Editor.php

<?php

session\_start();

if (!isset($\_SESSION['user\_id'])) {

    header('Location: index.php');

    exit();

}

$userName = isset($\_SESSION['user\_name']) ? $\_SESSION['user\_name'] : 'User';

$userId = $\_SESSION['user\_id'];

$baseStorageDir = './uploads/';

$uploadDir = $baseStorageDir . $userId . '/';

$imageName = isset($\_GET['image']) ? basename(urldecode($\_GET['image'])) : null;

if (!$imageName) {

    header('Location: drive.php');

    exit();

}

$imagePath = $uploadDir . 'images/' . $imageName;

if (!file\_exists($imagePath)) {

    header('Location: drive.php');

    exit();

}

?>

  <!DOCTYPE html>

  <html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <title>Advanced Editor | CloudDrive</title>

    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap" rel="stylesheet" />

    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet" />

    <style>

      /\* Reset and base \*/

      \*, \*::before, \*::after {

        box-sizing: border-box;

      }

      body {

        margin: 0;

        font-family: 'Inter', sans-serif;

        background: #ffffff;

        color: #333333;

        min-height: 100vh;

        display: flex;

        flex-direction: column;

        align-items: center;

        padding: 20px;

        -webkit-font-smoothing: antialiased;

        -moz-osx-font-smoothing: grayscale;

      }

      h1, h2 {

        margin-bottom: 0.25em;

        font-weight: 700;

        text-shadow: none;

        color: #222;

      }

      h1 {

        font-size: 2.5rem;

      }

      h2 {

        font-size: 1.25rem;

        font-weight: 600;

        color: #555;

        margin-bottom: 1rem;

      }

      /\* Container for editor \*/

      .editor-container {

        background: #f9fbff;

        border-radius: 16px;

        padding: 24px;

        max-width: 960px;

        width: 100%;

        box-shadow: 0 8px 24px rgba(100, 120, 160, 0.15);

        display: flex;

        flex-direction: column;

        align-items: center;

        gap: 24px;

      }

      /\* Toolbar styling \*/

      .toolbar {

        display: flex;

        flex-wrap: wrap;

        justify-content: center;

        gap: 16px;

        background: #e6f0ff;

        padding: 16px 20px;

        border-radius: 12px;

        box-shadow: 0 0 15px rgba(30, 100, 230, 0.2);

        user-select: none;

      }

      .toolbar-group {

        display: flex;

        gap: 12px;

        align-items: center;

      }

      /\* Button styles \*/

      button, select, input[type="range"], input[type="color"] {

        font-family: inherit;

        font-weight: 600;

        font-size: 0.9rem;

        border-radius: 12px;

        border: none;

        padding: 10px 18px;

        background: #87ceeb; /\* sky blue \*/

        color: #003366;

        cursor: pointer;

        box-shadow: 0 3px 8px rgba(135, 206, 235, 0.45);

        transition: background-color 0.3s ease, transform 0.2s ease;

        min-width: 75px;

        text-align: center;

        user-select:none;

        display: flex;

        align-items: center;

        justify-content: center;

        gap: 6px;

      }

      button:hover, select:hover, input[type="color"]:hover {

        background-color: #5dade2;

        transform: scale(1.05);

      }

      button:active, select:active {

        transform: scale(0.97);

      }

      button:focus, select:focus, input[type="color"]:focus, input[type="range"]:focus {

        outline: 2px solid #3399ff;

        outline-offset: 2px;

      }

      button:disabled {

        background-color: #aaccea;

        cursor: not-allowed;

        box-shadow: none;

        transform: none;

        color: #666666;

      }

      /\* Color picker and range slider \*/

      input[type="color"] {

        padding: 0;

        width: 40px;

        height: 40px;

        border-radius: 50%;

        border: 2px solid #87ceeb;

        cursor: pointer;

      }

      input[type="range"] {

        -webkit-appearance: none;

        appearance: none;

        width: 110px;

        height: 8px;

        border-radius: 8px;

        background: #b3d9ff;

        cursor: pointer;

        margin-left: 8px;

        margin-right: 8px;

        vertical-align: middle;

      }

      input[type="range"]::-webkit-slider-thumb {

        -webkit-appearance: none;

        appearance: none;

        width: 20px;

        height: 20px;

        border-radius: 50%;

        background: #3399ff;

        cursor: pointer;

        box-shadow: 0 0 8px rgba(51, 153, 255, 0.8);

        border: 1px solid #187bcd;

        transition: background-color 0.3s ease;

        margin-top: -6px;

      }

      input[type="range"]:focus::-webkit-slider-thumb {

        background-color: #66b3ff;

        box-shadow: 0 0 12px rgba(102, 179, 255, 1);

      }

      input[type="range"]::-moz-range-thumb {

        width: 20px;

        height: 20px;

        border-radius: 50%;

        background: #3399ff;

        cursor: pointer;

        border: 1px solid #187bcd;

        transition: background-color 0.3s ease;

        box-shadow: 0 0 8px rgba(51, 153, 255, 0.8);

      }

      input[type="range"]:focus::-moz-range-thumb {

        background-color: #66b3ff;

        box-shadow: 0 0 12px rgba(102, 179, 255, 1);

      }

      /\* Canvas container \*/

      #canvas-container {

        border-radius: 16px;

        background: white;

        box-shadow: 0 12px 30px rgba(51, 153, 255, 0.15);

        max-width: 100%;

        overflow: auto;

        cursor: crosshair;

        user-select:none;

      }

      #canvas {

        display: block;

        max-width: 100%;

        border-radius: 16px;

        background-color: #fefefe;

        box-shadow: inset 0 0 12px rgba(0,0,0,0.05);

        image-rendering: pixelated;

      }

      /\* Responsive \*/

      @media (max-width: 720px) {

        .toolbar {

          gap: 10px;

          padding: 12px 10px;

        }

        button, select {

          min-width: 60px;

          padding: 8px 12px;

          font-size: 0.85rem;

        }

        input[type="range"] {

          width: 90px;

        }

      /\* Material icons adjustment \*/

      .material-icons {

        font-size: 18px;

        vertical-align: middle;

        pointer-events: none;

      }

      /\* Material icons adjustment \*/

      .material-icons {

        font-size: 18px;

        vertical-align: middle;

        pointer-events: none;

      }

    </style>

  </head>

  <body>

  <h1>Welcome, <?php echo htmlspecialchars($userName); ?>!</h1>

  <h2>Editing: <?php echo htmlspecialchars($imageName); ?></h2>

  <div class="editor-container">

    <div id="canvas-container" aria-live="polite" aria-label="Image editing canvas">

      <canvas id="canvas" tabindex="0" role="img" aria-describedby="canvasDesc"></canvas>

      <div id="canvasDesc" style="position:absolute; left:-9999px; top:auto; width:1px; height:1px; overflow:hidden;">Canvas showing the image currently being edited.</div>

    </div>

        <div class="toolbar" role="toolbar" aria-label="Image editing tools">

      <div class="toolbar-group" aria-label="Drawing controls">

        <input type="color" id="drawColorPicker" title="Select drawing color" aria-label="Drawing color picker" value="#000000" />

        <label for="drawSizeSlider" style="color:#336699; font-weight:600; margin-left:4px;">Size</label>

        <input type="range" id="drawSizeSlider" min="1" max="50" value="5" aria-label="Drawing size slider" />

        <button id="drawBtn" aria-pressed="false" title="Toggle draw mode" aria-label="Toggle draw mode">

          <span class="material-icons" aria-hidden="true">brush</span> Draw

        </button>

      </div>

      <div class="toolbar-group" aria-label="Basic transformations">

        <button id="cropBtn" title="Crop image" aria-label="Crop image">

          <span class="material-icons" aria-hidden="true">crop</span> Crop

        </button>

        <button id="rotateBtn" title="Rotate 90 degrees" aria-label="Rotate image">

          <span class="material-icons" aria-hidden="true">rotate\_right</span> Rotate

        </button>

        <button id="flipHBtn" title="Flip Horizontally" aria-label="Flip image horizontally">

          <span class="material-icons" aria-hidden="true">flip</span> Flip H

        </button>

        <button id="flipVBtn" title="Flip Vertically" aria-label="Flip image vertically">

          <span class="material-icons" aria-hidden="true">flip\_camera\_android</span> Flip V

        </button>

      </div>

      <div class="toolbar-group" aria-label="Filters">

        <select id="filterSelect" aria-label="Select filter to apply">

          <option value="normal">Normal</option>

          <option value="grayscale">Grayscale</option>

          <option value="sepia">Sepia</option>

          <option value="invert">Invert</option>

          <option value="brightness">Brightness +20%</option>

          <option value="contrast">Contrast +20%</option>

        </select>

      </div>

      <div class="toolbar-group" aria-label="Undo and saving">

        <button id="undoBtn" title="Undo last action" aria-label="Undo last action">

          <span class="material-icons" aria-hidden="true">undo</span> Undo

        </button>

        <button id="saveBtn" title="Save edited image" aria-label="Save edited image">

          <span class="material-icons" aria-hidden="true">save</span> Save

        </button>

        <button id="backBtn" title="Return to drive" aria-label="Back to drive" onclick="window.location.href='drive.php'">

          <span class="material-icons" aria-hidden="true">arrow\_back</span> Back

        </button>

      </div>

    </div>

  </div>

  <script>

  window.currentImage = '<?php echo htmlspecialchars($imageName); ?>';

  window.currentUser  = '<?php echo htmlspecialchars($userName); ?>';

  const canvas = document.getElementById('canvas');

  const ctx = canvas.getContext('2d');

  const saveBtn = document.getElementById('saveBtn');

  const drawBtn = document.getElementById('drawBtn');

  const cropBtn = document.getElementById('cropBtn');

  const rotateBtn = document.getElementById('rotateBtn');

  const flipHBtn = document.getElementById('flipHBtn');

  const flipVBtn = document.getElementById('flipVBtn');

  const filterSelect = document.getElementById('filterSelect');

  const undoBtn = document.getElementById('undoBtn');

  const drawColorPicker = document.getElementById('drawColorPicker');

  const drawSizeSlider = document.getElementById('drawSizeSlider');

  const backBtn = document.getElementById('backBtn');

  let drawing = false, isDrawingMode = false;

  let lastX = 0, lastY = 0;

  // Store a copy of the original image for erasing only the drawing

  let baseImageData = null;

async function init() {

  const imageName = window.currentImage;

  const userId = '<?php echo $userId; ?>';

  const imagePath = `uploads/${userId}/images/${encodeURIComponent(imageName)}`;

  const img = new Image();

  img.crossOrigin = "anonymous";

  img.src = imagePath;

  await img.decode();

  canvas.width = img.width;

  canvas.height = img.height;

  ctx.drawImage(img, 0, 0);

  originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

  saveState();

  updateDrawButton();

}

  // Erase only the drawing (not the image)

  function eraseAt(x, y, size) {

    // Restore base image to a temp canvas

    const tempCanvas = document.createElement('canvas');

    tempCanvas.width = canvas.width;

    tempCanvas.height = canvas.height;

    const tempCtx = tempCanvas.getContext('2d');

    tempCtx.putImageData(baseImageData, 0, 0);

    // Copy current canvas to temp, but mask erase area with base image

    tempCtx.save();

    tempCtx.beginPath();

    tempCtx.arc(x, y, size / 2, 0, 2 \* Math.PI);

    tempCtx.clip();

    tempCtx.drawImage(canvas, 0, 0);

    tempCtx.restore();

    // Draw temp canvas back to main canvas, only in erase area

    ctx.save();

    ctx.beginPath();

    ctx.arc(x, y, size / 2, 0, 2 \* Math.PI);

    ctx.clip();

    ctx.drawImage(tempCanvas, 0, 0);

    ctx.restore();

  }

  // Override draw function to erase only drawing

  function draw(e) {

    if (!drawing) return;

    const rect = canvas.getBoundingClientRect();

    const x = e.clientX - rect.left;

    const y = e.clientY - rect.top;

    ctx.save();

    if (isErasingMode) {

      eraseAt(x, y, drawSizeSlider.value);

    } else if (isDrawingMode) {

      ctx.fillStyle = drawColorPicker.value;

      ctx.beginPath();

      ctx.arc(x, y, drawSizeSlider.value / 2, 0, 2 \* Math.PI);

      ctx.fill();

    }

    ctx.restore();

    lastX = x;

    lastY = y;

  }

  // When saving, update baseImageData so erase works after save

  saveBtn.addEventListener('click', async () => {

    try {

      const dataUrl = canvas.toDataURL('image/png');

      const blob = await (await fetch(dataUrl)).blob();

      const formData = new FormData();

      formData.append('edited\_image', blob, window.currentImage);

      formData.append('original\_name', window.currentImage);

      const saveResponse = await fetch('save\_image.php', { method: 'POST', body: formData });

      const result = await saveResponse.json();

      if (result.success) {

        alert('Image saved successfully!');

        // Update baseImageData to current image after save

        baseImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

        window.location.href = 'drive.php';

      } else {

        alert('Failed to save image.');

      }

    } catch (error) {

      alert('An error occurred while saving the image.');

      console.error(error);

    }

    // No erase tool, nothing to do here.

  });

  // Erase tool state

  canvas.addEventListener('mousedown', (e) => {

    const rect = canvas.getBoundingClientRect();

    if (cropMode) {

      cropStartX = e.clientX - rect.left;

      cropStartY = e.clientY - rect.top;

    } else if (isDrawingMode || isErasingMode) {

      drawing = true;

      const x = e.clientX - rect.left;

      const y = e.clientY - rect.top;

      lastX = x;

      lastY = y;

      draw(e);

    }

  });

  canvas.addEventListener('mousemove', (e) => {

    if (!drawing) return;

    const rect = canvas.getBoundingClientRect();

    const x = e.clientX - rect.left;

    const y = e.clientY - rect.top;

    ctx.save();

    if (isErasingMode) {

      ctx.globalCompositeOperation = 'destination-out';

      ctx.lineCap = 'round';

      ctx.lineJoin = 'round';

      ctx.lineWidth = drawSizeSlider.value;

      ctx.beginPath();

      ctx.moveTo(lastX, lastY);

      ctx.lineTo(x, y);

      ctx.stroke();

      ctx.globalCompositeOperation = 'source-over';

    } else if (isDrawingMode) {

      ctx.strokeStyle = drawColorPicker.value;

      ctx.lineCap = 'round';

      ctx.lineJoin = 'round';

      ctx.lineWidth = drawSizeSlider.value;

      ctx.beginPath();

      ctx.moveTo(lastX, lastY);

      ctx.lineTo(x, y);

      ctx.stroke();

    }

    ctx.restore();

    lastX = x;

    lastY = y;

  });

  canvas.addEventListener('mouseup', (e) => {

    if (cropMode) {

      const rect = canvas.getBoundingClientRect();

      const endX = e.clientX - rect.left;

      const endY = e.clientY - rect.top;

      const cropWidth = endX - cropStartX;

      const cropHeight = endY - cropStartY;

      const safeCropWidth = Math.min(cropWidth, canvas.width - cropStartX);

      const safeCropHeight = Math.min(cropHeight, canvas.height - cropStartY);

      if(safeCropWidth > 0 && safeCropHeight > 0){

        const imageData = ctx.getImageData(cropStartX, cropStartY, safeCropWidth, safeCropHeight);

        canvas.width = safeCropWidth;

        canvas.height = safeCropHeight;

        ctx.putImageData(imageData, 0, 0);

        originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

        saveState();

      }

      cropMode = false;

    }

    if (drawing && (isDrawingMode || isErasingMode)) {

      saveState();

    }

    drawing = false;

  });

  let isErasingMode = false;

  const eraseBtn = document.createElement('button');

  eraseBtn.id = 'eraseBtn';

  eraseBtn.title = 'Toggle erase mode';

  eraseBtn.setAttribute('aria-label', 'Toggle erase mode');

  eraseBtn.innerHTML = '<span class="material-icons" aria-hidden="true">auto\_fix\_off</span> Erase';

  // Insert erase button after drawBtn in the toolbar

  drawBtn.parentNode.insertBefore(eraseBtn, drawBtn.nextSibling);

  function updateEraseButton() {

    eraseBtn.setAttribute('aria-pressed', isErasingMode ? 'true' : 'false');

    eraseBtn.style.backgroundColor = isErasingMode ? '#5dade2' : '#87ceeb';

  }

  eraseBtn.addEventListener('click', () => {

      isErasingMode = !isErasingMode;

      isDrawingMode = false;

      cropMode = false;

      updateEraseButton();

      updateDrawButton();

  });

  // Update drawBtn to disable erase mode when drawing mode is toggled

  drawBtn.addEventListener('click', () => {

      isErasingMode = false;

      updateEraseButton();

  });

  // Update cropBtn to disable erase mode when crop mode is toggled

  cropBtn.addEventListener('click', () => {

      isErasingMode = false;

      updateEraseButton();

  });

  // Modify draw function to support erase mode

  function draw(e) {

      if (!drawing) return;

      const rect = canvas.getBoundingClientRect();

      const x = e.clientX - rect.left;

      const y = e.clientY - rect.top;

      ctx.save();

      if (isErasingMode) {

          eraseAt(x, y, drawSizeSlider.value);

      } else if (isDrawingMode) {

          ctx.fillStyle = drawColorPicker.value;

          ctx.beginPath();

          ctx.arc(x, y, drawSizeSlider.value / 2, 0, 2 \* Math.PI);

          ctx.fill();

      }

      ctx.restore();

  }

  let cropMode = false, cropStartX, cropStartY;

  let undoStack = [];

  let originalImageData = null;

  async function init() {

      const imageName = window.currentImage;

      const userId = '<?php echo $userId; ?>';

      const imagePath = `uploads/${userId}/images/${encodeURIComponent(imageName)}`;

      const img = new Image();

      img.crossOrigin = "anonymous";

      img.src = imagePath;

      await img.decode();

      canvas.width = img.width;

      canvas.height = img.height;

      ctx.drawImage(img, 0, 0);

      originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

      saveState();

      updateDrawButton();

  }

  function updateDrawButton() {

    drawBtn.setAttribute('aria-pressed', isDrawingMode ? 'true' : 'false');

    drawBtn.style.backgroundColor = isDrawingMode ? '#5dade2' : '#87ceeb';

  }

  drawBtn.addEventListener('click', () => {

      isDrawingMode = !isDrawingMode;

      cropMode = false;

      updateDrawButton();

  });

  canvas.addEventListener('mousedown', (e) => {

      const rect = canvas.getBoundingClientRect();

      if (cropMode) {

          cropStartX = e.clientX - rect.left;

          cropStartY = e.clientY - rect.top;

      } else if (isDrawingMode) {

          drawing = true;

          draw(e);

      }

  });

  canvas.addEventListener('mouseup', (e) => {

      if (cropMode) {

          const rect = canvas.getBoundingClientRect();

          const endX = e.clientX - rect.left;

          const endY = e.clientY - rect.top;

          const cropWidth = endX - cropStartX;

          const cropHeight = endY - cropStartY;

          const safeCropWidth = Math.min(cropWidth, canvas.width - cropStartX);

          const safeCropHeight = Math.min(cropHeight, canvas.height - cropStartY);

          if(safeCropWidth > 0 && safeCropHeight > 0){

            const imageData = ctx.getImageData(cropStartX, cropStartY, safeCropWidth, safeCropHeight);

            canvas.width = safeCropWidth;

            canvas.height = safeCropHeight;

            ctx.putImageData(imageData, 0, 0);

            originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

            saveState();

          }

          cropMode = false;

      }

      drawing = false;

  });

  canvas.addEventListener('mousemove', draw);

  function draw(e) {

      if (!drawing) return;

      const rect = canvas.getBoundingClientRect();

      const x = e.clientX - rect.left;

      const y = e.clientY - rect.top;

      ctx.fillStyle = drawColorPicker.value;

      ctx.beginPath();

      ctx.arc(x, y, drawSizeSlider.value / 2, 0, 2 \* Math.PI);

      ctx.fill();

  }

  cropBtn.addEventListener('click', () => {

      cropMode = true;

      isDrawingMode = false;

      updateDrawButton();

      alert('Click and drag on canvas to select crop area.');

  });

  rotateBtn.addEventListener('click', () => {

      const tmpCanvas = document.createElement('canvas');

      const tmpCtx = tmpCanvas.getContext('2d');

      tmpCanvas.width = canvas.height;

      tmpCanvas.height = canvas.width;

      tmpCtx.translate(tmpCanvas.width / 2, tmpCanvas.height / 2);

      tmpCtx.rotate(90 \* Math.PI / 180);

      tmpCtx.drawImage(canvas, -canvas.width / 2, -canvas.height / 2);

      canvas.width = tmpCanvas.width;

      canvas.height = tmpCanvas.height;

      ctx.drawImage(tmpCanvas, 0, 0);

      originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

      saveState();

  });

  flipHBtn.addEventListener('click', () => {

      ctx.translate(canvas.width, 0);

      ctx.scale(-1, 1);

      ctx.drawImage(canvas, 0, 0);

      ctx.setTransform(1, 0, 0, 1, 0, 0);

      originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

      saveState();

  });

  flipVBtn.addEventListener('click', () => {

      ctx.translate(0, canvas.height);

      ctx.scale(1, -1);

      ctx.drawImage(canvas, 0, 0);

      ctx.setTransform(1, 0, 0, 1, 0, 0);

      originalImageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

      saveState();

  });

  filterSelect.addEventListener('change', () => {

      const filter = filterSelect.value;

      if (filter === 'normal') {

          ctx.putImageData(originalImageData, 0, 0);

          saveState();

          return;

      }

      const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

      const data = imageData.data;

      if (filter === 'grayscale') {

          for (let i = 0; i < data.length; i += 4) {

              const avg = (data[i] + data[i + 1] + data[i + 2]) / 3;

              data[i] = data[i + 1] = data[i + 2] = avg;

          }

      }

      else if (filter === 'sepia') {

          for (let i = 0; i < data.length; i += 4) {

              const r = data[i], g = data[i + 1], b = data[i + 2];

              data[i] = r \* 0.393 + g \* 0.769 + b \* 0.189;

              data[i + 1] = r \* 0.349 + g \* 0.686 + b \* 0.168;

              data[i + 2] = r \* 0.272 + g \* 0.534 + b \* 0.131;

          }

      }

      else if (filter === 'invert') {

          for (let i = 0; i < data.length; i += 4) {

              data[i] = 255 - data[i];

              data[i + 1] = 255 - data[i + 1];

              data[i + 2] = 255 - data[i + 2];

          }

      }

      else if (filter === 'brightness') {

          for (let i = 0; i < data.length; i += 4) {

              data[i] = Math.min(255, data[i] \* 1.2);

              data[i + 1] = Math.min(255, data[i + 1] \* 1.2);

              data[i + 2] = Math.min(255, data[i + 2] \* 1.2);

          }

      }

      else if (filter === 'contrast') {

          const contrast = 1.2;

          const intercept = 128 \* (1 - contrast);

          for (let i = 0; i < data.length; i += 4) {

              data[i] = Math.min(255, data[i] \* contrast + intercept);

              data[i + 1] = Math.min(255, data[i + 1] \* contrast + intercept);

              data[i + 2] = Math.min(255, data[i + 2] \* contrast + intercept);

          }

      }

      ctx.putImageData(imageData, 0, 0);

      saveState();

  });

  undoBtn.addEventListener('click', () => {

      if (undoStack.length > 1) {

          undoStack.pop();

          const prevImage = undoStack[undoStack.length - 1];

          const img = new Image();

          img.onload = () => ctx.drawImage(img, 0, 0, canvas.width, canvas.height);

          img.src = prevImage;

      } else {

          alert('No more undo steps available.');

      }

  });

  saveBtn.addEventListener('click', async () => {

      try {

          const dataUrl = canvas.toDataURL('image/png');

          const blob = await (await fetch(dataUrl)).blob();

          const formData = new FormData();

          formData.append('edited\_image', blob, window.currentImage);

          formData.append('original\_name', window.currentImage);

          const saveResponse = await fetch('save\_image.php', { method: 'POST', body: formData });

          const result = await saveResponse.json();

          if (result.success) {

              alert('Image saved successfully!');

              window.location.href = 'drive.php';

          } else {

              alert('Failed to save image.');

          }

      } catch (error) {

          alert('An error occurred while saving the image.');

          console.error(error);

      }

  });

  function saveState() {

      undoStack.push(canvas.toDataURL());

  }

  window.addEventListener('load', init);

  </script>

  </body>

  </html>

Saved\_image.php

<?php

session\_start();

header('Content-Type: application/json');

if (!isset($\_SESSION['user\_id'])) {

    echo json\_encode(['success' => false, 'message' => 'Not logged in']);

    exit();

}

$userId = $\_SESSION['user\_id'];

if (!isset($\_FILES['edited\_image']) || $\_FILES['edited\_image']['error'] !== UPLOAD\_ERR\_OK || !isset($\_POST['original\_name'])) {

    echo json\_encode(['success' => false, 'message' => 'Missing image or file name']);

    exit();

}

$filename = basename($\_POST['original\_name']);

$fileType = strtolower(pathinfo($filename, PATHINFO\_EXTENSION));

$saveFolder = './uploads/' . $userId . '/images/';

if (!is\_dir($saveFolder)) {

    mkdir($saveFolder, 0755, true);

}

$savePath = $saveFolder . $filename;

if (move\_uploaded\_file($\_FILES['edited\_image']['tmp\_name'], $savePath)) {

    chmod($savePath, 0644);

    try {

        $pdo = new PDO("mysql:host=localhost;dbname=user\_auth", "root", "");

        $pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

        // Update or insert

        $stmt = $pdo->prepare("UPDATE uploaded\_images SET file\_path = ?, uploaded\_at = NOW() WHERE file\_name = ? AND user\_id = ?");

        $stmt->execute([$savePath, $filename, $userId]);

        if ($stmt->rowCount() === 0) {

            $stmt = $pdo->prepare("INSERT INTO uploaded\_images (user\_id, file\_name, file\_path, file\_type, uploaded\_at) VALUES (?, ?, ?, ?, NOW())");

            $stmt->execute([$userId, $filename, $savePath, $fileType]);

        }

    } catch (Exception $e) {

        echo json\_encode(['success' => false, 'message' => 'Database error: ' . $e->getMessage()]);

        exit();

    }

    echo json\_encode(['success' => true]);

    exit();

} else {

    echo json\_encode(['success' => false, 'message' => 'Failed to move uploaded image']);

    exit();

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Saved Images</title>

  <style>

    /\* Base and body styles \*/

    body {

      font-family: 'Inter', Arial, sans-serif;

      background-color: #ffffff; /\* White background \*/

      color: #003366; /\* Dark blue text \*/

      text-align: center;

      margin: 40px 20px;

      padding: 0;

      line-height: 1.6;

    }

    h1 {

      font-weight: 700;

      font-size: 2.2rem;

      margin-bottom: 40px;

      letter-spacing: 0.03em;

    }

    /\* Gallery grid styles \*/

    .gallery {

      display: grid;

      grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));

      gap: 32px; /\* generous gap \*/

      max-width: 1200px;

      margin: 0 auto 48px auto;

      padding: 0 16px;

    }

    /\* Each image container \*/

    .gallery > div {

      display: flex;

      flex-direction: column;

      align-items: center;

      background: #f8fbff;

      border-radius: 12px;

      padding: 16px;

      box-shadow: 0 4px 16px rgba(3, 27, 68, 0.1);

      transition: box-shadow 0.3s ease;

    }

    .gallery > div:hover {

      box-shadow: 0 8px 28px rgba(3, 27, 68, 0.2);

    }

    /\* Images styling \*/

    img {

      width: 100%;

      height: auto;

      border-radius: 10px;

      border: 1px solid #cbd6f0;

      object-fit: cover;

      margin-bottom: 16px;

    }

    /\* Form and buttons styling \*/

    form {

      display: flex;

      justify-content: center;

      gap: 12px;

      flex-wrap: wrap;

    }

    button {

      background-color: #87ceeb; /\* sky blue \*/

      color: #ffffff;

      border: none;

      padding: 10px 20px;

      border-radius: 8px;

      font-size: 1rem;

      font-weight: 600;

      cursor: pointer;

      transition: background-color 0.3s ease, transform 0.2s ease;

      min-width: 90px;

      user-select: none;

      box-shadow: 0 2px 8px rgba(135, 206, 235, 0.5);

    }

    button:hover,

    button:focus {

      background-color: #5aa7db;

      outline: none;

      transform: translateY(-2px);

      box-shadow: 0 4px 14px rgba(90, 167, 219, 0.7);

    }

    button:active {

      transform: translateY(0);

      box-shadow: 0 2px 8px rgba(135, 206, 235, 0.5);

    }

    /\* Back to drive button styles \*/

    body > button {

      background-color: #87ceeb;

      color: white;

      border: none;

      padding: 12px 32px;

      border-radius: 10px;

      font-size: 1.1rem;

      font-weight: 700;

      cursor: pointer;

      transition: background-color 0.3s ease, transform 0.2s ease;

      user-select: none;

      box-shadow: 0 4px 16px rgba(135, 206, 235, 0.6);

      margin-top: 32px;

      max-width: 240px;

    }

    body > button:hover,

    body > button:focus {

      background-color: #5aa7db;

      outline: none;

      transform: translateY(-2px);

      box-shadow: 0 8px 28px rgba(90, 167, 219, 0.8);

    }

    body > button:active {

      transform: translateY(0);

      box-shadow: 0 4px 16px rgba(135, 206, 235, 0.6);

    }

    /\* Responsive adjustments \*/

    @media (max-width: 480px) {

      .gallery {

        grid-template-columns: 1fr;

        gap: 24px;

      }

      form {

        flex-direction: column;

        gap: 10px;

      }

      button {

        min-width: 100%;

        padding: 14px;

        font-size: 1.1rem;

      }

      body > button {

        max-width: 100%;

        padding: 14px;

        font-size: 1.2rem;

      }

    }

  </style>

  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap" rel="stylesheet">

</head>

<body>

  <h1>Saved Images for <?php echo htmlspecialchars($username); ?></h1>

  <div class="gallery">

  <?php foreach ($images as $img): ?>

    <div>

      <img src="<?php echo htmlspecialchars($uploadDir . $img); ?>" alt="<?php echo htmlspecialchars($img); ?>">

      <form method="POST" aria-label="Actions for image <?php echo htmlspecialchars($img); ?>">

        <input type="hidden" name="image" value="<?php echo htmlspecialchars($img); ?>">

        <button type="submit" name="action" value="view">View</button>

        <button type="submit" name="action" value="download">Download</button>

        <button type="submit" name="action" value="delete" onclick="return confirm('Are you sure you want to delete this image?');">Delete</button>

      </form>

    </div>

  <?php endforeach; ?>

  </div>

  <button onclick="window.location.href='drive.php'" aria-label="Back to Drive">Back to Drive</button>

</body>

</html>