TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

LAB REPORT

JS Lab Assignment

Submitted by:
Name: Chandan Kumar Shah
Roll No: 080BCT023
Department: III/I

Submitted to:
Department of Electronics
and Computer Engineering

GitHub Repository:
`https://github.com/chandan11248/web_programming_lab`

Contents

1 Assignment 1: Smart Calculator with History

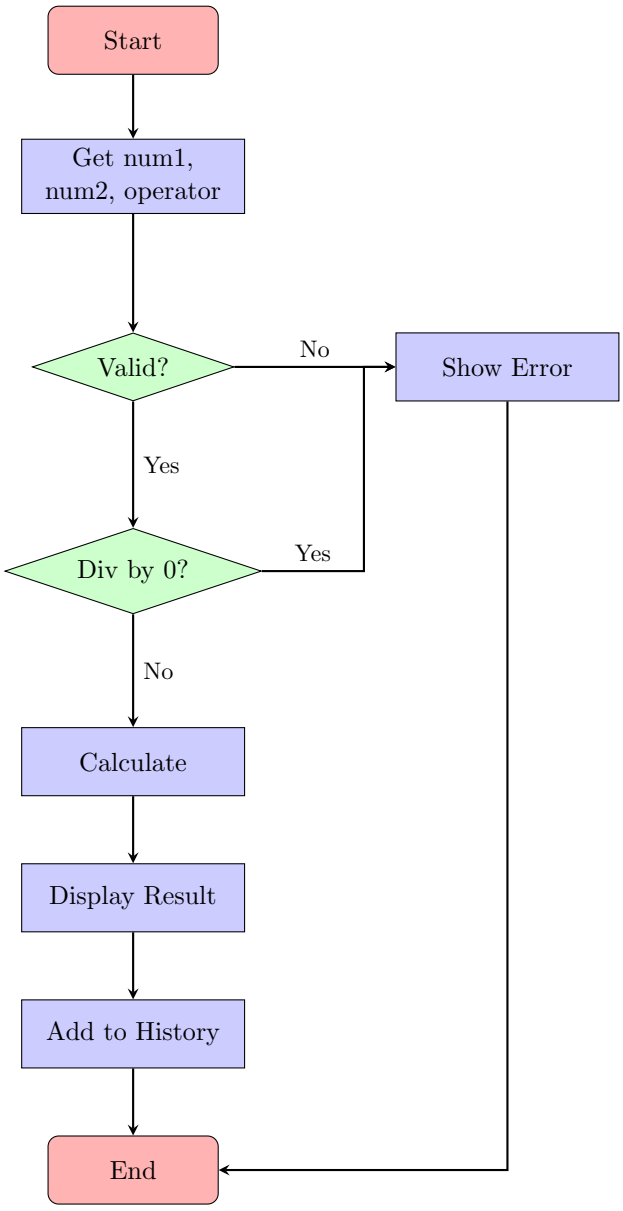
1.1 Problem Understanding

Design a calculator that performs basic arithmetic operations (addition, subtraction, multiplication, division), stores the last 5 calculations in history, displays calculation history dynamically, and allows clearing history. The calculator must handle invalid inputs such as division by zero.

1.2 List of JavaScript Concepts Used

- **Variables (let):** For storing numbers, results, and history array
- **Arrays:** To store calculation history (push, pop, unshift)
- **Functions:** calculate(), showResult(), addToHistory(), displayHistory(), clearHistory()
- **Switch Statement:** To handle different arithmetic operators
- **DOM Manipulation:** getElementById(), textContent, innerHTML
- **Input Validation:** isNaN() to check for valid numbers
- **Template Literals:** For string formatting with ${}
- **Array Methods:** forEach() for looping through history

1.3 Flow Diagram



1.4 Source Code

```
let history = [];

function appendToDisplay(value) {
    const display = document.getElementById('display');
    display.value += value;
}

function clearDisplay() {
    document.getElementById('display').value = '';
}

function deleteLast() {
    const display = document.getElementById('display');
    display.value = display.value.slice(0, -1);
```

```
}

function calculateResult() {
    const display = document.getElementById('display');
    const expression = display.value;

    try {
        const result = eval(expression);
        display.value = result;
        addToHistory(expression + " = " + result);

    } catch (error) {
        display.value = 'Error';
    }
}

function addToHistory(entry) {
    history.unshift(entry);
    if (history.length > 5) {
        history.pop();
    }
    updateHistoryUI();
}

function updateHistoryUI() {
    const historyList = document.getElementById('historyList');
    historyList.innerHTML = '';
    for (let i = 0; i < history.length; i++) {
        const item = history[i];
        const li = document.createElement('li');
        li.textContent = item;
        historyList.appendChild(li);
    }
}

function clearHistory() {
    history = [];
    updateHistoryUI();
}
```

Listing 1: Calculator Core Functions

1.5 Output Screenshots

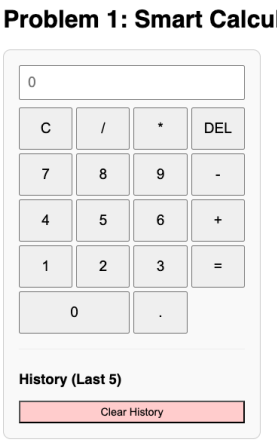


Figure 1: Smart Calculator with History

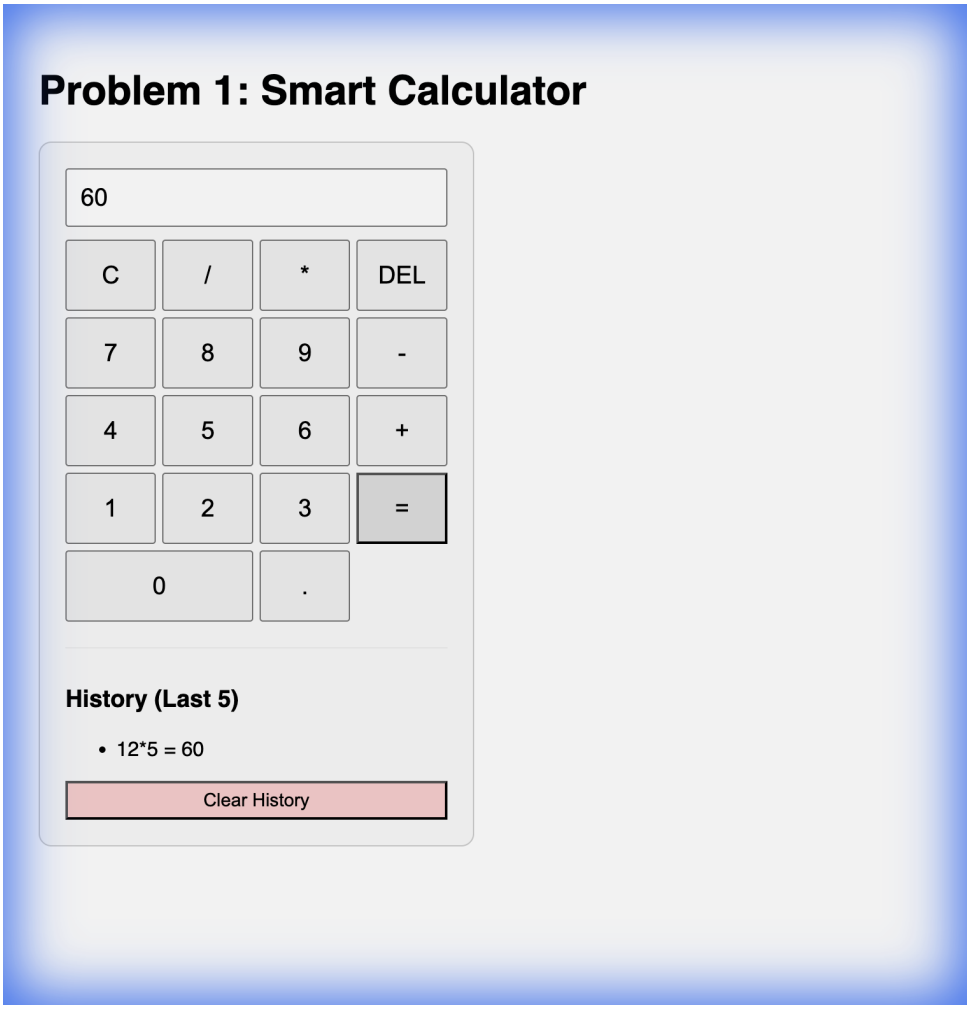


Figure 2: Calculator displaying calculation history

2 Assignment 2: Student Marks & Grade Analyzer

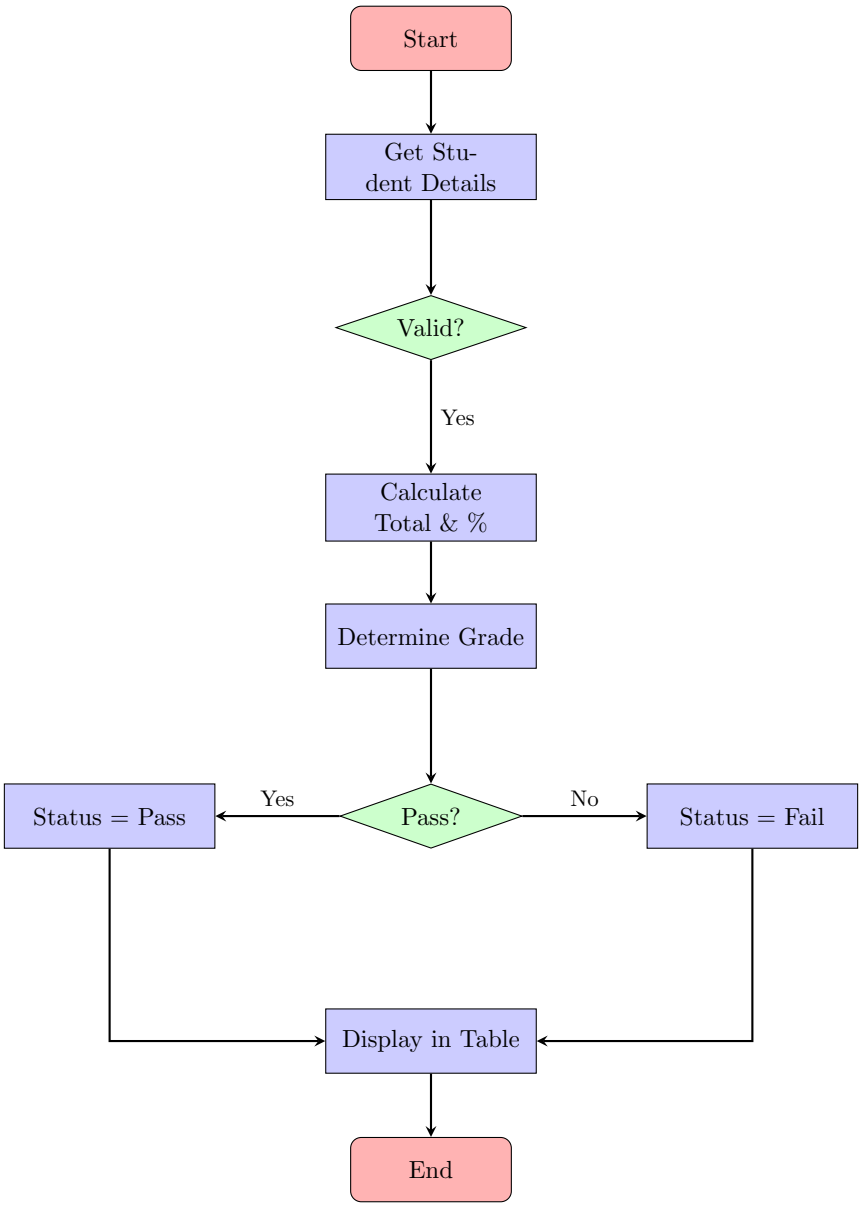
2.1 Problem Understanding

Create a system where student details are stored as objects, marks for multiple subjects are entered, total, percentage, and grade are calculated, results are displayed in a table, and pass/fail rows are highlighted using CSS via JavaScript.

2.2 List of JavaScript Concepts Used

- **Objects:** To store student data (name, rollNo, marks, grade)
- **Array of Objects:** students[] array holding multiple student objects
- **Functions:** addStudent(), calculateGrade(), displayResults(), clearForm()
- **If-Else Statements:** For grade calculation based on percentage
- **DOM Manipulation:** createElement(), appendChild(), innerHTML
- **CSS Class Manipulation:** classList.add() for pass/fail highlighting
- **parseInt():** Convert string input to integer
- **Ternary Operator:** For pass/fail status determination

2.3 Flow Diagram



2.4 Source Code

```
let students = [];

function addStudent() {
    const name = document.getElementById('name').value;
    const math = parseFloat(document.getElementById('math').value);
    const science = parseFloat(document.getElementById('science').value);
    const english = parseFloat(document.getElementById('english').value);

    if (name === "" || isNaN(math) || isNaN(science) || isNaN(english)) {
        alert("Please fill in all fields correctly.");
        return;
```

```
    }

    const total = math + science + english;
    const percentage = (total / 300) * 100;

    let grade = "";
    if (percentage >= 90) {
        grade = "A";
    } else if (percentage >= 80) {
        grade = "B";
    } else if (percentage >= 70) {
        grade = "C";
    } else if (percentage >= 60) {
        grade = "D";
    } else {
        grade = "F";
    }

    const student = {
        name: name,
        total: total,
        percentage: percentage.toFixed(2),
        grade: grade
    };

    students.push(student);
    addStudentToTable(student);

    document.getElementById('name').value = '';
    document.getElementById('math').value = '';
    document.getElementById('science').value = '';
    document.getElementById('english').value = '';
}

function addStudentToTable(student) {
    const tableBody = document.querySelector('#resultTable tbody');
    const row = document.createElement('tr');

    if (student.grade === "F") {
        row.classList.add('fail');
    } else {
        row.classList.add('pass');
    }

    row.innerHTML = `
        <td>${student.name}</td>
        <td>${student.total}</td>
        <td>${student.percentage}%</td>
        <td>${student.grade}</td>
    `;

    tableBody.appendChild(row);
}
```

Listing 2: Student Grade Analyzer Functions

2.5 Output Screenshots

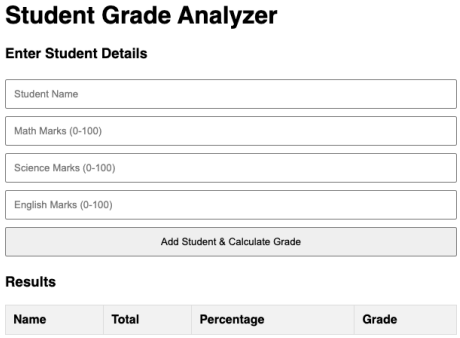


Figure 3: Student Marks & Grade Analyzer

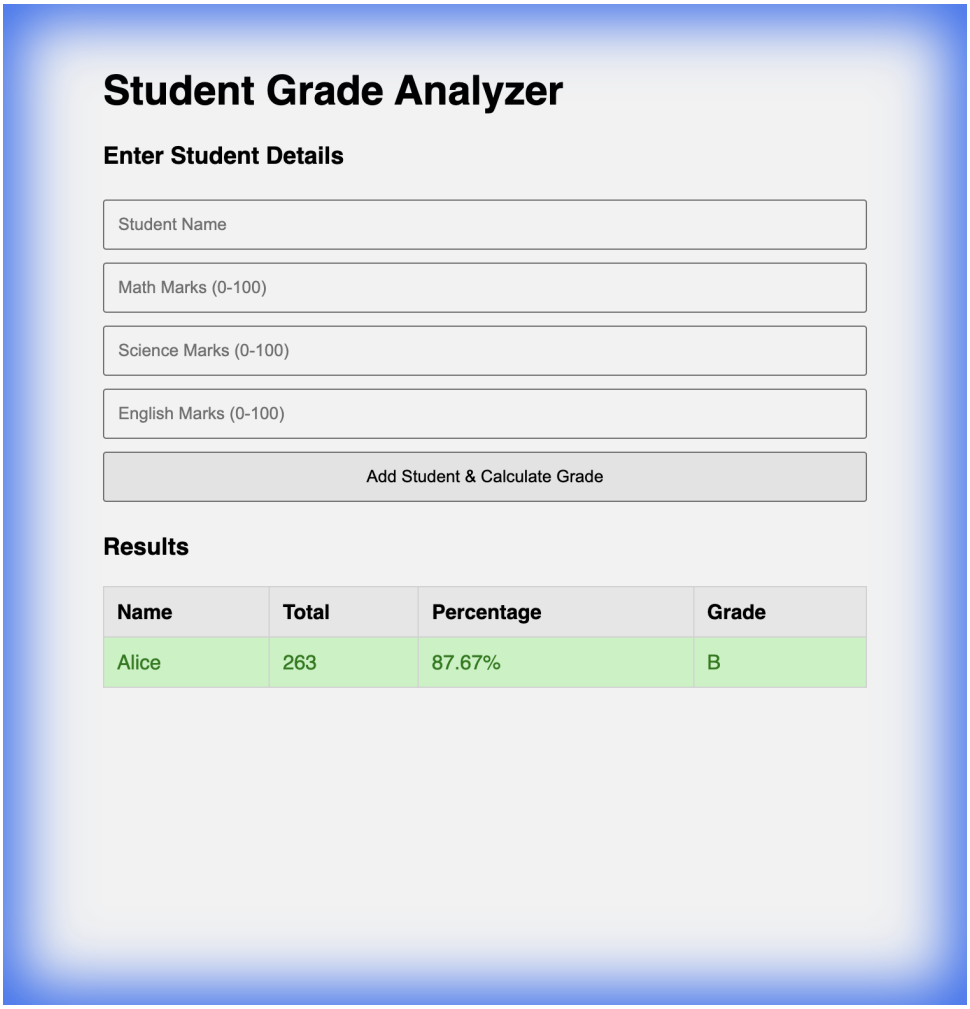


Figure 4: Grade Analyzer with student data added

3 Assignment 3: Dynamic To-Do List with Status Filter

3.1 Problem Understanding

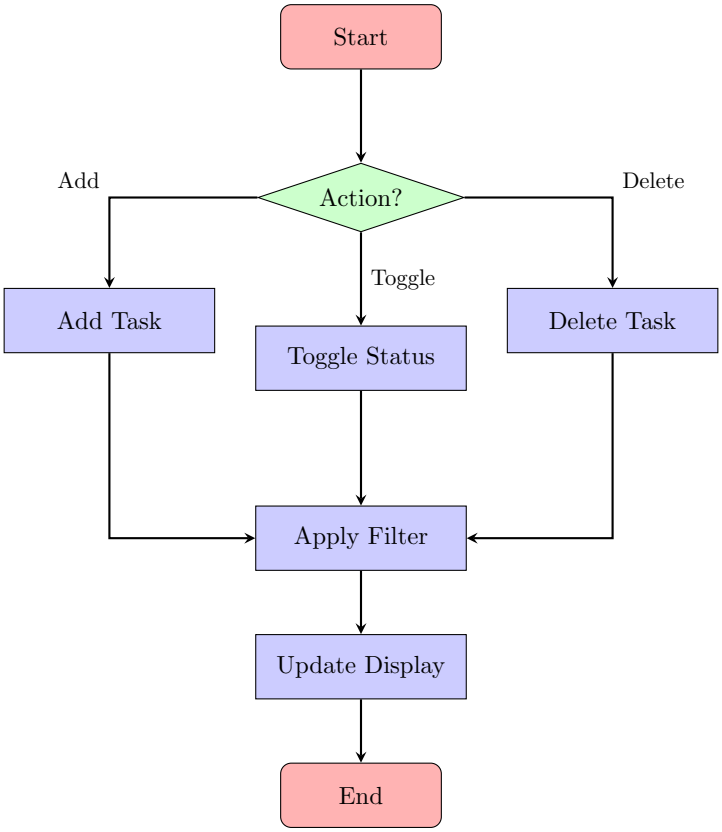
Build a To-Do application that allows users to add, delete, and mark tasks as completed, filter tasks by status (All/Completed/Pending), and display count of completed tasks.

3.2 List of JavaScript Concepts Used

- **Arrays:** To store task objects
- **Objects:** Task objects with id, text, completed properties
- **Date.now():** Generate unique IDs for tasks
- **filter() Method:** Filter tasks by completion status
- **Boolean Toggle:** !task.completed to toggle status
- **Event Listeners:** For keyboard input (Enter key)

- **Template Literals:** Dynamic HTML generation

3.3 Flow Diagram



3.4 Source Code

```
let tasks = [];

function addTask() {
    const taskInput = document.getElementById('taskInput');
    const taskText = taskInput.value;

    if (taskText === '') {
        alert("Please enter a task!");
        return;
    }

    const task = {
        id: Date.now(),
        text: taskText,
        completed: false
    };

    tasks.push(task);
    taskInput.value = '';

    renderTasks();
}

```

```
function toggleComplete(id) {
    for (let i = 0; i < tasks.length; i++) {
        if (tasks[i].id === id) {
            tasks[i].completed = !tasks[i].completed;
            break;
        }
    }
    renderTasks();
}

function deleteTask(id) {
    tasks = tasks.filter(task => task.id !== id);
    renderTasks();
}

let currentFilter = 'all';

function filterTasks(status) {
    currentFilter = status;
    renderTasks();
}

function renderTasks() {
    const taskList = document.getElementById('taskList');
    taskList.innerHTML = '';

    let completedCount = 0;

    for (let i = 0; i < tasks.length; i++) {
        const task = tasks[i];

        if (currentFilter === 'active' && task.completed) continue;
        if (currentFilter === 'completed' && !task.completed) continue;

        if (task.completed) completedCount++;

        const li = document.createElement('li');
        if (task.completed) {
            li.classList.add('completed');
        }

        li.innerHTML = `
            <span onclick="toggleComplete(${task.id})" style="cursor:
               pointer;">${task.text}</span>
            <button onclick="deleteTask(${task.id})" style="background:
                #ffcccc; border: none; color: red;">X</button>
        `;

        taskList.appendChild(li);
    }

    document.getElementById('completedCount').innerText =
       completedCount;
}
```

Listing 3: To-Do List Core Functions

3.5 Output Screenshots

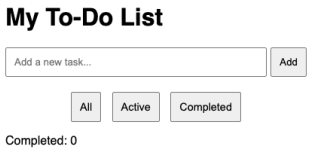


Figure 5: Dynamic To-Do List with Status Filter

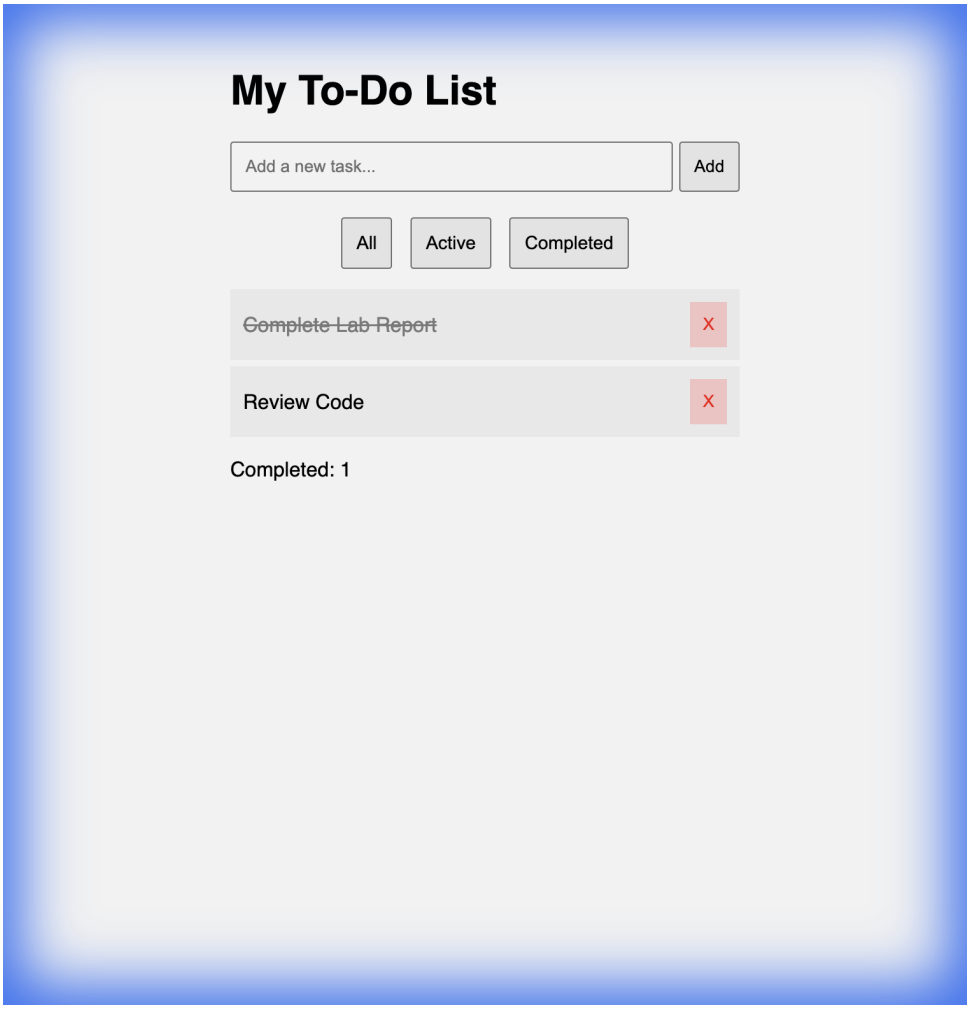


Figure 6: To-Do List showing active and completed tasks

4 Assignment 4: Number Guessing Game

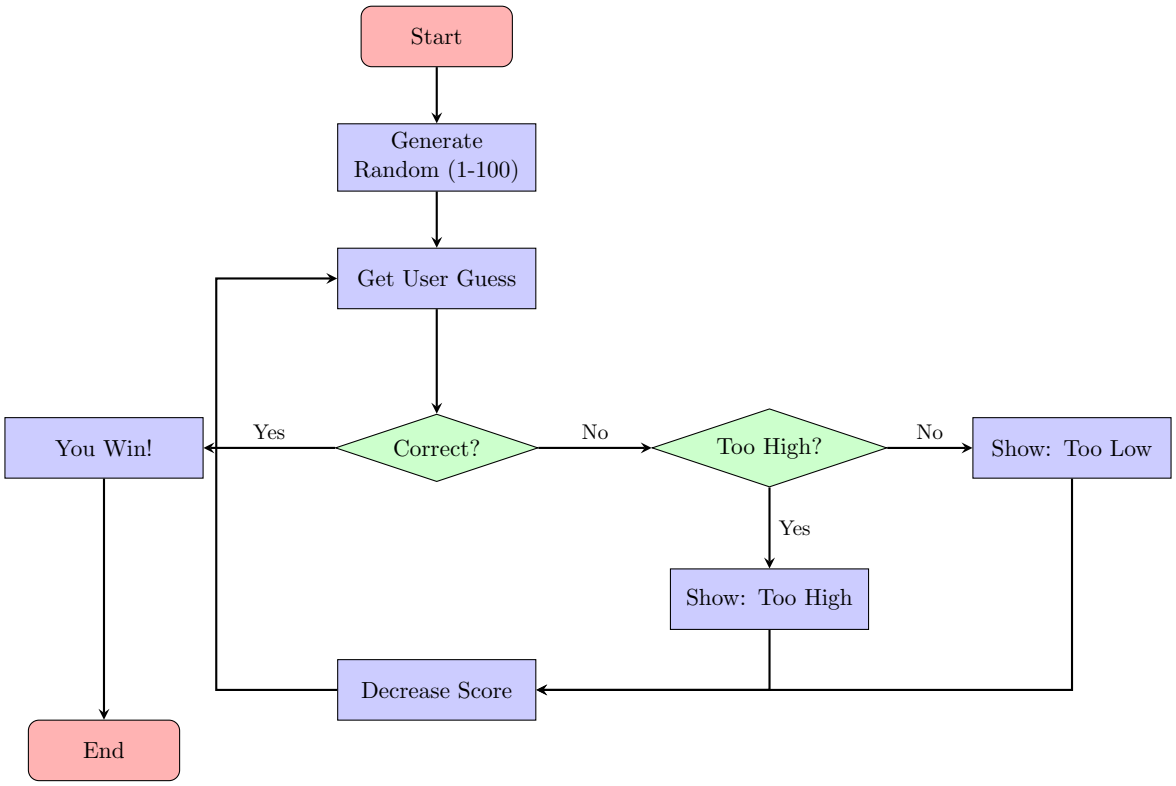
4.1 Problem Understanding

Develop a game where the system generates a random number between 1-100, user guesses the number, feedback is shown (High/Low/Correct), and score decreases on wrong attempts.

4.2 List of JavaScript Concepts Used

- **Math.random():** Generate random number
- **Math.floor():** Round down to integer
- **Comparison Operators:** ===, ¿, ¡ for guess checking
- **Boolean Variables:** gameOver flag
- **Score Tracking:** Decrement score on wrong guesses
- **Input Validation:** Range checking (1-100)

4.3 Flow Diagram



4.4 Source Code

```
let secretNumber;
let score = 100;
let attempts = 0;
// Variables to store game state
let secretNumber = Math.floor(Math.random() * 100) + 1;
let score = 100;

function checkGuess() {
    const input = document.getElementById('guessInput');
    const guess = Number(input.value);
    const messageDisplay = document.getElementById('message');
    const scoreDisplay = document.getElementById('score');

    if (!guess || guess < 1 || guess > 100) {
        messageDisplay.innerText = "Please enter a valid number between
            1 and 100.";
        messageDisplay.style.color = "red";
        return;
    }

    if (guess === secretNumber) {
        messageDisplay.innerText = "Correct! You won!       ";
        messageDisplay.style.color = "green";
        document.body.style.backgroundColor = "#ccffcc";
    } else if (guess > secretNumber) {
        messageDisplay.innerText = "Too High! Try again.";
        messageDisplay.style.color = "orange";
        score -= 10;
```

```
    } else {
        messageDisplay.innerText = "Too Low! Try again.";
        messageDisplay.style.color = "orange";
        score -= 10;
    }

    if (score <= 0) {
        messageDisplay.innerText = "Game Over! The number was " +
            secretNumber;
        messageDisplay.style.color = "red";
        score = 0;
    }

    scoreDisplay.innerText = score;

    input.value = '';
    input.focus();
}

function restartGame() {
    score = 100;
    secretNumber = Math.floor(Math.random() * 100) + 1;

    document.getElementById('score').innerText = score;
    document.getElementById('message').innerText = "";
    document.getElementById('guessInput').value = "";
    document.body.style.backgroundColor = "white";
}
```

Listing 4: Number Guessing Game Functions

4.5 Output Screenshots

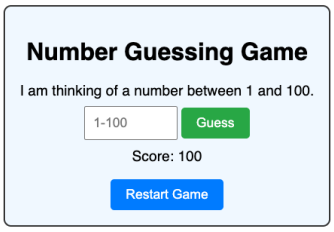


Figure 7: Number Guessing Game

5 Assignment 5: Interactive Quiz Application

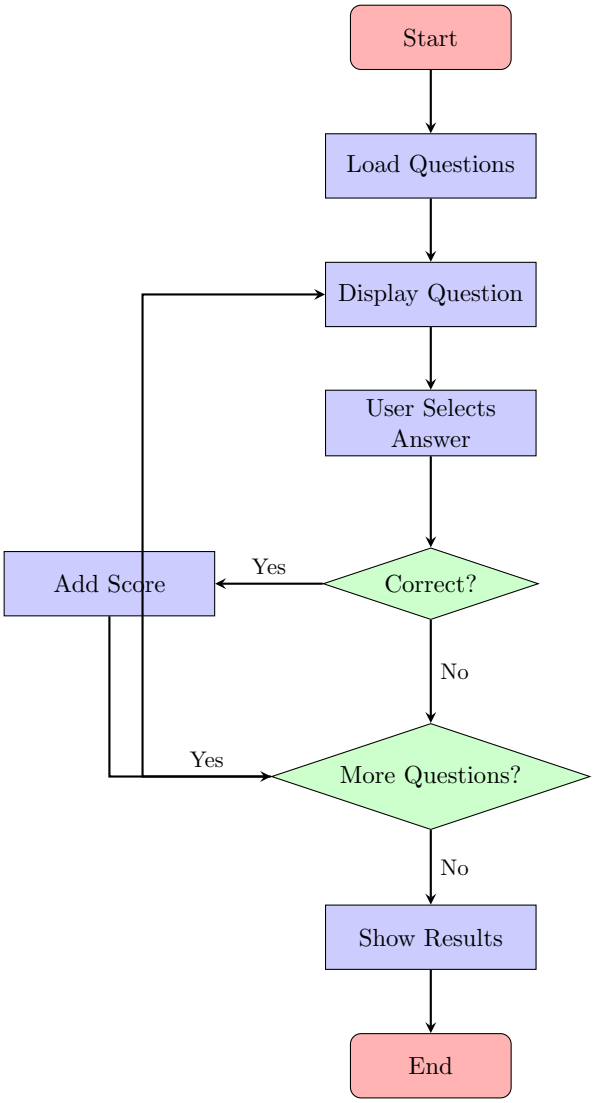
5.1 Problem Understanding

Create a quiz app where questions are stored as objects, one question is shown at a time, user selects an answer, and score is calculated and shown at the end.

5.2 List of JavaScript Concepts Used

- **const:** For immutable questions array
- **Array of Objects:** Questions with options and correct index
- **Array Index Access:** questions[currentQuestionIndex]
- **String.fromCharCode():** Generate A, B, C, D labels
- **querySelectorAll():** Select multiple elements
- **classList Manipulation:** For correct/wrong styling

5.3 Flow Diagram



5.4 Source Code

```
const questions = [
    {
        question: "What is the capital of France?",
        options: ["Berlin", "Madrid", "Paris", "Lisbon"],
        answer: "Paris"
    },
    {
        question: "Which language runs in a web browser?",
        options: ["Java", "C", "Python", "JavaScript"],
        answer: "JavaScript"
    },
    {
        question: "What does CSS stand for?",
        options: ["Central Style Sheets", "Cascading Style Sheets", "
           Cascading Simple Sheets", "Cars SUVs Sailboats"],
        answer: "Cascading Style Sheets"
    },
    {
```

```
        question: "What year was JavaScript launched?",
        options: ["1996", "1995", "1994", "None of the above"],
        answer: "1995"
    }
];

let currentQuestionIndex = 0;
let score = 0;

loadQuestion();

function loadQuestion() {
    const currentQuestion = questions[currentQuestionIndex];

    document.getElementById('question').innerText = currentQuestion.
       question;

    const optionsDiv = document.getElementById('options');
    optionsDiv.innerHTML = '';

    for (let i = 0; i < currentQuestion.options.length; i++) {
        const option = currentQuestion.options[i];

        const button = document.createElement('button');
        button.innerText = option;
        button.onclick = function () { selectAnswer(option); };
        optionsDiv.appendChild(button);
    }
}

function selectAnswer(selectedOption) {
    const currentQuestion = questions[currentQuestionIndex];

    if (selectedOption === currentQuestion.answer) {
        score++;
        alert("Correct!");
    } else {
        alert("Wrong! The correct answer was: " + currentQuestion.
           answer);
    }

    document.getElementById('nextBtn').style.display = 'block';

    const buttons = document.querySelectorAll('.options button');
    for (let i = 0; i < buttons.length; i++) {
        buttons[i].disabled = true;
    }
}

function nextQuestion() {
    currentQuestionIndex++;

    if (currentQuestionIndex < questions.length) {
        loadQuestion();
        document.getElementById('nextBtn').style.display = 'none';
    } else {
        showResult();
    }
```

```
74 }
75
76 function showResult() {
77     document.getElementById('quiz').style.display = 'none'; // Hide
       quiz
78     document.getElementById('nextBtn').style.display = 'none';
79
80     const resultDiv = document.getElementById('result');
81     resultDiv.innerText = "You scored " + score + " out of " +
       questions.length;
82 }
```

Listing 5: Quiz Application Functions

5.5 Output Screenshots

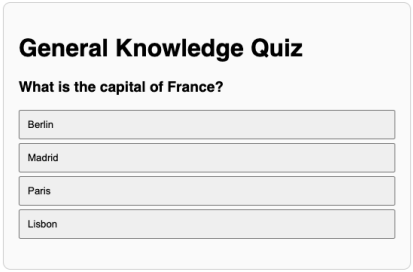


Figure 8: Interactive Quiz Application

6 Assignment 6: Digital Clock & Countdown Timer

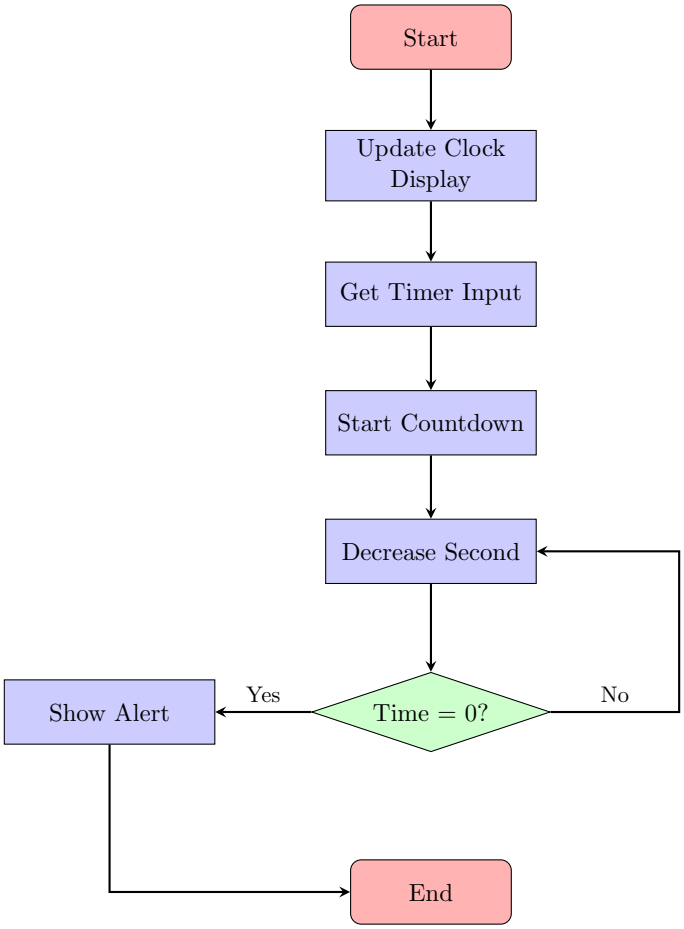
6.1 Problem Understanding

Create a real-time digital clock and a countdown timer that takes user input and alerts when time reaches zero.

6.2 List of JavaScript Concepts Used

- **Date Object:** new Date() for current time
- **Date Methods:** getHours(), getMinutes(), getSeconds()
- **setInterval():** Execute function repeatedly
- **clearInterval():** Stop the interval
- **padStart():** Format numbers with leading zeros
- **alert():** Show popup when timer ends

6.3 Flow Diagram



6.4 Source Code

```
// --- Digital Clock Logic ---

function updateClock() {
    const now = new Date();

    let hours = now.getHours();
    let minutes = now.getMinutes();
    let seconds = now.getSeconds();

    hours = hours.toString().padStart(2, '0');
    minutes = minutes.toString().padStart(2, '0');
    seconds = seconds.toString().padStart(2, '0');

    const timeString = \`\${hours}:\${minutes}:\${seconds}\`;
    document.getElementById('clock').innerText = timeString;
}

setInterval(updateClock, 1000);
updateClock();

// --- Countdown Timer Logic ---

let timerInterval;

function startTimer() {
    clearInterval(timerInterval);

    const minInput = document.getElementById('minutes').value;
    const secInput = document.getElementById('seconds').value;

    let totalTimeInSeconds = (Number(minInput) * 60) + Number(secInput)
        ;

    if (totalTimeInSeconds <= 0) {
        alert("Please enter a valid time.");
        return;
    }

    displayTime(totalTimeInSeconds);

    timerInterval = setInterval(function () {
        totalTimeInSeconds--;

        displayTime(totalTimeInSeconds);

        if (totalTimeInSeconds <= 0) {
            clearInterval(timerInterval);
            alert("Time's Up!");
        }
    }, 1000);
}

function displayTime(seconds) {
    const m = Math.floor(seconds / 60);
    const s = seconds % 60;

```

```
    const mString = m.toString().padStart(2, '0');
    const sString = s.toString().padStart(2, '0');

    document.getElementById('timerDisplay').innerText = \`\${mString}:\
        ${sString}\`;
}
```

Listing 6: Clock and Timer Functions

6.5 Output Screenshots

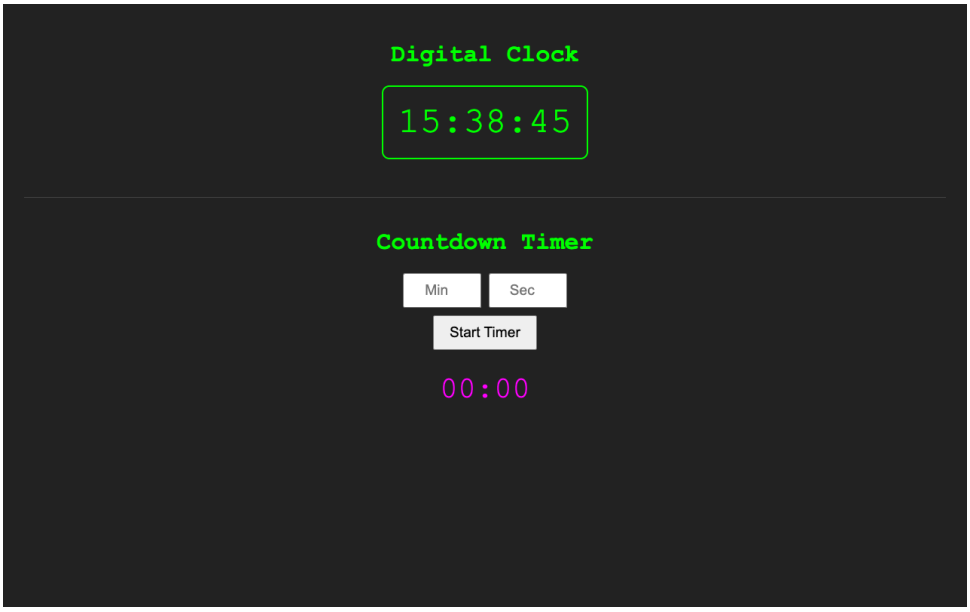


Figure 9: Digital Clock & Countdown Timer

7 Assignment 7: Dynamic Table Generator

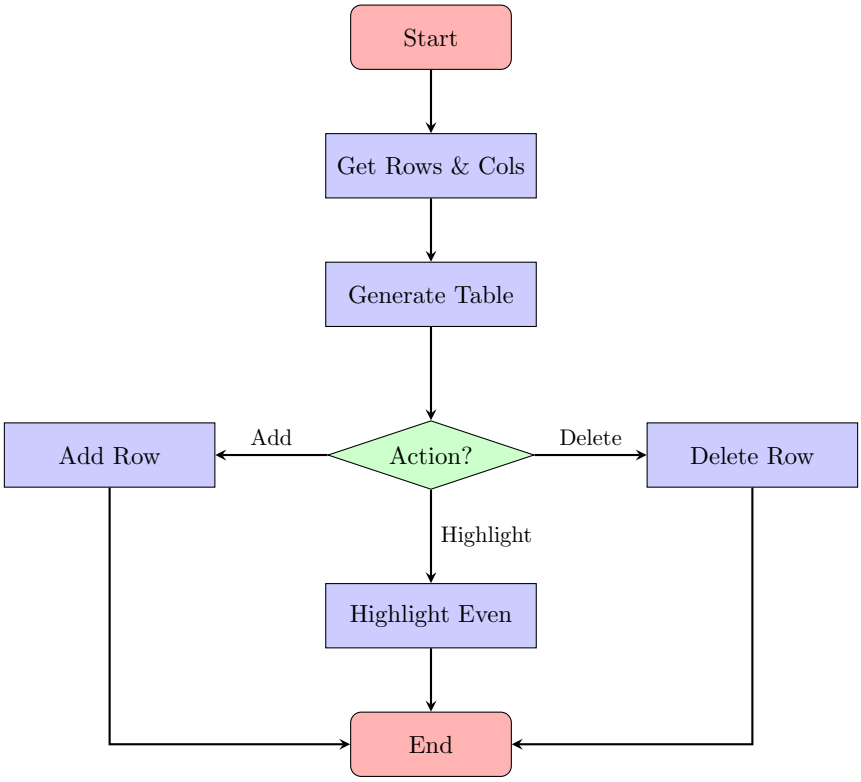
7.1 Problem Understanding

Design a program that takes number of rows and columns as input, generates a table dynamically, and adds buttons to add row, delete row, and highlight even rows.

7.2 List of JavaScript Concepts Used

- **Nested Loops:** For creating rows and columns
- **createElement():** Create HTML elements dynamically
- **appendChild():** Add elements to DOM
- **getElementsByTagName():** Select elements by tag
- **Modulo Operator (%):** Check for even/odd rows
- **lastElementChild:** Access last child element
- **removeChild():** Remove elements from DOM

7.3 Flow Diagram



7.4 Source Code

```
function generateTable() {
    const rows = document.getElementById('rows').value;
    const cols = document.getElementById('cols').value;
    const container = document.getElementById('tableContainer');

    const table = document.createElement('table');

    for (let i = 0; i < rows; i++) {
        const tr = document.createElement('tr');

        for (let j = 0; j < cols; j++) {
            const td = document.createElement('td');
            td.innerText = "Row " + (i + 1) + ", Col " + (j + 1);
            tr.appendChild(td);
        }

        table.appendChild(tr);
    }

    container.innerHTML = '';
    container.appendChild(table);
}

function highlightEven() {
    const rows = document.querySelectorAll('table tr');

    for (let i = 0; i < rows.length; i++) {
        if (i % 2 !== 0) {
            rows[i].classList.toggle('even-row');
        }
    }
}

function addRow() {
    const table = document.querySelector('table');

    if (!table) {
        alert("Please generate a table first!");
        return;
    }

    const newRow = table.insertRow();

    let colCount = 3;
    if (table.rows.length > 1) {
        colCount = table.rows[0].cells.length;
    } else {
        colCount = document.getElementById('cols').value;
    }

    const rowIndex = table.rows.length;

    for (let i = 0; i < colCount; i++) {
        const cell = newRow.insertCell();
        cell.innerText = "Row " + rowIndex + ", Col " + (i + 1);
    }
}
```

```
58
59 function deleteRow() {
60     const table = document.querySelector('table');
61
62     if (!table || table.rows.length === 0) {
63         alert("No rows to delete!");
64         return;
65     }
66
67     table.deleteRow(table.rows.length - 1);
68 }
```

Listing 7: Dynamic Table Functions

7.5 Output Screenshots

Figure 10: Dynamic Table Generator

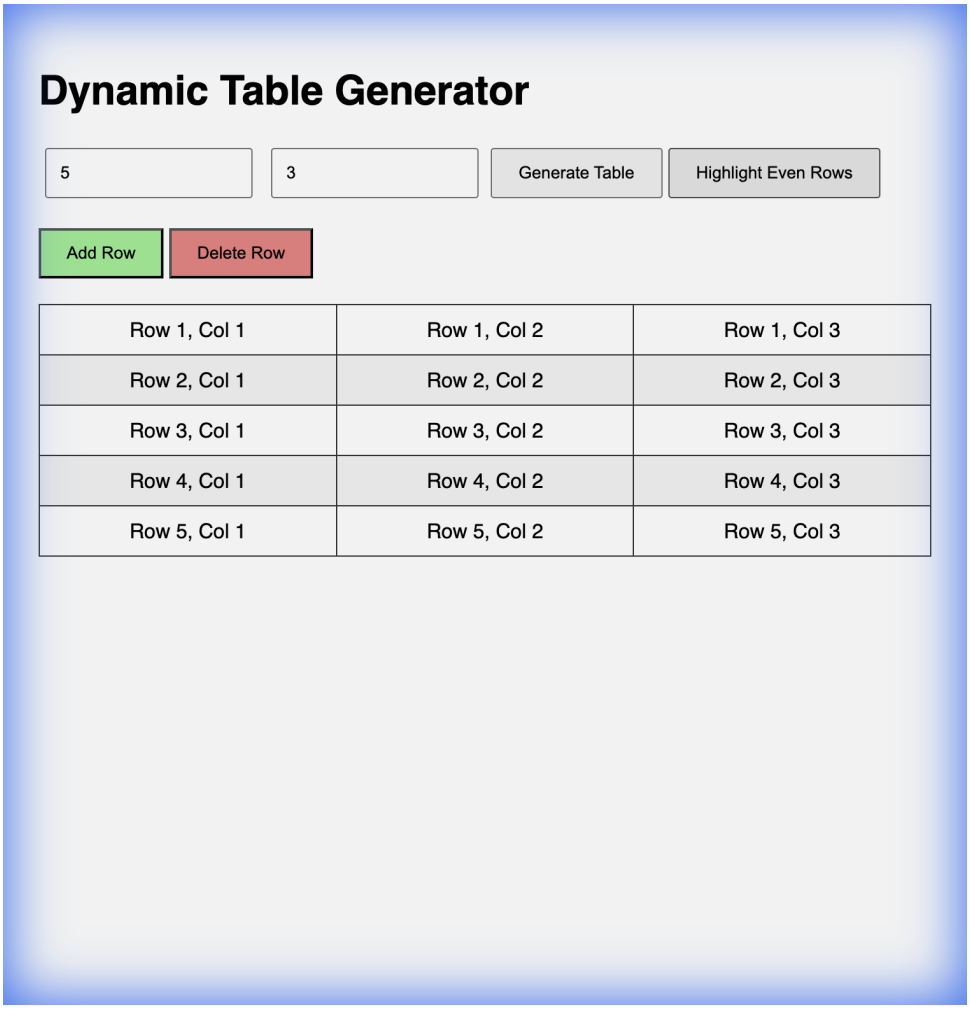


Figure 11: Dynamic Table with highlighted even rows

8 Assignment 8: Simple E-Commerce Cart

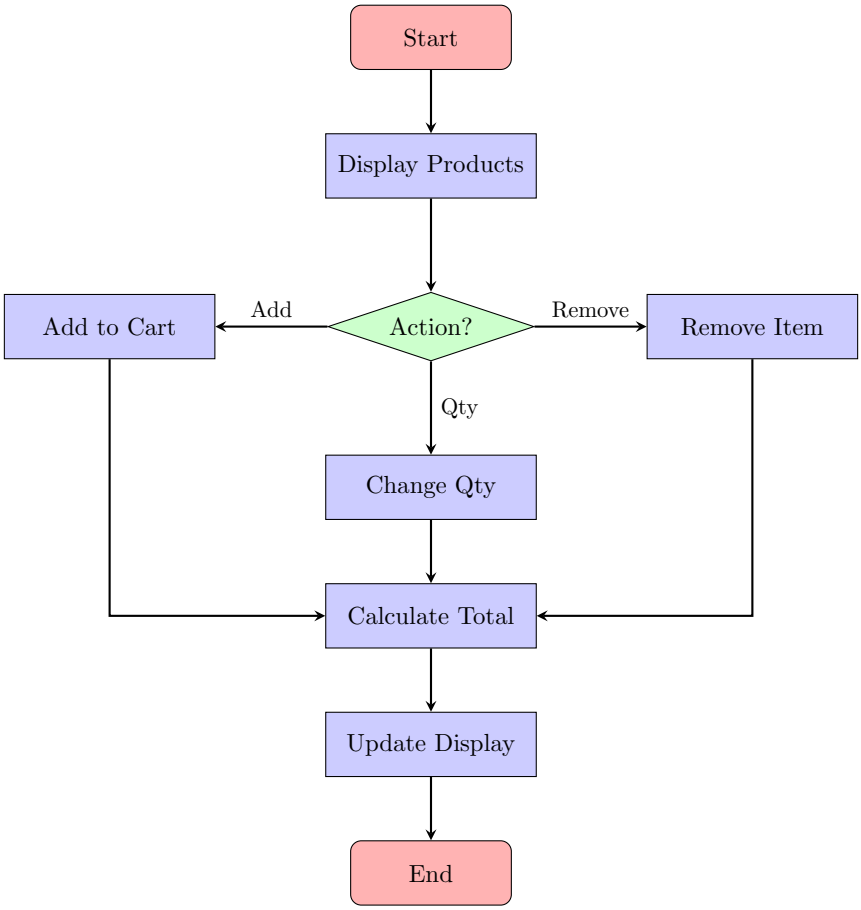
8.1 Problem Understanding

Create a shopping cart where products are stored as objects, user can add/remove items, total price updates dynamically, and quantity changes affect total.

8.2 List of JavaScript Concepts Used

- **Array of Product Objects:** Store product data
- **find() Method:** Locate product by ID
- **toFixed():** Format currency to 2 decimal places
- **filter() Method:** Remove items from cart
- **Accumulator Pattern:** Calculate total price
- **Dynamic Updates:** Real-time cart total calculation

8.3 Flow Diagram



8.4 Source Code

```
const products = [
    { id: 1, name: 'Laptop', price: 999.99, emoji: '       ' },
    { id: 2, name: 'Phone', price: 699.99, emoji: '      ' }
];

let cart = [];

function addToCart(productId) {
    let product = products.find(p => p.id === productId);
    let existing = cart.find(item => item.id === productId);

    if (existing) {
        existing.quantity++;
    } else {
        cart.push({
            id: product.id, name: product.name,
            price: product.price, quantity: 1
        });
    }
    updateCartDisplay();
}

function calculateTotal() {
    let total = 0;
```

```
    cart.forEach(function(item) {
        total += item.price * item.quantity;
    });
    return total;
}

function removeFromCart(productId) {
    cart = cart.filter(item => item.id !== productId);
    updateCartDisplay();
}
```

Listing 8: Shopping Cart Functions

8.5 Output Screenshots

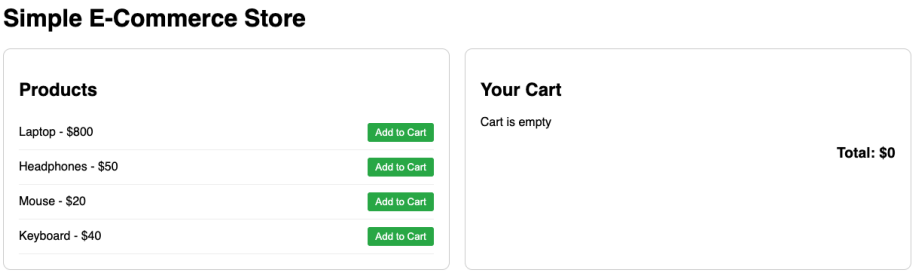


Figure 12: Simple E-Commerce Cart

9 Assignment 9: Form Validation System

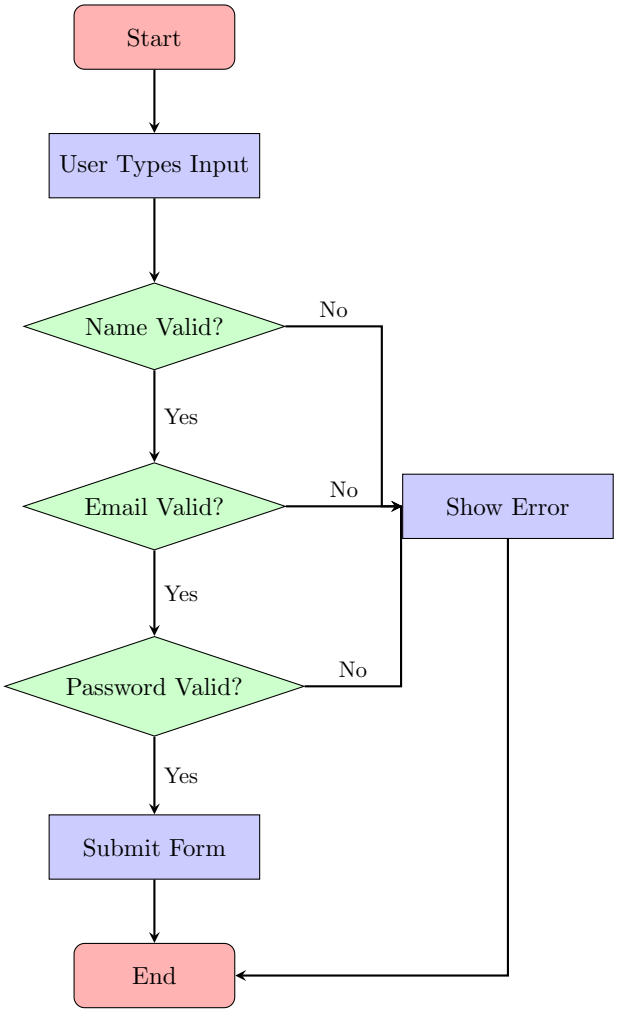
9.1 Problem Understanding

Develop a form that validates name, email, and password fields, shows error messages dynamically, and prevents submission if validation fails.

9.2 List of JavaScript Concepts Used

- **trim() Method:** Remove whitespace from inputs
- **Regular Expressions:** Email pattern validation
- **test() Method:** Check if string matches regex
- **oninput Event:** Real-time validation as user types
- **preventDefault():** Stop form default submission
- **classList Manipulation:** Show/hide error messages

9.3 Flow Diagram



9.4 Source Code

```
let isNameValid = false;
let isEmailValid = false;
let isPasswordValid = false;

function validateName() {
    let name = document.getElementById('name').value.trim();
    let errorDiv = document.getElementById('nameError');

    if (name.length >= 3) {
        isNameValid = true;
        errorDiv.classList.remove('show');
    } else {
        isNameValid = false;
        errorDiv.classList.add('show');
    }
}

function validateEmail() {
    let email = document.getElementById('email').value.trim();
    let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    isEmailValid = emailPattern.test(email);
}

function validateForm(event) {
    event.preventDefault();
    validateName();
    validateEmail();
    validatePassword();

    if (isNameValid && isEmailValid && isPasswordValid) {
        showSuccess();
    } else {
        alert('Please fix errors');
    }
}
```

Listing 9: Form Validation Functions

9.5 Output Screenshots

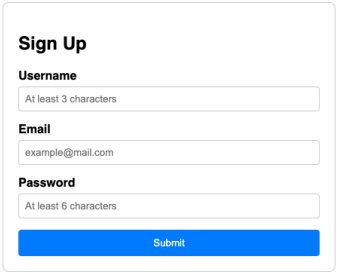


Figure 13: Form Validation System

10 Assignment 10: Color & Theme Manager

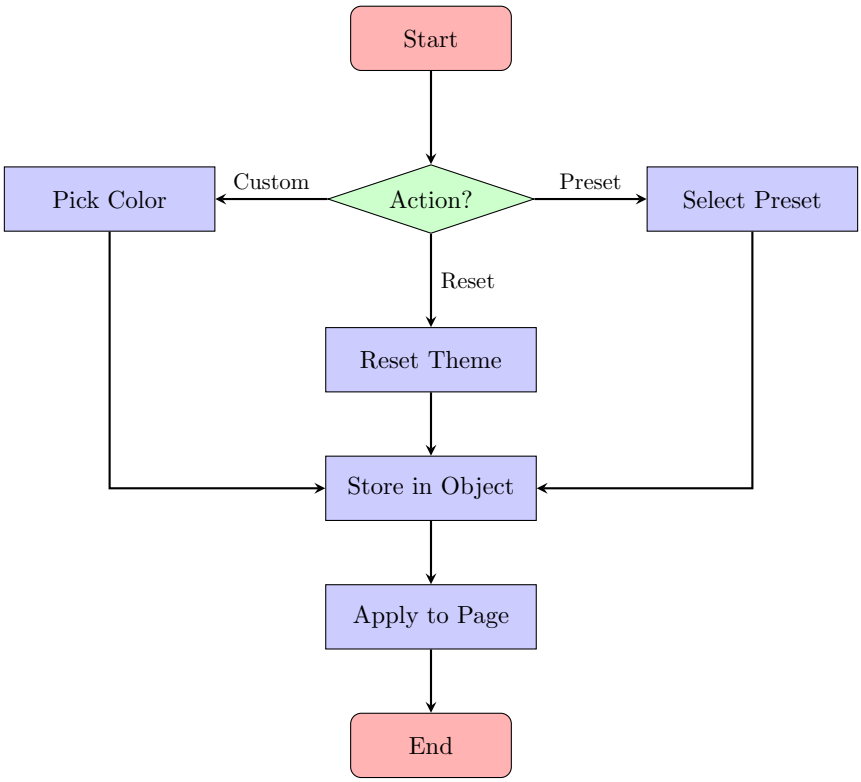
10.1 Problem Understanding

Create a webpage where user selects theme colors, background/text/button colors change dynamically, selected theme is stored in an object, and reset option restores default theme.

10.2 List of JavaScript Concepts Used

- **Objects:** Store current theme and preset themes
- **Color Input:** HTML5 color picker input type
- **style Property:** Modify CSS via JavaScript
- **Object Properties:** Access/modify theme values
- **Event Handling:** onchange for color picker
- **Preset Themes:** Object containing multiple theme configurations

10.3 Flow Diagram



10.4 Source Code

```
let currentTheme = {
    background: 'white',
    text: 'black'
};

function applyTheme() {
    const bgSelect = document.getElementById('bgColor').value;
    const textSelect = document.getElementById('textColor').value;

    document.body.style.backgroundColor = bgSelect;
    document.body.style.color = textSelect;

    currentTheme.background = bgSelect;
    currentTheme.text = textSelect;

    displayThemeStatus();
}

function resetTheme() {
    document.body.style.backgroundColor = 'white';
    document.body.style.color = 'black';

    document.getElementById('bgColor').value = 'white';
    document.getElementById('textColor').value = 'black';

    currentTheme = {
        background: 'white',
        text: 'black'
```

```
29      };
30
31      displayThemeStatus();
32  }
33
34  function displayThemeStatus() {
35      const log = document.getElementById('currentThemeLog');
36      log.innerText = "Current Theme Object:\n" + JSON.stringify(
            currentTheme, null, 2);
37  }
```

Listing 10: Theme Manager Functions

10.5 Output Screenshots

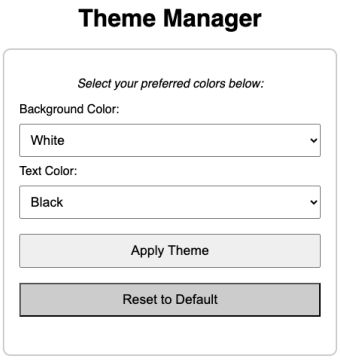


Figure 14: Color & Theme Manager

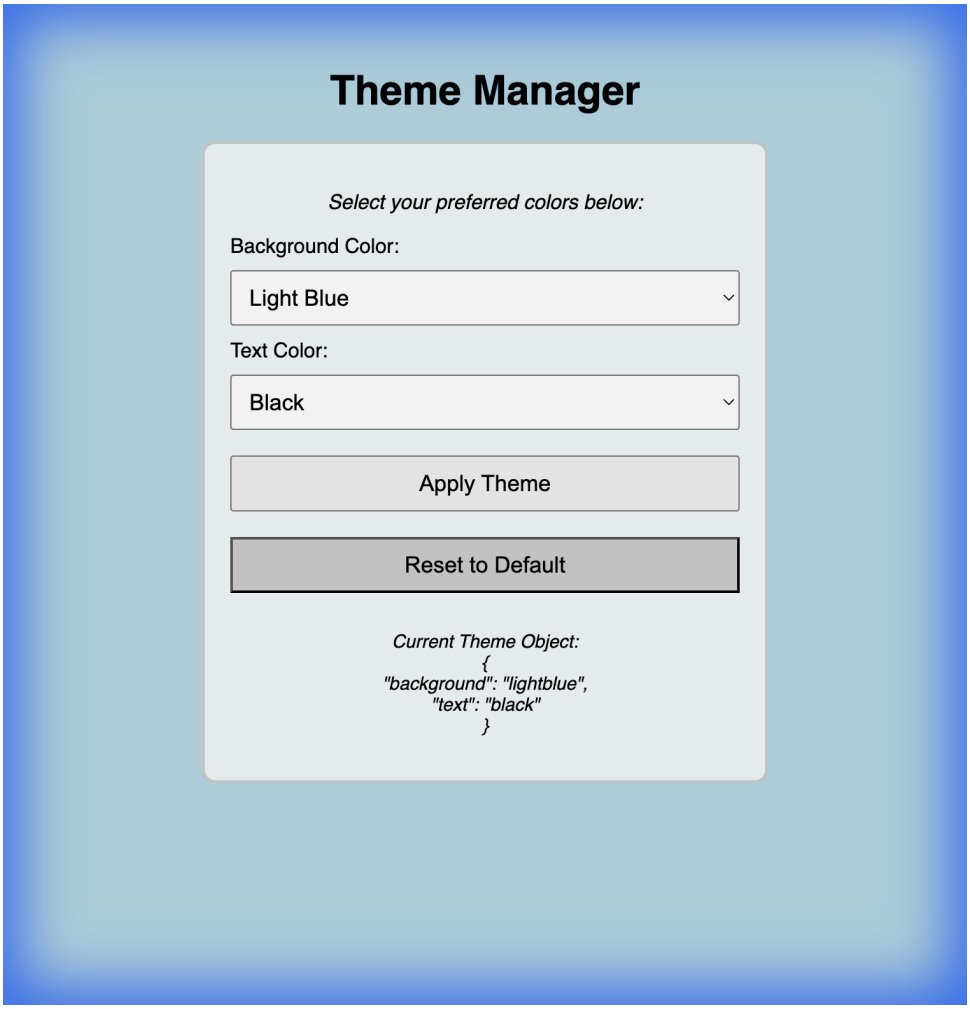


Figure 15: Theme Manager with blue background applied

11 Conclusion

In this lab, we successfully demonstrated the implementation of 10 different JavaScript applications using Vanilla JavaScript (without any external libraries or frameworks). Key concepts we have learned include:

- DOM manipulation for dynamic content updates
- Event handling for user interactions
- Array methods (push, pop, filter, find, forEach)
- Object-oriented data storage
- Form validation using regular expressions
- Timer functions (setInterval, clearInterval)
- CSS manipulation through JavaScript

We implemented all applications with clean, readable code and proper error handling to ensure robust functionality.