logic combined

```
function calculateFinalGrade() {
  const resultDiv = document.getElementById("result");

  if (grades.length === 0) {
    resultDiv.innerHTML = "No grades yet";
    return;
  }

  // Calculate weighted average
  const totalWeight = grades.reduce((sum, grade) => sum + grade.weight, 0);

  if (totalWeight === 0) {
    resultDiv.innerHTML = "Total weight must be greater than 0";
    return;
  }

  const weightedSum = grades.reduce((sum, grade) => {
    return sum + grade.score * grade.weight;
  }, 0);

  const finalGrade = weightedSum / totalWeight;
  const letterGrade = getLetterGrade(finalGrade);

  resultDiv.innerHTML = `
Final Grade: ${finalGrade.toFixed(2)}% (${letterGrade})
        <br>
        <small>Total weight: ${totalWeight}%</small>
    `;
}
```

Event Handler

Add grade when Enter is pressed:

```
// Keyboard support
document.addEventListener("DOMContentLoaded", () => {
  const inputs = document.querySelectorAll("input");
  inputs.forEach((input) => {
    input.addEventListener("keypress", (e) => {
      if (e.key === "Enter") {
        addGrade();
      }
    });
  });
});
```

Question

- We solved the problem, but it is not the best solution as we do not isolate MVC components enough.
- In other words, this is not high-quality software products.
- What is the best way to solve this issue?
- What are your thoughts?
  solve this issue?
- What are your thoughts?
- What are the answers from LLMs?