EXP-2.3

AIM- Interactive SVG Drawing Tool with Mouse Event Handlers

CODE-

```
<!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: #0f1220;
   --panel: #161a2b;
   --ink: #e8ebff;
   --muted: #8e96b8;
   --accent: #6ea8fe;
   --danger: #ff6b6b;
   --ok: #1ac486;
  }
  * { box-sizing: border-box; }
  html, body { height: 100%; }
  body {
   margin: 0;
   font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto,
"Helvetica Neue", Arial, "Apple Color Emoji", "Segoe UI Emoji";
   background: radial-gradient(1200px 800px at 100% -10%, #1f2348, var(--
bg));
   color: var(--ink);
   display: grid;
   grid-template-rows: auto 1fr;
   gap: 12px;
  }
  header {
   display: flex;
```

```
      align-items: center;
      gap: 14px;
      padding: 12px 16px;
      background: linear-gradient(180deg, rgba(255,255,255,0.06),
rgba(255,255,255,0.02));
      border-bottom: 1px solid rgba(255,255,255,0.08);
      backdrop-filter: blur(8px);
      position: sticky;
      top: 0;
      z-index: 10;
    }
    header h1 { font-size: 18px; margin: 0 8px 0 0; letter-spacing: .3px; }
    .toolbar {
      display: flex; flex-wrap: wrap; align-items: center; gap: 10px;
    }
    .toolgroup {
      display: inline-flex; align-items: center; gap: 6px; padding: 8px; border-
radius: 12px; background: var(--panel); border: 1px solid
rgba(255,255,255,0.06);
    }
    .segmented {
      display: inline-flex; border-radius: 10px; overflow: hidden; border: 1px
solid rgba(255,255,255,0.1);
    }
    .segmented button {
      background: transparent; color: var(--ink); border: 0; padding: 8px 10px;
cursor: pointer; font-weight: 600; letter-spacing: .2px;
    }
    .segmented button.active { background: rgba(255,255,255,0.12); }
    label { font-size: 12px; color: var(--muted); }
    input[type="number"], select {
      background: #0e1122; border: 1px solid rgba(255,255,255,0.12); color:
var(--ink); border-radius: 8px; padding: 6px 8px; font-weight: 600;
    }
    input[type="range"] { accent-color: var(--accent); }
    input[type="color"] { width: 36px; height: 28px; border: none; background:
transparent; cursor: pointer; }
    .btn { border: 1px solid rgba(255,255,255,0.14); background: #0e1223;
```

```
color: var(--ink); padding: 8px 12px; border-radius: 10px; cursor: pointer;
font-weight: 700; letter-spacing: .2px; }
   .btn:hover { filter: brightness(1.1); }
   .btn.danger { border-color: rgba(255,107,107,.4); color: #ffd9d9; }

   main { padding: 0 14px 14px; }
   .stage-wrapper { width: 100%; height: calc(100vh - 110px); background:
#0a0d1a; border: 1px solid rgba(255,255,255,0.08); border-radius: 16px;
overflow: hidden; position: relative; }
   .overlay-hint { position: absolute; top: 10px; right: 12px; font-size: 12px;
color: var(--muted); background: rgba(0,0,0,0.35); padding: 6px 8px; border-
radius: 8px; pointer-events: none; }

   svg { width: 100%; height: 100%; display: block; background-image: linear-
gradient(rgba(255,255,255,.04) 1px, transparent 1px), linear-gradient(90deg,
rgba(255,255,255,.04) 1px, transparent 1px);
     background-size: 24px 24px; background-position: center; }
   .ghost { stroke-dasharray: 6 6; opacity: .8; pointer-events: none; }
  </style>
</head>
<body>
  <header>
   <h1>Interactive SVG Drawing Tool</h1>
   <div class="toolbar">
    <div class="toolgroup">
     <label>Tool</label>
     <div class="segmented" id="toolButtons">
      <button data-tool="rectangle" class="active"
title="R">Rectangle</button>
       <button data-tool="line" title="L">Line</button>
       <button data-tool="ellipse" title="E">Ellipse</button>
       <button data-tool="freehand" title="F">Freehand</button>
     </div>
    </div>

    <div class="toolgroup">
     <label>Stroke</label>
     <input type="color" id="strokeColor" value="#6ea8fe" />
```

```
      <label>Width</label>
      <input type="range" id="strokeWidth" min="1" max="20" value="3" />
    </div>

    <div class="toolgroup">
      <label>Fill</label>
      <input type="color" id="fillColor" value="#1ac486" />
      <label>
        <input type="checkbox" id="fillToggle" checked /> enable
      </label>
    </div>

    <div class="toolgroup">
      <button class="btn" id="undoBtn">Undo</button>
      <button class="btn danger" id="clearBtn">Clear</button>
      <select id="exportSelect">
        <option value="">Export…</option>
        <option value="svg">Download SVG</option>
        <option value="png">Download PNG</option>
      </select>
    </div>
  </div>
 </header>

 <main>
  <div class="stage-wrapper">
   <div class="overlay-hint">Tip: Hold <b>Shift</b> to lock to square/circle
or straight line.</div>
   <svg id="stage" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1200
800">
     <!-- Shapes will be appended here -->
   </svg>
  </div>
 </main>

 <script>
  const svg = document.getElementById('stage');
```

```
  const toolButtons = document.getElementById('toolButtons');
  const strokeColor = document.getElementById('strokeColor');
  const strokeWidth = document.getElementById('strokeWidth');
  const fillColor = document.getElementById('fillColor');
  const fillToggle = document.getElementById('fillToggle');
  const undoBtn = document.getElementById('undoBtn');
  const clearBtn = document.getElementById('clearBtn');
  const exportSelect = document.getElementById('exportSelect');

  let tool = 'rectangle';

  // Toolbar events
  toolButtons.addEventListener('click', (e) => {
    const btn = e.target.closest('button[data-tool]');
    if (!btn) return;
    tool = btn.dataset.tool;
    [...toolButtons.children].forEach(b => b.classList.toggle('active', b ===
btn));
  });

  // Utility: convert mouse to SVG coordinates
  function svgPoint(evt) {
    const pt = svg.createSVGPoint();
    pt.x = evt.clientX; pt.y = evt.clientY;
    return pt.matrixTransform(svg.getScreenCTM().inverse());
  }

  // Drawing state
  let isDrawing = false;
  let start = { x: 0, y: 0 };
  let draftEl = null; // element being drawn

  function applyCommonStyles(el) {
    el.setAttribute('stroke', strokeColor.value);
    el.setAttribute('stroke-width', strokeWidth.value);
    el.setAttribute('vector-effect', 'non-scaling-stroke');
    if (el.tagName === 'rect' || el.tagName === 'ellipse' || el.tagName === 'path'
```

```
|| el.tagName === 'polygon' || el.tagName === 'polyline') {
        el.setAttribute('fill', fillToggle.checked ? fillColor.value + '55' : 'none');
      } else {
        el.setAttribute('fill', 'none');
      }
    }

    function startDraw(evt) {
      if (evt.button !== 0) return; // left click only
      isDrawing = true;
      const p = svgPoint(evt);
      start = { x: p.x, y: p.y };

      switch (tool) {
        case 'rectangle': {
          draftEl = document.createElementNS('http://www.w3.org/2000/svg',
'rect');
          draftEl.classList.add('ghost');
          draftEl.setAttribute('x', start.x);
          draftEl.setAttribute('y', start.y);
          draftEl.setAttribute('width', 0);
          draftEl.setAttribute('height', 0);
          applyCommonStyles(draftEl);
          svg.appendChild(draftEl);
          break;
        }
        case 'line': {
          draftEl = document.createElementNS('http://www.w3.org/2000/svg',
'line');
          draftEl.classList.add('ghost');
          draftEl.setAttribute('x1', start.x);
          draftEl.setAttribute('y1', start.y);
          draftEl.setAttribute('x2', start.x);
          draftEl.setAttribute('y2', start.y);
          applyCommonStyles(draftEl);
          svg.appendChild(draftEl);
          break;
```

```
      }
      case 'ellipse': {
        draftEl = document.createElementNS('http://www.w3.org/2000/svg',
'ellipse');
        draftEl.classList.add('ghost');
        draftEl.setAttribute('cx', start.x);
        draftEl.setAttribute('cy', start.y);
        draftEl.setAttribute('rx', 0);
        draftEl.setAttribute('ry', 0);
        applyCommonStyles(draftEl);
        svg.appendChild(draftEl);
        break;
      }
      case 'freehand': {
        draftEl = document.createElementNS('http://www.w3.org/2000/svg',
'polyline');
        draftEl.classList.add('ghost');
        draftEl.setAttribute('points', `${start.x},${start.y}`);
        applyCommonStyles(draftEl);
        svg.appendChild(draftEl);
        break;
      }
    }

    // prevent text selection while dragging
    evt.preventDefault();
  }

  function updateDraw(evt) {
    if (!isDrawing || !draftEl) return;
    const p = svgPoint(evt);
    const dx = p.x - start.x;
    const dy = p.y - start.y;

    if (tool === 'rectangle') {
      let x = Math.min(p.x, start.x);
      let y = Math.min(p.y, start.y);
```

```
      let w = Math.abs(dx);
      let h = Math.abs(dy);
      if (evt.shiftKey) { const s = Math.max(w, h); w = h = s; x = start.x < p.x ?
start.x : start.x - s; y = start.y < p.y ? start.y : start.y - s; }
      draftEl.setAttribute('x', x);
      draftEl.setAttribute('y', y);
      draftEl.setAttribute('width', w);
      draftEl.setAttribute('height', h);
    } else if (tool === 'line') {
      let x2 = p.x, y2 = p.y;
      if (evt.shiftKey) {
        // lock to horizontal/vertical (whichever is closer)
        if (Math.abs(dx) > Math.abs(dy)) { y2 = start.y; } else { x2 = start.x; }
      }
      draftEl.setAttribute('x2', x2);
      draftEl.setAttribute('y2', y2);
    } else if (tool === 'ellipse') {
      let rx = Math.abs(dx) / 2; let ry = Math.abs(dy) / 2;
      if (evt.shiftKey) { const r = Math.max(rx, ry); rx = ry = r; }
      draftEl.setAttribute('cx', (start.x + p.x) / 2);
      draftEl.setAttribute('cy', (start.y + p.y) / 2);
      draftEl.setAttribute('rx', rx);
      draftEl.setAttribute('ry', ry);
    } else if (tool === 'freehand') {
      const pts = draftEl.getAttribute('points');
      draftEl.setAttribute('points', pts + ` ${p.x},${p.y}`);
    }
  }

  function finishDraw(evt) {
    if (!isDrawing || !draftEl) return;
    isDrawing = false;
    draftEl.classList.remove('ghost');

    // If the shape is too small, remove it
    const removeTiny = () => { svg.removeChild(draftEl); };
    if (tool === 'rectangle') {
```

```
    const w = parseFloat(draftEl.getAttribute('width'));
    const h = parseFloat(draftEl.getAttribute('height'));
    if (w < 2 && h < 2) removeTiny();
  } else if (tool === 'line') {
    const x1 = +draftEl.getAttribute('x1');
    const y1 = +draftEl.getAttribute('y1');
    const x2 = +draftEl.getAttribute('x2');
    const y2 = +draftEl.getAttribute('y2');
    if (Math.hypot(x2 - x1, y2 - y1) < 2) removeTiny();
  } else if (tool === 'ellipse') {
    const rx = +draftEl.getAttribute('rx');
    const ry = +draftEl.getAttribute('ry');
    if (rx < 1 && ry < 1) removeTiny();
  } else if (tool === 'freehand') {
    const pts = draftEl.getAttribute('points').trim().split(/\s+/);
    if (pts.length < 3) removeTiny();
  }

  draftEl = null;
}

svg.addEventListener('mousedown', startDraw);
svg.addEventListener('mousemove', updateDraw);
window.addEventListener('mouseup', finishDraw);

// Undo & Clear
undoBtn.addEventListener('click', () => {
  const children = [...svg.children];
  const last = children[children.length - 1];
  if (last) svg.removeChild(last);
});
clearBtn.addEventListener('click', () => {
  while (svg.lastChild) svg.removeChild(svg.lastChild);
});

// Export (SVG/PNG)
exportSelect.addEventListener('change', async (e) => {
```

```
    const choice = e.target.value; e.target.value = '';
    if (!choice) return;
    if (choice === 'svg') {
      const blob = new Blob([svg.outerHTML], { type: 'image/svg+xml' });
      const url = URL.createObjectURL(blob);
      triggerDownload(url, 'drawing.svg');
      URL.revokeObjectURL(url);
    } else if (choice === 'png') {
      const xml = new XMLSerializer().serializeToString(svg);
      const svg64 = btoa(unescape(encodeURIComponent(xml)));
      const img = new Image();
      img.onload = () => {
        const canvas = document.createElement('canvas');
        const vb = svg.viewBox.baseVal;
        canvas.width = vb.width; canvas.height = vb.height;
        const ctx = canvas.getContext('2d');
        ctx.drawImage(img, 0, 0);
        canvas.toBlob((blob) => {
          const url = URL.createObjectURL(blob);
          triggerDownload(url, 'drawing.png');
          URL.revokeObjectURL(url);
        });
      };
      img.src = '' + svg64;
    }
  });

  function triggerDownload(url, filename) {
    const a = document.createElement('a');
    a.href = url; a.download = filename; document.body.appendChild(a);
a.click(); a.remove();
  }

  // Keyboard shortcuts for quick switching
  window.addEventListener('keydown', (e) => {
    if (['INPUT', 'TEXTAREA',
'SELECT'].includes(document.activeElement.tagName)) return;
```

```
    const key = e.key.toLowerCase();
    const map = { r: 'rectangle', l: 'line', e: 'ellipse', f: 'freehand' };
    if (map[key]) {
      tool = map[key];
      [...toolButtons.children].forEach(b => b.classList.toggle('active',
b.dataset.tool === tool));
    }
    if (key === 'z' && (e.ctrlKey || e.metaKey)) { undoBtn.click(); }
    if (key === 'delete' || key === 'backspace') { clearBtn.click(); }
  });
 </script>
</body>
</html>
```

OUTPUT-