EXPERIMENT 6

Aim

To design and implement a web-based interactive drawing tool using SVG and JavaScript, where users can draw shapes dynamically with mouse event handling.

Objectives

- To understand JavaScript DOM manipulation for creating and modifying SVG elements.
- To implement mouse event handlers (mousedown, mousemove, mouseup) for interactive drawing.
- To enable real-time rendering of shapes (line, rectangle, polyline) on an SVG canvas.
- To allow multiple shapes to be drawn without reloading the page.
- To provide additional features such as undo, clear, and SVG export.

Theory

SVG (Scalable Vector Graphics):
SVG is an XML-based format used to define vector-based graphics that are scalable without loss of quality. It supports shapes such as rectangles, lines, circles, and paths, which can be manipulated via JavaScript.

JavaScript & DOM Manipulation:
JavaScript allows creating and modifying SVG elements dynamically using the Document Object Model (DOM). Functions like createElementNS() are used to create SVG nodes, while attributes such as x, y, width, height, and stroke define shape properties.

Mouse Events:
- mousedown → Detects the starting point of a shape.
- mousemove → Updates the shape as the cursor moves (real-time

rendering).
- mouseup → Finalizes the shape when the mouse is released.

CODE-

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello, World!</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
      <h1 class="title">Hello World! </h1>
      <p id="currentTime"></p>
      <script src="script.js"></script>
  </body>
</html><!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <title>Interactive SVG Drawing Tool</title>
  <style>
    :root{--bg:#f7fafc;--panel:#ffffff;--accent:#0ea5a4;--muted:#6b7280}
    html,body{height:100%;margin:0;font-family:Arial, Helvetica, sans-
serif;background:var(--bg);color:#111827}
    .app{max-width:1000px;margin:28px auto;padding:18px;}
    .top{display:flex;gap:12px;align-items:center;margin-bottom:12px}
    .panel{background:var(--panel);box-shadow:0 6px 18px
rgba(16,24,40,0.06);border-
radius:12px;padding:12px;display:flex;gap:8px;align-items:center}
    select,input[type=color],button{border:1px solid
#e6e9ee;padding:8px;border-radius:8px;background:white}
    label{font-size:13px;color:var(--muted);display:flex;flex-
direction:column;gap:6px}
    .canvas-wrap{border-radius:12px;overflow:hidden;border:2px solid
rgba(15,23,42,0.06)}
    svg{width:100%;height:600px;background:#fff;display:block;cursor:cross
hair}
    .toolbar-right{margin-left:auto;display:flex;gap:8px}
    .info{font-size:13px;color:var(--muted);margin-top:8px}
    button#download{background:var(--accent);color:white;border:none}
    .shape-list{max-height:120px;overflow:auto;padding:8px;border-
radius:8px;background:#fff;border:1px solid #eee}
```

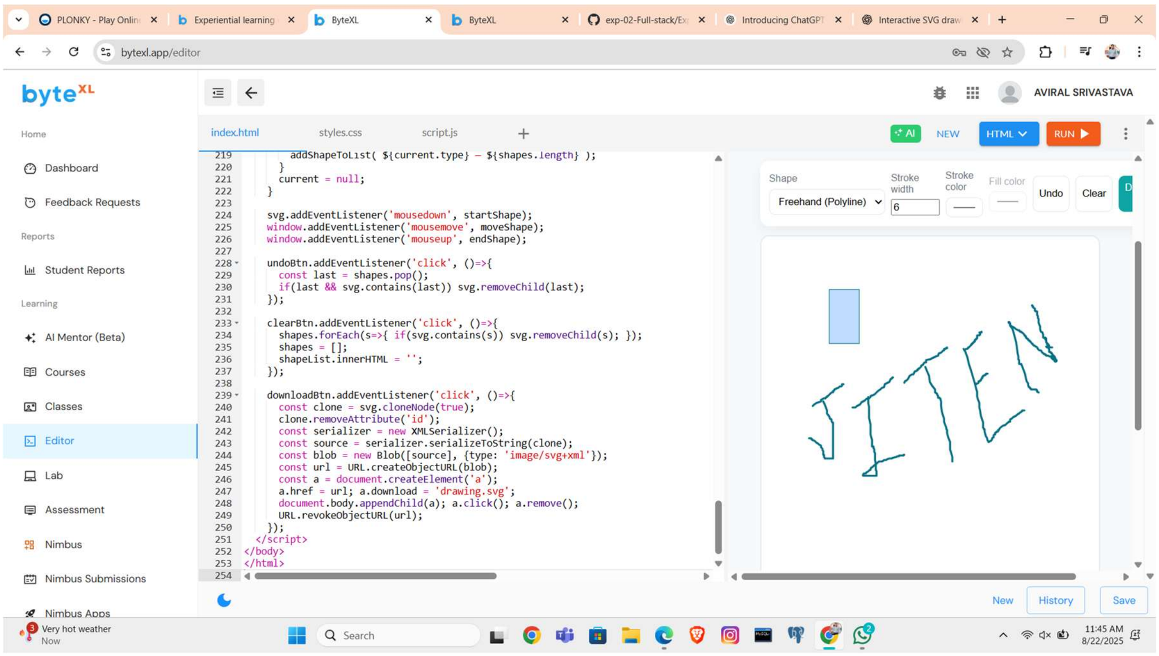
```
    .shape-item{font-size:13px;color:#374151;padding:6px;border-bottom:1px
dashed #f1f5f9}
  </style>
</head>
<body>
  <div class="app">
    <h2>Interactive SVG Drawing Tool</h2>
    <div class="top">
      <div class="panel">
        <label>Shape
          <select id="shapeSelect">
            <option value="rect">Rectangle</option>
            <option value="line">Line</option>
            <option value="polyline">Freehand (Polyline)</option>
          </select>
        </label>
        <label>Stroke width
          <input id="strokeWidth" type="number" min="1" max="20" value="2"
/>
        </label>
        <label>Stroke color
          <input id="strokeColor" type="color" value="#0b7285" />
        </label>
        <label id="fillLabel">Fill color
          <input id="fillColor" type="color" value="#60a5fa" />
        </label>
        <div class="toolbar-right">
          <button id="undo">Undo</button>
          <button id="clear">Clear</button>
          <button id="download">Download SVG</button>
        </div>
      </div>
    </div>

    <div class="canvas-wrap">
      <svg id="drawingArea" viewBox="0 0 1000 600"
xmlns="http://www.w3.org/2000/svg" preserveAspectRatio="xMidYMid
meet"></svg>
    </div>

    <div style="display:flex;gap:12px;margin-top:12px;align-items:flex-
start">
      <div style="flex:1">
        <div class="panel" style="flex-direction:column;gap:8px">
```

```
          <div class="info">Click + drag inside the canvas to draw. Draw
multiple shapes by repeating the action.</div>
        </div>
      </div>
      <div style="width:280px">
        <div class="panel" style="flex-direction:column;gap:8px">
          <strong style="font-size:14px">Shapes drawn</strong>
          <div id="shapeList" class="shape-list"></div>
        </div>
      </div>
    </div>
  </div>

  <script>
    function clientToSvg(svg, clientX, clientY){
      const pt = svg.createSVGPoint();
      pt.x = clientX; pt.y = clientY;
      const ctm = svg.getScreenCTM().inverse();
      return pt.matrixTransform(ctm);
    }

    const svg = document.getElementById('drawingArea');
    const shapeSelect = document.getElementById('shapeSelect');
    const strokeColor = document.getElementById('strokeColor');
    const fillColor = document.getElementById('fillColor');
    const strokeWidth = document.getElementById('strokeWidth');
    const fillLabel = document.getElementById('fillLabel');
    const undoBtn = document.getElementById('undo');
    const clearBtn = document.getElementById('clear');
    const downloadBtn = document.getElementById('download');
    const shapeList = document.getElementById('shapeList');

    let drawing = false;
    let current = null;
    let shapes = [];

    function updateFillVisibility(){
      const shape = shapeSelect.value;
      if(shape === 'line' || shape === 'polyline'){
        fillLabel.style.opacity = '0.6';
        fillLabel.style.pointerEvents = 'none';
      } else {
        fillLabel.style.opacity = '1';
        fillLabel.style.pointerEvents = 'auto';
      }
    }
    shapeSelect.addEventListener('change', updateFillVisibility);
    updateFillVisibility();

    function addShapeToList(entry){
      const el = document.createElement('div');
      el.className = 'shape-item';
      el.textContent = entry;
      shapeList.prepend(el);
    }

    function startShape(evt){
      if(evt.button !== 0) return;
      drawing = true;
      const p = clientToSvg(svg, evt.clientX, evt.clientY);
      const s = shapeSelect.value;
      const sw = parseFloat(strokeWidth.value) || 2;
      const sc = strokeColor.value;
      const fc = fillColor.value;

      if(s === 'rect'){
        const rect =
document.createElementNS('http://www.w3.org/2000/svg','rect');
        rect.setAttribute('x', p.x);
        rect.setAttribute('y', p.y);
        rect.setAttribute('width', 0);
        rect.setAttribute('height', 0);
        rect.setAttribute('stroke', sc);
        rect.setAttribute('stroke-width', sw);
        rect.setAttribute('fill', fc);
        rect.setAttribute('fill-opacity', 0.4);
        svg.appendChild(rect);
        current = {el:rect,type:'rect',start:p};
      } else if(s === 'line'){
        const line =
document.createElementNS('http://www.w3.org/2000/svg','line');
        line.setAttribute('x1', p.x);
        line.setAttribute('y1', p.y);
        line.setAttribute('x2', p.x);
        line.setAttribute('y2', p.y);
        line.setAttribute('stroke', sc);
        line.setAttribute('stroke-width', sw);
        line.setAttribute('stroke-linecap','round');
        svg.appendChild(line);
        current = {el:line,type:'line',start:p};
      } else if(s === 'polyline'){
        const pl =
document.createElementNS('http://www.w3.org/2000/svg','polyline');
        pl.setAttribute('points', `${p.x},${p.y}`);
        pl.setAttribute('fill','none');
        pl.setAttribute('stroke', sc);
        pl.setAttribute('stroke-width', sw);
        pl.setAttribute('stroke-linecap','round');
        pl.setAttribute('stroke-linejoin','round');
        svg.appendChild(pl);
        current = {el:pl,type:'polyline',points:[p]};
      }
    }

    function moveShape(evt){
      if(!drawing || !current) return;
      const p = clientToSvg(svg, evt.clientX, evt.clientY);
      if(current.type === 'rect'){
        const x = Math.min(current.start.x, p.x);
        const y = Math.min(current.start.y, p.y);
        const w = Math.abs(p.x - current.start.x);
        const h = Math.abs(p.y - current.start.y);
        current.el.setAttribute('x', x);
        current.el.setAttribute('y', y);
        current.el.setAttribute('width', w);
        current.el.setAttribute('height', h);
      } else if(current.type === 'line'){
        current.el.setAttribute('x2', p.x);
        current.el.setAttribute('y2', p.y);
      } else if(current.type === 'polyline'){
        const last = current.points[current.points.length-1];
        const dx = p.x - last.x; const dy = p.y - last.y;
        if(Math.hypot(dx,dy) > 2){
          current.points.push(p);
          const pts = current.points.map(pt=>`${pt.x},${pt.y}`).join(' ');
          current.el.setAttribute('points', pts);
        }
      }
    }

    function endShape(evt){
      if(!drawing || !current) return;
      drawing = false;
      const el = current.el;
      let acceptable = true;
```

```
  if(current.type === 'rect'){
    const w = parseFloat(el.getAttribute('width')) || 0;
    const h = parseFloat(el.getAttribute('height')) || 0;
    if(w < 2 && h < 2) acceptable = false;
  } else if(current.type === 'line'){
    const x1 = parseFloat(el.getAttribute('x1'));
    const y1 = parseFloat(el.getAttribute('y1'));
    const x2 = parseFloat(el.getAttribute('x2'));
    const y2 = parseFloat(el.getAttribute('y2'));
    if(Math.hypot(x2-x1,y2-y1) < 2) acceptable = false;
  } else if(current.type === 'polyline'){
    if(current.points.length < 2) acceptable = false;
  }
  if(!acceptable){
    svg.removeChild(el);
  } else {
    shapes.push(el);
    addShapeToList(`${current.type} — ${shapes.length}`);
  }
  current = null;
}

svg.addEventListener('mousedown', startShape);
window.addEventListener('mousemove', moveShape);
window.addEventListener('mouseup', endShape);

undoBtn.addEventListener('click', ()=>{
  const last = shapes.pop();
  if(last && svg.contains(last)) svg.removeChild(last);
});

clearBtn.addEventListener('click', ()=>{
  shapes.forEach(s=>{ if(svg.contains(s)) svg.removeChild(s); });
  shapes = [];
  shapeList.innerHTML = '';
});

downloadBtn.addEventListener('click', ()=>{
  const clone = svg.cloneNode(true);
  clone.removeAttribute('id');
  const serializer = new XMLSerializer();
  const source = serializer.serializeToString(clone);
  const blob = new Blob([source], {type: 'image/svg+xml'});
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
```

```
      a.href = url; a.download = 'drawing.svg';
      document.body.appendChild(a); a.click(); a.remove();
      URL.revokeObjectURL(url);
    });
  </script>
</body>
</html>
```

OUTPUT-



```
        addShapeToList( ${current.type} — ${shapes.length} );
      }
      current = null;
    }

    svg.addEventListener('mousedown', startShape);
    window.addEventListener('mousemove', moveShape);
    window.addEventListener('mouseup', endShape);

    undoBtn.addEventListener('click', ()=>{
      const last = shapes.pop();
      if(last && svg.contains(last)) svg.removeChild(last);
    });

    clearBtn.addEventListener('click', ()=>{
      shapes.forEach(s=>{ if(svg.contains(s)) svg.removeChild(s); });
      shapes = [];
      shapeList.innerHTML = '';
    });

    downloadBtn.addEventListener('click', ()=>{
      const clone = svg.cloneNode(true);
      clone.removeAttribute('id');
      const serializer = new XMLSerializer();
      const source = serializer.serializeToString(clone);
      const blob = new Blob([source], {type: 'image/svg+xml'});
      const url = URL.createObjectURL(blob);
      const a = document.createElement('a');
      a.href = url; a.download = 'drawing.svg';
      document.body.appendChild(a); a.click(); a.remove();
      URL.revokeObjectURL(url);
    });
  </script>
</body>
</html>
```

Shape: Freehand (Polyline)
Stroke width: 6

Learning Outcomes

- Demonstrate the use of SVG as a drawing canvas in web development.
- Implement interactive graphics using JavaScript event handling.
- Apply DOM manipulation techniques to dynamically create and update elements.
- Develop user-friendly web applications with real-time interactivity.
- Gain hands-on experience in front-end development concepts useful in UI/UX design and graphical applications.