Experiment 2.3

Aim

To create a web-based, interactive drawing tool using Scalable Vector Graphics (SVG) and JavaScript, which allows a user to draw freehand lines on a canvas using mouse events.

Objective

The primary objective is to implement a responsive and functional drawing application that demonstrates a practical understanding of SVG elements, DOM manipulation, and JavaScript event handling, specifically mousedown, mousemove, and mouseup. The tool will enable users to draw, clear the canvas, and change the stroke color.

Theory

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. Unlike raster formats (like PNG or JPG) that use pixels, SVG uses mathematical descriptions of lines, curves, and shapes. This makes SVG images resolution-independent, meaning they can be scaled to any size without losing quality.

The drawing tool will utilize the <svg> element as its canvas. We will dynamically create <polyline> elements to represent the freehand lines. A polyline is a set of connected straight line segments. By updating the points attribute of a single polyline element in real-time, we can create the illusion of a continuous, freehand line.

The interactivity is managed through three core mouse events in JavaScript:

- **mousedown:** This event is triggered when the user presses a mouse button down. We will use it to initiate the drawing process, creating a new polyline and starting to record the mouse's position.
- **mousemove:** This event fires whenever the cursor moves. As long as the mouse button is held down (tracked by a flag), this event will continuously append the current cursor coordinates to the points attribute of the active polyline, effectively drawing the line.
- **mouseup:** This event occurs when the user releases the mouse button. We will use this to stop the drawing process and finalize the current line.

By combining these events, we can simulate the user's freehand drawing motion on the SVG canvas.

Procedure

1. **HTML Structure:** Set up a basic HTML page with a <div id="app"> container, an SVG element for the canvas, and controls for the drawing tool (e.g., color pickers, a "Clear" button).

2. **CSS Styling:** Style the canvas and controls to be visually appealing and user-friendly. Ensure the canvas fills the available space and the controls are easily accessible.
3. **JavaScript Logic:**
   - **Initialization:** Get references to the SVG canvas element and the control buttons.
   - **State Management:** Use a variable, like isDrawing, to track whether the user is currently drawing. Use another variable, like currentColor, to store the selected stroke color.
   - **mousedown handler:** When a mouse click is detected on the SVG canvas, set isDrawing to true. Create a new <polyline> element and add it to the SVG. Set its stroke and stroke-width attributes. Store the initial coordinates.
   - **mousemove handler:** If isDrawing is true, get the current mouse coordinates and append them to the points attribute of the active polyline element.
   - **mouseup handler:** When the mouse button is released, set isDrawing to false.
   - **Control Functions:** Implement event listeners for the "Clear" button to remove all <polyline> elements from the SVG, and for the color pickers to update the currentColor variable.

Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SVG Drawing Tool</title>
  <script src="https://cdn.tailwindcss.com"></script>
  <style>
    body {
      font-family: 'Inter', sans-serif;
      background-color: #f0f4f8;
      height: 100vh;
      display: flex;
      justify-content: center;
      align-items: center;
    }
    svg {
      cursor: crosshair;
      background-color: #ffffff;
      border-radius: 12px;
      box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
      touch-action: none; /* Prevents default touch gestures */
      user-select: none;
      width: 100%;
      height: 100%;
```

```
        }
    </style>
</head>
<body class="p-4">
    <div class="flex flex-col h-full w-full max-w-7xl mx-auto rounded-xl shadow-2xl
bg-gray-100 p-6">
        <!-- Controls -->
        <div class="flex flex-col sm:flex-row items-center justify-between mb-4 space-y-4
sm:space-y-0 sm:space-x-4 p-4 bg-white rounded-lg shadow">
            <h1 class="text-2xl font-bold text-gray-800">Interactive SVG Canvas</h1>
            <div class="flex items-center space-x-2">
                <label for="colorPicker" class="text-gray-600 font-medium">Color:</label>
                <input type="color" id="colorPicker" value="#000000" class="w-12 h-8
rounded-md border border-gray-300 cursor-pointer">
                <button id="clearBtn" class="bg-red-500 hover:bg-red-600 text-white
font-semibold py-2 px-4 rounded-lg shadow-md transition-colors duration-200">
                    Clear Canvas
                </button>
            </div>
        </div>

        <!-- SVG Drawing Area -->
        <div class="flex-grow rounded-xl overflow-hidden">
            <svg id="drawingCanvas" class="w-full h-full"></svg>
        </div>
    </div>

    <script>
        // Get references to the DOM elements
        const drawingCanvas = document.getElementById('drawingCanvas');
        const clearBtn = document.getElementById('clearBtn');
        const colorPicker = document.getElementById('colorPicker');

        // State variables
        let isDrawing = false;
        let activeLine = null;
        let currentColor = colorPicker.value;

        // Function to get mouse/touch coordinates relative to the SVG canvas
        function getMousePos(evt) {
            const rect = drawingCanvas.getBoundingClientRect();
            let x, y;
            if (evt.type.startsWith('touch')) {
```

```
        const touch = evt.touches[0];
        x = touch.clientX - rect.left;
        y = touch.clientY - rect.top;
    } else {
        x = evt.clientX - rect.left;
        y = evt.clientY - rect.top;
    }
    return { x, y };
}

// Mouse/Touch down event handler
const startDrawing = (evt) => {
    evt.preventDefault(); // Prevent default browser actions like scrolling
    isDrawing = true;

    // Create a new polyline element
    activeLine = document.createElementNS('http://www.w3.org/2000/svg', 'polyline');
    activeLine.setAttribute('points', '');
    activeLine.setAttribute('stroke', currentColor);
    activeLine.setAttribute('stroke-width', '4');
    activeLine.setAttribute('fill', 'none');
    activeLine.setAttribute('stroke-linecap', 'round');
    activeLine.setAttribute('stroke-linejoin', 'round');

    // Add the new line to the canvas
    drawingCanvas.appendChild(activeLine);

    // Get initial position and add it to the polyline's points
    const pos = getMousePos(evt);
    activeLine.setAttribute('points', `${pos.x},${pos.y}`);
};

// Mouse/Touch move event handler
const drawLine = (evt) => {
    if (!isDrawing) return;
    evt.preventDefault();

    const pos = getMousePos(evt);
    const currentPoints = activeLine.getAttribute('points');
    activeLine.setAttribute('points', `${currentPoints} ${pos.x},${pos.y}`);
};

// Mouse/Touch up event handler
```

```
        const stopDrawing = () => {
            isDrawing = false;
            activeLine = null;
        };

        // Clear canvas event handler
        clearBtn.addEventListener('click', () => {
            // Remove all polyline elements from the SVG
            while (drawingCanvas.firstChild) {
                drawingCanvas.removeChild(drawingCanvas.firstChild);
            }
        });

        // Color picker event handler
        colorPicker.addEventListener('change', (evt) => {
            currentColor = evt.target.value;
        });

        // Add mouse event listeners to the canvas
        drawingCanvas.addEventListener('mousedown', startDrawing);
        drawingCanvas.addEventListener('mousemove', drawLine);
        drawingCanvas.addEventListener('mouseup', stopDrawing);

        // Add touch event listeners for mobile devices
        drawingCanvas.addEventListener('touchstart', startDrawing);
        drawingCanvas.addEventListener('touchmove', drawLine);
        drawingCanvas.addEventListener('touchend', stopDrawing);
        drawingCanvas.addEventListener('touchcancel', stopDrawing);

        // Stop drawing if the mouse leaves the canvas while drawing
        drawingCanvas.addEventListener('mouseleave', () => {
            if (isDrawing) {
                stopDrawing();
            }
        });
    </script>
</body>
</html>
```

Learning Outcomes

1. **Understanding of Vector Graphics:** Students will gain a practical understanding of the

difference between vector and raster graphics and how SVG's XML-based format allows for scalable and interactive designs.

2. **JavaScript Event Handling:** The project reinforces the use of core JavaScript mouse event handlers (mousedown, mousemove, mouseup) and their role in creating dynamic, interactive user interfaces. It also introduces touch events for mobile compatibility.
3. **DOM Manipulation:** By dynamically creating and modifying SVG elements (<polyline>) using JavaScript, students will learn essential Document Object Model (DOM) manipulation techniques.