Lab 11: Canvas & SVG — Interactive Graphics

- Objectives:

- Use Canvas API and SVG to draw and animate simple graphics; build a simple drawing app.

- Tools/Tech:

- VS Code, Canvas API, optionally a library (paper.js) — optional

- Tasks/Steps:

1. Create a canvas and draw shapes/text/images via JS.
2. Implement basic user drawing (mouse events) and a clear/save as image feature.
3. Create a simple SVG animation (e.g., moving object).

- Deliverables:

- Drawing app and an example SVG animation; code and demo.

HTML CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <title>Lab 11 - Canvas & SVG Interactive Graphics</title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <header>
    <h1>Lab 11 — Canvas & SVG: Interactive Graphics</h1>
    <p>Draw on the canvas, choose color/size, clear or save. Below is a simple SVG animation example.</p>
  </header>

  <main>
    <section class="canvas-section">
      <div class="toolbar">
        <label>
          Color:
```

```
        <input id="colorPicker" type="color" value="#111111" />
      </label>

      <label>
        Brush:
        <input id="brushSize" type="range" min="1" max="40" value="4" />
        <span id="brushSizeLabel">4</span> px
      </label>

      <button id="undoBtn" title="Undo last stroke">Undo</button>
      <button id="clearBtn">Clear</button>
      <button id="saveBtn">Save as PNG</button>
    </div>

    <div class="canvas-wrap">
      <canvas id="drawCanvas"></canvas>
    </div>

    <p class="hint">Tip: Use mouse or touch. Resize your browser — canvas auto-fits while preserving drawing.</p>
  </section>

  <hr />

  <section class="svg-section">
    <h2>SVG Animation Example</h2>
    <!-- Simple SVG with a moving circle (JS-driven) -->
    <svg id="demoSvg" viewBox="0 0 800 150" preserveAspectRatio="xMidYMid meet">
      <rect width="100%" height="100%" fill="#f6f8fa" />
      <g id="movingGroup">
        <circle id="movingCircle" cx="50" cy="75" r="20" fill="#ff6b6b" />
        <text id="labelText" x="50" y="75" dy="6" text-anchor="middle" font-size="10" fill="#fff">SVG</text>
      </g>
    </svg>
    <div class="svg-controls">
      <button id="playSvg">Play</button>
      <button id="pauseSvg">Pause</button>
      <label>
        Speed:
        <input id="svgSpeed" type="range" min="0.5" max="5" step="0.1" value="1" />
        <span id="svgSpeedLabel">1.0x</span>
      </label>
    </div>
  </section>
 </main>

 <footer>
  <small>Lab 11 — Canvas & SVG · Generated code</small>
 </footer>

 <script src="script.js"></script>
</body>
</html>
```

CSS CODE:

```
:root{
  --bg:#fafafa;
  --card:#fff;
  --accent:#007bff;
  --muted:#666;
  --pad:14px;
}

*{box-sizing:border-box}
body{
  font-family: Inter, system-ui, Arial, sans-serif;
  margin:0;
  padding:20px;
  background:var(--bg);
  color:#111;
}

header{
  text-align:center;
  margin-bottom:18px;
}

h1{margin:0 0 6px;font-size:22px}
p{margin:0;color:var(--muted)}

main{max-width:980px;margin:16px auto}

.canvas-section, .svg-section{
  background:var(--card);
  padding:18px;
  border-radius:10px;
  box-shadow:0 6px 18px rgba(15,15,15,0.06);
  margin-bottom:18px;
}

.toolbar{
  display:flex;
  gap:10px;
  align-items:center;
  flex-wrap:wrap;
  margin-bottom:10px;
}
```

```
.toolbar label{
  display:flex;align-items:center;gap:8px;font-size:14px;color:var(--muted)
}

.toolbar input[type="color"]{
  width:36px;height:28px;padding:0;border:0;background:transparent;
  cursor:pointer;
}

.toolbar input[type="range"]{vertical-align:middle}

.toolbar button{
  background:var(--accent);
  color:white;
  border:none;
  padding:8px 12px;
  border-radius:6px;
  cursor:pointer;
}
.toolbar button[title]{font-size:13px}
.toolbar button:active{transform:translateY(1px)}

.canvas-wrap{
  background:#fff;border-radius:8px;padding:8px;display:flex;justify-content:center;align-
items:center;
  border:1px solid #eee;
}

/* Canvas takes available width; maintain aspect */
canvas{
  width:100%;
  height:420px; /* CSS height; actual pixel size set by JS for crispness */
  display:block;
  border-radius:6px;
  touch-action: none; /* prevent page scrolling while drawing */
  background: linear-gradient(180deg,#ffffff,#fbfbfb);
  box-shadow: inset 0 1px 0 rgba(0,0,0,0.02);
}

.hint{font-size:13px;color:var(--muted);margin-top:8px}

.svg-section svg{
  width:100%;
  height:150px;
  border-radius:8px;
  border:1px solid #eee;
  display:block;
```

```
  margin-bottom:8px;
}

.svg-controls{display:flex;gap:10px;align-items:center;flex-wrap:wrap}
.svg-controls button{background:#444;color:#fff;padding:7px 10px;border-
radius:6px;border:0;cursor:pointer}
footer{text-align:center;color:#999;margin-top:12px;font-size:13px}
```

JS CODE:

```
// ===== Canvas Drawing App =====
// Features:
// - Mouse + touch drawing
// - Color & brush size controls
// - Undo (per stroke), Clear, Save as PNG
// - Canvas auto-resizes for device pixel ratio for crisp lines

// DOM elements
const canvas = document.getElementById('drawCanvas');
const ctx = canvas.getContext('2d');
const colorPicker = document.getElementById('colorPicker');
const brushSize = document.getElementById('brushSize');
const brushSizeLabel = document.getElementById('brushSizeLabel');
const clearBtn = document.getElementById('clearBtn');
const saveBtn = document.getElementById('saveBtn');
const undoBtn = document.getElementById('undoBtn');

// Drawing state
let drawing = false;
let currentStroke = null; // {color, size, points: [{x,y}, ...]}
let strokes = []; // array of strokes for undo/redo

// device pixel ratio for high-DPI
function fitCanvasToContainer() {
  // CSS size
  const rect = canvas.getBoundingClientRect();
  const dpr = window.devicePixelRatio || 1;
  canvas.width = Math.round(rect.width * dpr);
  canvas.height = Math.round(rect.height * dpr);
  canvas.style.width = rect.width + 'px';
  canvas.style.height = rect.height + 'px';
  // scale context so drawing coordinates map to CSS pixels
  ctx.setTransform(dpr, 0, 0, dpr, 0, 0);
  // redraw existing strokes after resize
  redrawAll();
```

```
}

// convert client coords to canvas (CSS) coords
function getCanvasPoint(clientX, clientY) {
  const rect = canvas.getBoundingClientRect();
  return {
    x: clientX - rect.left,
    y: clientY - rect.top
  };
}

// Start stroke
function beginStroke(x, y) {
  drawing = true;
  currentStroke = {
    color: colorPicker.value,
    size: parseInt(brushSize.value, 10),
    points: [{ x, y }]
  };
}

// Continue stroke (add point)
function addPointToStroke(x, y) {
  if (!drawing || !currentStroke) return;
  currentStroke.points.push({ x, y });
  // draw just the last segment for performance
  const pts = currentStroke.points;
  const len = pts.length;
  if (len >= 2) {
    drawSegment(pts[len - 2], pts[len - 1], currentStroke.color, currentStroke.size);
  } else {
    drawDot(x, y, currentStroke.color, currentStroke.size);
  }
}

// Finish stroke
function endStroke() {
  if (!drawing) return;
  drawing = false;
  if (currentStroke && currentStroke.points.length > 0) {
    strokes.push(currentStroke);
    currentStroke = null;
  }
}

// Draw utilities
function drawSegment(p1, p2, color, size) {
```

```
  ctx.lineJoin = 'round';
  ctx.lineCap = 'round';
  ctx.strokeStyle = color;
  ctx.lineWidth = size;
  ctx.beginPath();
  ctx.moveTo(p1.x, p1.y);
  ctx.lineTo(p2.x, p2.y);
  ctx.stroke();
}

function drawDot(x, y, color, size) {
  ctx.fillStyle = color;
  ctx.beginPath();
  ctx.arc(x, y, size / 2, 0, Math.PI * 2);
  ctx.fill();
}

// Redraw entire canvas from strokes array
function redrawAll() {
  // clear canvas (use CSS width/height scaled)
  const rect = canvas.getBoundingClientRect();
  ctx.clearRect(0, 0, rect.width, rect.height);

  for (const stroke of strokes) {
    const pts = stroke.points;
    if (pts.length === 1) {
      drawDot(pts[0].x, pts[0].y, stroke.color, stroke.size);
    } else {
      for (let i = 1; i < pts.length; i++) {
        drawSegment(pts[i - 1], pts[i], stroke.color, stroke.size);
      }
    }
  }
}

// Undo last stroke
function undoLast() {
  strokes.pop();
  redrawAll();
}

// Clear canvas entirely
function clearCanvas() {
  strokes = [];
  currentStroke = null;
  redrawAll();
}
```

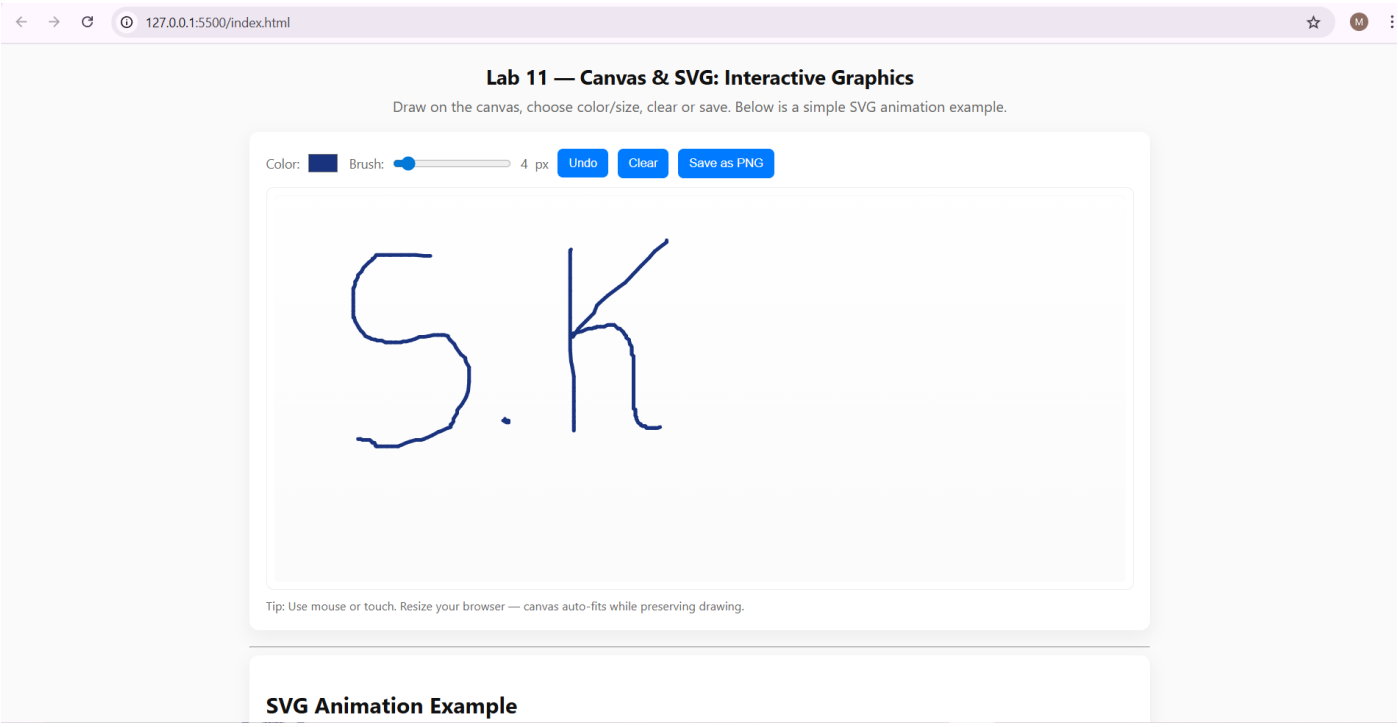
```
// Save as PNG
function saveAsImage() {
  // To save crisp, convert canvas to a blob from actual pixel buffer (canvas.width/height)
  canvas.toBlob(function (blob) {
    if (!blob) {
      alert('Save failed (browser may not support canvas.toBlob).');
      return;
    }
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `drawing_${Date.now()}.png`;
    document.body.appendChild(a);
    a.click();
    a.remove();
    URL.revokeObjectURL(url);
  }, 'image/png');
}

// Event handlers — mouse
canvas.addEventListener('mousedown', (e) => {
  const p = getCanvasPoint(e.clientX, e.clientY);
  beginStroke(p.x, p.y);
});
window.addEventListener('mousemove', (e) => {
  if (!drawing) return;
  const p = getCanvasPoint(e.clientX, e.clientY);
  addPointToStroke(p.x, p.y);
});
window.addEventListener('mouseup', () => endStroke());

// Touch support
canvas.addEventListener('touchstart', (e) => {
  e.preventDefault();
  const touch = e.changedTouches[0];
  const p = getCanvasPoint(touch.clientX, touch.clientY);
  beginStroke(p.x, p.y);
}, { passive: false });

canvas.addEventListener('touchmove', (e) => {
  e.preventDefault();
  const touch = e.changedTouches[0];
  const p = getCanvasPoint(touch.clientX, touch.clientY);
  addPointToStroke(p.x, p.y);
}, { passive: false });
```

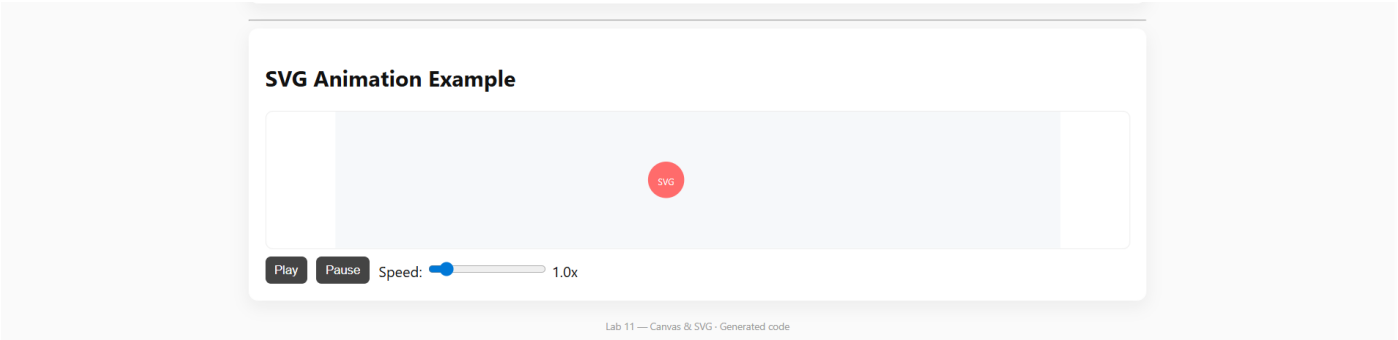
```
canvas.addEventListener('touchend', (e) => {
  e.preventDefault();
  endStroke();
}, { passive: false });

// controls
brushSize.addEventListener('input', () => {
  brushSizeLabel.textContent = brushSize.value;
});
clearBtn.addEventListener('click', () => {
  // simple confirm to avoid accidental clears
  if (confirm('Clear the canvas? This cannot be undone (you can use Undo for previous
strokes).')) {
    clearCanvas();
  }
});
saveBtn.addEventListener('click', saveAsImage);
undoBtn.addEventListener('click', undoLast);

// Handle window resize to keep canvas crisp
let resizeTimeout = null;
function handleResize() {
  // debounce
  clearTimeout(resizeTimeout);
  resizeTimeout = setTimeout(() => {
    fitCanvasToContainer();
  }, 120);
}
window.addEventListener('resize', handleResize);

// Initial setup: set canvas CSS height (already set in CSS), then fit to container
window.addEventListener('load', () => {
  fitCanvasToContainer();
});

// ===== Simple SVG Animation (JS-driven) =====
// We'll animate the movingGroup element horizontally and bounce left-right.
// Controls: play/pause and speed.

const svg = document.getElementById('demoSvg');
const movingGroup = document.getElementById('movingGroup');
const playSvg = document.getElementById('playSvg');
const pauseSvg = document.getElementById('pauseSvg');
const svgSpeed = document.getElementById('svgSpeed');
const svgSpeedLabel = document.getElementById('svgSpeedLabel');

let svgAnimReq = null;
```

```
let svgX = 50; // starting x
let svgDir = 1; // 1 -> right, -1 -> left
let svgMin = 30;
let svgMax = 770; // viewBox width 800 - keep margin
let svgVelocityBase = 100; // pixels per second at speed=1
let svgPlaying = true;

function updateSvgSpeedLabel() {
  svgSpeedLabel.textContent = parseFloat(svgSpeed.value).toFixed(1) + 'x';
}
svgSpeed.addEventListener('input', updateSvgSpeedLabel);

// animation loop using requestAnimationFrame for smooth motion
let lastTimestamp = null;
function svgAnimate(ts) {
  if (!lastTimestamp) lastTimestamp = ts;
  const dt = (ts - lastTimestamp) / 1000; // seconds
  lastTimestamp = ts;

  const speedFactor = parseFloat(svgSpeed.value);
  const velocity = svgVelocityBase * speedFactor; // px / sec
  svgX += svgDir * velocity * dt;

  // bounce
  if (svgX >= svgMax) {
    svgX = svgMax;
    svgDir = -1;
  } else if (svgX <= svgMin) {
    svgX = svgMin;
    svgDir = 1;
  }

  movingGroup.setAttribute('transform', `translate(${svgX - 50}, 0)`); // group initially at
x=50

  if (svgPlaying) svgAnimReq = requestAnimationFrame(svgAnimate);
}

function startSvg() {
  if (svgPlaying) return;
  svgPlaying = true;
  lastTimestamp = null;
  svgAnimReq = requestAnimationFrame(svgAnimate);
}

function pauseSvgAnim() {
  svgPlaying = false;
```

```
  if (svgAnimReq) {
    cancelAnimationFrame(svgAnimReq);
    svgAnimReq = null;
  }
}

playSvg.addEventListener('click', startSvg);
pauseSvg.addEventListener('click', pauseSvgAnim);

// Start animation automatically
updateSvgSpeedLabel();
svgAnimReq = requestAnimationFrame(svgAnimate);
```

OUTPUT SCREEN





CONCLUSION:

I have Completed the task of making Design Editor.Good To Learn.

RUBRICS:

Performance			Lab Report		
Description	**Total Marks**	**Marks Obtained**	**Description**	**Total Marks**	**Marks Obtained**
Ability to Conduct practical	5		**Structure**	5	
Data Analysis & Interpretation	5		**Efficiency**	5	
Total Marks obtained			**Total Marks Obtained**		

Instructor Signature _____