

FRONT end

Source Code

Front End

```
import Link from "next/link";
import Header from "../components/Header";
import Footer from "../components/Footer";

export default function HomePage() {
  return (
    <main className="bg-gray-50 min-h-screen flex flex-col">
      <Header />

      {/* Landing Page*/}
      <section className="flex flex-col items-center text-center
py-12 bg-gray-50 flex-grow">
        <h1 className="text-[20px] md:text-[80px] font-bold mb-2
text-black">
          Stati
          <span className="text-[#1848a0]">
            Calcs
          </span>
        </h1>
        <p className="text-gray-600 mb-12 text-[18px]">
          Interactive calculators for learning and solving
          Statics of Rigid Bodies.
        </p>

        {/* Chapter List */}
        <div className="w-full max-w-2xl">
          <div className="grid grid-cols-2 gap-4 items-center">

            {/* Chapter 1 */}
            <p className="text-left text-black text-[18px]
font-bold font-bold">Chapter 1: Introduction to Statics</p>

            <Link
              href="/Introduction"
              className="bg-[#1848a0] text-white px-6 py-3
rounded-md shadow hover:bg-[#163d8a] transition text-[18px]"
            >
              Introduction
            </Link>

            {/* Chapter 2 */}
```

```
        <p className="text-left text-black text-[18px]
font-bold font-bold">Chapter 2: Force Systems</p>
        <Link
            href="/2D-solver"
            className="bg-[#1848a0] text-white px-6 py-3
rounded-md shadow hover:bg-[#163d8a] transition text-[18px]"
        >
            2D Resultant Solver
        </Link>
```

```
        {/* Chapter 3 */}
        <p className="text-left text-black text-[18px]
font-bold font-bold">Chapter 3: Equilibrium</p>
        <Link
            href="/Equilibrium"
            className="bg-[#1848a0] text-white px-6 py-3
rounded-md shadow hover:bg-[#163d8a] transition text-[18px]"
        >
            Equilibrium Solver
        </Link>
```

```
        {/* Chapter 4 */}
        <p className="text-left text-black text-[18px]
font-bold font-bold font-bold">Chapter 4: Structures</p>
        <Link
            href="/Structures"
            className="bg-[#1848a0] text-white px-6 py-3
rounded-md shadow hover:bg-[#163d8a] transition text-[18px]"
        >
            Truss Calculator
        </Link>
```

```
        {/* Chapter 5 */}
        <p className="text-left text-black text-[18px]
font-bold font-bold font-bold">Chapter 5: Distributed Loads</p>
        <Link
            href="/Distributed-Loads"
            className="bg-[#1848a0] text-white px-6 py-3
rounded-md shadow hover:bg-[#163d8a] transition text-[18px]"
        >
            Structures Solver
        </Link>
```

```
        {/* Chapter 6 */}
        <p className="text-left text-black text-[18px]
font-bold font-bold font-bold">Chapter 6: Friction</p>
```

```
        <button className="border-2 border-[#1848a0]
text-[#1848a0] px-6 py-3 rounded-md hover:bg-[#163d8a]
hover:text-white transition text-[18px]">
            Coming Soon
        </button>
```

```
        { /* Chapter 7 */ }
        <p className="text-left text-black text-[18px]
font-bold font-bold font-bold">Chapter 7: Virtual Work</p>
        <button className="border-2 border-[#1848a0]
text-[#1848a0] px-6 py-3 rounded-md hover:bg-[#163d8a]
hover:text-white transition text-[18px]">
            Coming Soon
        </button>
```

```
    </div>
</div>
</section>
```

```
    <Footer />
</main>
);
}
```

2D solver code

2D Solver Code

```
"use client";

import { useRef, useState } from "react";
import Header from "../../components/Header";
import Footer from "../../components/Footer";
import "katex/dist/katex.min.css";
import { BlockMath } from "react-katex";

/* Force System Logic */
class ForceSystem2D {
  vectors: { fx: number; fy: number; magnitude: number; angleDeg: number }[];

  constructor() {
    this.vectors = [];
  }

  addForce(magnitude: number, angleDeg: number) {
    const angleRad = (angleDeg * Math.PI) / 180;
    const fx = magnitude * Math.cos(angleRad);
    const fy = magnitude * Math.sin(angleRad);
    this.vectors.push({ fx, fy, magnitude, angleDeg });
  }

  stepByStepSolution() {
    const steps: string[] = [];
    steps.push("Step 1: Resolve each force into components:");

    let sumFx = 0;
    let sumFy = 0;

    this.vectors.forEach((v, i) => {
      steps.push(
        `\\text{Force } ${i + 1}: `
        + `|F|=${v.magnitude}\\,\\text{kN},\\,; \\theta=${v.angleDeg}^\\circ`
      );

      steps.push(
        `\\begin{align*}`
        + `F_{x${i + 1}} &= `
        + `${v.magnitude}\\cos(${v.angleDeg}^\\circ) \\\\`
      );
    });
  }
}
```

```

        &=  $\{v.fx.toFixed(3)\} \text{ kN}$ 
        F_{y_{i+1}} &=
 $\{v.magnitude\} \sin(\{v.angleDeg\}^\circ)$ 
        &=  $\{v.fy.toFixed(3)\} \text{ kN}$ 
    \end{align*}
`);

    sumFx += v.fx;
    sumFy += v.fy;
});

steps.push("Step 2: Sum of components:");
steps.push(`
    \begin{align*}
    \Sigma F_x &= \{sumFx.toFixed(3)\} \text{ kN} \\
    \Sigma F_y &= \{sumFy.toFixed(3)\} \text{ kN}
    \end{align*}
`);

const R = Math.hypot(sumFx, sumFy);
const theta = (Math.atan2(sumFy, sumFx) * 180) / Math.PI;

const arrow = theta >= 0 ? "↗" : "↘";

steps.push("Step 3: Resultant force:");
steps.push(`
    \begin{align*}
    R &= \sqrt{(\Sigma F_x)^2 + (\Sigma F_y)^2} \\
    &= \{R.toFixed(3)\} \text{ kN} \\
    \theta &= \tan^{-1}\left(\frac{\Sigma F_y}{\Sigma F_x}\right) \\
    &= \{\theta.toFixed(2)\}^\circ \\
    \text{\{arrow\}} &\text{ from +x axis}
    \end{align*}
`);

    return { steps, sumFx, sumFy, R, theta };
}
}

type ForceInput = {
    magnitude: string;
    angle: string;
};

```

```

type ForceResult = {
  steps: string[];
  sumFx: number;
  sumFy: number;
  R: number;
  theta: number;
};

/*FULL FBD FOR STEP 4 (all forces + resultant)*/
function ResultantFBD({
  forces,
  result,
}: {
  forces: ForceInput[];
  result: ForceResult;
}) {
  const vectors = forces
    .map((f) => {
      const m = parseFloat(f.magnitude);
      const a = parseFloat(f.angle);
      if (isNaN(m) || isNaN(a)) return null;
      const rad = (a * Math.PI) / 180;
      return { x: m * Math.cos(rad), y: m * Math.sin(rad) };
    })
    .filter(Boolean) as { x: number; y: number }[];

  const R = { x: result.sumFx, y: result.sumFy };

  const magnitudes = [
    ...vectors.map((v) => Math.hypot(v.x, v.y)),
    Math.hypot(R.x, R.y),
  ];
  const maxMag = Math.max(1, ...magnitudes);

  const scale = 90 / maxMag;

  return (
    <svg
      width="300"
      height="300"
      className="border rounded-lg bg-white shadow mx-auto"
    >
      <g transform="translate(150,150)">
        { /* Axes */ }
      </g>
    </svg>
  );
}

```



```

    <line x1={-140} y1={0} x2={140} y2={0} stroke="gray"
strokeWidth="1" />
    <line x1={0} y1={-140} x2={0} y2={140} stroke="gray"
strokeWidth="1" />

```

```

    { /* Draw each force */
    {vectors.map((v, i) => {
      const x = v.x * scale;
      const y = -v.y * scale;

      return (
        <g key={i}>
          <line
            x1={0}
            y1={0}
            x2={x}
            y2={y}
            stroke="#1848a0"
            strokeWidth="3"
            markerEnd="url(#arrowF) "
          />
          <text
            x={x * 0.55}
            y={y * 0.55}
            fontSize="14"
            fill="#1848a0"
            fontWeight="bold"
          >
            F{i + 1}
          </text>
        </g>
      );
    })}

```

```

    { /* Draw resultant */
    <line
      x1={0}
      y1={0}
      x2={R.x * scale}
      y2={-R.y * scale}
      stroke="#009900"
      strokeWidth="4"
      markerEnd="url(#arrowR) "
    />
    <text
      x={(R.x * scale) * 0.55}
      y={(-R.y * scale) * 0.55}
      fontSize="16"

```

```

        fill="#009900"
        fontWeight="bold"
    >
        R
    </text>

    { /* Arrow definitions */ }
    <defs>
        <marker
            id="arrowF"
            markerWidth="10"
            markerHeight="10"
            refX="5"
            refY="3"
            orient="auto"
        >
            <polygon points="0 0, 6 3, 0 6" fill="#1848a0" />
        </marker>

        <marker
            id="arrowR"
            markerWidth="12"
            markerHeight="12"
            refX="6"
            refY="3"
            orient="auto"
        >
            <polygon points="0 0, 7 3, 0 6" fill="#009900" />
        </marker>
    </defs>
</g>
</svg>
);
}

/*SVG FBD Component (draggable + resultant)*/
function FBD({ forces, setForces }: { forces: ForceInput[];
setForces: (f: ForceInput[]) => void }) {
    const svgRef = useRef<SVGSVGElement | null>(null);
    const [dragIndex, setDragIndex] = useState<number |
null>(null);

    // Convert forces to vectors (math coords; y positive up)
    const vectors = forces
        .map((f) => {
            const m = parseFloat(f.magnitude);
            const a = parseFloat(f.angle);

```

```

    if (isNaN(m) || isNaN(a)) return null;
    const rad = (a * Math.PI) / 180;
    return {
      x: m * Math.cos(rad),
      y: m * Math.sin(rad),
    };
  })
  .filter(Boolean) as { x: number; y: number }[];

// Determine scale so arrows fit nicely
const maxMag = Math.max(1, ...vectors.map((v) =>
Math.hypot(v.x, v.y)));
const scale = 80 / maxMag; // dynamic scale

const screenPointToSvg = (clientX: number, clientY: number) =>
{
  const svg = svgRef.current;
  if (!svg) return null;
  const pt = svg.createSVGPoint();
  pt.x = clientX;
  pt.y = clientY;
  const ctm = svg.getScreenCTM();
  if (!ctm) return null;
  return pt.matrixTransform(ctm.inverse());
};

const handleMouseMove = (e: React.MouseEvent) => {
  if (dragIndex === null) return;

  const svg = svgRef.current;
  if (!svg) return;

  const pt = svg.createSVGPoint();
  pt.x = e.clientX;
  pt.y = e.clientY;

  // convert cursor to SVG coordinates
  const cursor =
pt.matrixTransform(svg.getScreenCTM()?.inverse());
  const x = cursor.x - 150;
  const y = cursor.y - 150;

```

```

const newAngle = (Math.atan2(-y, x) * 180) / Math.PI;

const newForces = [...forces];

newForces[dragIndex] = {
  ...newForces[dragIndex],
  angle: newAngle.toFixed(3), // Only angle changes
};

setForces(newForces);
};

const stopDrag = () => setDragIndex(null);

// compute resultant in math coords
const sum = vectors.reduce((acc, v) => ({ x: acc.x + v.x, y:
acc.y + v.y }), { x: 0, y: 0 });
const Rx = sum.x * scale;
const Ry = -sum.y * scale; // svg y inverted

return (
  <svg
    ref={svgRef}
    width="300"
    height="300"
    className="border rounded-lg bg-white shadow"
    style={{ background: "white" }}
    onMouseMove={handleMouseMove}
    onMouseUp={stopDrag}
    onMouseLeave={stopDrag}
  >
    <g transform="translate(150,150)">
      /* Axes */
      <line x1={-140} y1={0} x2={140} y2={0} stroke="gray"
strokeWidth="1" />
      <line x1={0} y1={-140} x2={0} y2={140} stroke="gray"
strokeWidth="1" />

      /* Force vectors */
      {vectors.map((v, i) => {
        const x = v.x * scale;
        const y = -v.y * scale; // invert for svg

```

```

        return (
          <g key={i}>
            <line
              x1={0}
              y1={0}
              x2={x}
              y2={y}
              stroke="#1848a0"
              strokeWidth="3"
              markerEnd="url(#arrow)"
              className="cursor-pointer"
              onMouseDown={() => setDragIndex(i)}
            />
            <text x={x * 0.55} y={y * 0.55} fontSize="14"
fill="black">
              F{i + 1}
            </text>
          </g>
        );
      }}}

      {/* Arrow definitions */}
      <defs>
        <marker id="arrow" markerWidth="10" markerHeight="10"
refX="5" refY="3" orient="auto">
          <polygon points="0 0, 6 3, 0 6" fill="#1848a0" />
        </marker>

        <marker id="arrowR" markerWidth="12" markerHeight="12"
refX="6" refY="3" orient="auto">
          <polygon points="0 0, 7 3, 0 6" fill="#009900" />
        </marker>
      </defs>
    </g>
  </svg>

  );
}

/* ===== MAIN COMPONENT ===== */
export default function Solver2D() {
  const [forces, setForces] = useState<ForceInput[]>([
    {
      magnitude: "", angle: ""
    }
  ]);

  const [result, setResult] = useState<ForceResult | null>(null);

```

```

    const handleInputChange = (index: number, field: "magnitude" |
"angle", value: string) => {
      const newForces = [...forces];
      newForces[index][field] = value;
      setForces(newForces);
    };

    const calculateResultant = () => {
      const system = new ForceSystem2D();

      forces.forEach((f) => {
        const mag = parseFloat(f.magnitude);
        const ang = parseFloat(f.angle);
        if (!isNaN(mag) && !isNaN(ang)) system.addForce(mag, ang);
      });

      setResult(system.stepByStepSolution());
    };

    return (
      <div className="flex flex-col min-h-screen bg-gray-50
text-gray-900 text-[18px]">
        <Header />

        <main className="flex-grow flex flex-col items-center px-4
py-10">
          <h1 className="text-[32px] font-bold mb-6">2D Resultant
Force Calculator</h1>

          {/* FBD Live Preview */}
          <div className="mb-8">
            <h2 className="text-[20px] font-semibold text-center
mb-2">Real-Time Free Body Diagram</h2>
            <FBD forces={forces} setForces={setForces} />
          </div>

          {/* Inputs */}
          <div className="w-full max-w-xl bg-white rounded-2xl
shadow p-6 space-y-6">
            <h2 className="text-[20px] font-semibold">Force
setup</h2>

```

```

        <div className="grid grid-cols-2 gap-4">
          {forces.map((f, i) => (
            <div key={i} className="col-span-2 flex gap-4
items-end">
              <div className="flex-1">
                <label className="block font-medium
text-[18px]">
                  Force {i + 1} (kN)
                </label>
                <input
                  type="number"
                  value={f.magnitude}
                  onChange={ (e) =>
                    e.target.value)
                  }
                  placeholder="Magnitude (kN)"
                  className="w-full mt-1 rounded-lg
border-gray-300 text-[18px] p-2"
                />
              </div>
              <div className="flex-1">
                <label className="block font-medium
text-[18px]">
                  Angle {i + 1} (°)
                </label>
                <input
                  type="number"
                  value={f.angle}
                  onChange={ (e) =>
                    e.target.value)
                  }
                  placeholder="Angle (deg)"
                  className="w-full mt-1 rounded-lg
border-gray-300 text-[18px] p-2"
                />
              </div>
              {forces.length > 1 && (
                <button
                  onClick={ () =>
                    setForces(forces.filter((_, idx) => idx !==
i))
                  }
                  className="px-3 py-1 bg-red-500 text-white
rounded-lg hover:bg-red-600 text-[18px]"
                >
                  -
                </button>
              )}
            </div>
          )}
        </div>

```

```

    ))}
  </div>
  <button onClick={() => setForces([...forces, {
magnitude: "", angle: "" }])} className="w-full bg-[#008409]
text-white py-3 rounded-lg hover:bg-[#15711b] transition
text-[18px]">
    + Add Force
  </button>

  <button onClick={calculateResultant} className="w-full
bg-[#1848a0] text-white py-3 rounded-lg hover:bg-[#163d8a]
transition text-[18px]">
    Calculate
  </button>
</div>

  { /* Output */ }
  {result && (
    <div className="w-full max-w-xl mt-6 bg-white
rounded-2xl shadow p-6 space-y-4">
      <h2 className="text-[20px] font-semibold">Resultant
Force (kN)</h2>
      <div>
        <label className="block font-medium
text-[18px]">Horizontal component (Fx)</label>
        <input type="text"
value={` ${result.sumFx.toFixed(3)} kN` } readOnly
className="w-full mt-1 rounded-lg border-gray-300 text-[18px]
p-2" />
      </div>

      <div>
        <label className="block font-medium
text-[18px]">Vertical component (Fy)</label>
        <input type="text"
value={` ${result.sumFy.toFixed(3)} kN` } readOnly
className="w-full mt-1 rounded-lg border-gray-300 text-[18px]
p-2" />
      </div>

      <div>
        <label className="block font-medium
text-[18px]">Magnitude of resultant force (R)</label>
        <input type="text" value={` ${result.R.toFixed(3)}
kN` } readOnly className="w-full mt-1 rounded-lg border-gray-300
text-[18px] p-2" />
      </div>

```



```

        <div>
            <label className="block font-medium
text-[18px]">Direction of resultant force ( $\theta$ )</label>
            <input type="text"
value={`\${result.theta.toFixed(2)}^\circ`} readOnly className="w-full
mt-1 rounded-lg border-gray-300 text-[18px] p-2" />
        </div>
    </div>
    )}

```

```

    {/* Step-by-Step Solution */}
    {result && (
        <div className="w-full max-w-xl mt-6 bg-white
rounded-2xl shadow p-6">
            <h2 className="text-[20px] font-semibold
mb-2">Step-by-Step Solution</h2>

```

```

        <div className="space-y-4">
            {result.steps.map((line, i) =>
                line.startsWith("Step") ? (
                    <p key={i} className="font-medium text-[18px]">
                        {line}
                    </p>
                ) : (
                    <div key={i} className="text-[18px]">
                        <BlockMath>{line}</BlockMath>
                    </div>
                )
            )}
        </div>

```

```

    {/* Step 4 */}
    <div className="mt-8">
        <p className="font-medium text-[18px] mb-2">
            Step 4: Final Free Body Diagram
        </p>
        <ResultantFBD forces={forces} result={result} />
    </div>
</div>
    )}
</main>

```

```

<Footer />

```

```

    </div>
  );
}

"use client";

import { useRef, useState } from "react";
import Header from "../../components/Header";
import Footer from "../../components/Footer";
import "katex/dist/katex.min.css";
import { BlockMath } from "react-katex";

/* ===== Force System Logic =====
*/
class ForceSystem2D {
  vectors: { fx: number; fy: number; magnitude: number; angleDeg:
number }[];

  constructor() {
    this.vectors = [];
  }

  addForce(magnitude: number, angleDeg: number) {
    const angleRad = (angleDeg * Math.PI) / 180;
    const fx = magnitude * Math.cos(angleRad);
    const fy = magnitude * Math.sin(angleRad);
    this.vectors.push({ fx, fy, magnitude, angleDeg });
  }

  stepByStepSolution() {
    const steps: string[] = [];
    steps.push("Step 1: Resolve each force into components:");

    let sumFx = 0;
    let sumFy = 0;

    this.vectors.forEach((v, i) => {
      steps.push(
        `\\text{Force } ${i + 1}: `
        |F|=${v.magnitude}\\,\\text{kN},\\; \\theta=${v.angleDeg}^\\circ`
      );
    });
  }
}

```

```

        steps.push(`
            \\begin{align*}
                F_{x${i + 1}} &=
                ${v.magnitude}\\cos(${v.angleDeg}^\\circ) \\\\
                &= ${v.fx.toFixed(3)}\\,,\\text{kN} \\\\
                F_{y${i + 1}} &=
                ${v.magnitude}\\sin(${v.angleDeg}^\\circ) \\\\
                &= ${v.fy.toFixed(3)}\\,,\\text{kN}
            \\end{align*}
        `);

        sumFx += v.fx;
        sumFy += v.fy;
    });

    steps.push("Step 2: Sum of components:");
    steps.push(`
        \\begin{align*}
            \\Sigma F_x &= ${sumFx.toFixed(3)}\\,,\\text{kN} \\\\
            \\Sigma F_y &= ${sumFy.toFixed(3)}\\,,\\text{kN}
        \\end{align*}
    `);

    const R = Math.hypot(sumFx, sumFy);
    const theta = (Math.atan2(sumFy, sumFx) * 180) / Math.PI;

    const arrow = theta >= 0 ? "↗" : "↖";

    steps.push("Step 3: Resultant force:");
    steps.push(`
        \\begin{align*}
            R &= \\sqrt{(\\Sigma F_x)^2 + (\\Sigma F_y)^2} \\\\
            &= ${R.toFixed(3)}\\,,\\text{kN} \\\\
            \\theta &= \\tan^{-1}\\left(\\tfrac{\\Sigma F_y}{\\Sigma F_x}\\right) \\\\
            &= ${theta.toFixed(2)}^\\circ
            ${arrow}\\,,\\text{from +x axis}
        \\end{align*}
    `);

    return { steps, sumFx, sumFy, R, theta };
}

```

```

type ForceInput = {
  magnitude: string;
  angle: string;
};

type ForceResult = {
  steps: string[];
  sumFx: number;
  sumFy: number;
  R: number;
  theta: number;
};

/*FULL FBD FOR STEP 4 (all forces + resultant)*/
function ResultantFBD({
  forces,
  result,
}: {
  forces: ForceInput[];
  result: ForceResult;
}) {
  const vectors = forces
    .map((f) => {
      const m = parseFloat(f.magnitude);
      const a = parseFloat(f.angle);
      if (isNaN(m) || isNaN(a)) return null;
      const rad = (a * Math.PI) / 180;
      return { x: m * Math.cos(rad), y: m * Math.sin(rad) };
    })
    .filter(Boolean) as { x: number; y: number }[];

  const R = { x: result.sumFx, y: result.sumFy };

  const magnitudes = [
    ...vectors.map((v) => Math.hypot(v.x, v.y)),
    Math.hypot(R.x, R.y),
  ];
  const maxMag = Math.max(1, ...magnitudes);

  const scale = 90 / maxMag;

  return (
    <svg
      width="300"
      height="300"

```

```

        className="border rounded-lg bg-white shadow mx-auto"
    >
    <g transform="translate(150,150)">
        {/* Axes */}
        <line x1={-140} y1={0} x2={140} y2={0} stroke="gray"
strokeWidth="1" />
        <line x1={0} y1={-140} x2={0} y2={140} stroke="gray"
strokeWidth="1" />

        {/* Draw each force */}
        {vectors.map((v, i) => {
            const x = v.x * scale;
            const y = -v.y * scale;

            return (
                <g key={i}>
                    <line
                        x1={0}
                        y1={0}
                        x2={x}
                        y2={y}
                        stroke="#1848a0"
                        strokeWidth="3"
                        markerEnd="url(#arrowF)"
                    />
                    <text
                        x={x * 0.55}
                        y={y * 0.55}
                        fontSize="14"
                        fill="#1848a0"
                        fontWeight="bold"
                    >
                        F{i + 1}
                    </text>
                </g>
            );
        })}

        {/* Draw resultant */}
        <line
            x1={0}
            y1={0}
            x2={R.x * scale}
            y2={-R.y * scale}
            stroke="#009900"
            strokeWidth="4"
            markerEnd="url(#arrowR)"
        />

```

```

    <text
      x={(R.x * scale) * 0.55}
      y={(-R.y * scale) * 0.55}
      fontSize="16"
      fill="#009900"
      fontWeight="bold"
    >
      R
    </text>

    {/* Arrow definitions */}
    <defs>
      <marker
        id="arrowF"
        markerWidth="10"
        markerHeight="10"
        refX="5"
        refY="3"
        orient="auto"
      >
        <polygon points="0 0, 6 3, 0 6" fill="#1848a0" />
      </marker>

      <marker
        id="arrowR"
        markerWidth="12"
        markerHeight="12"
        refX="6"
        refY="3"
        orient="auto"
      >
        <polygon points="0 0, 7 3, 0 6" fill="#009900" />
      </marker>
    </defs>
  </g>
</svg>
);
}

/*SVG FBD Component (draggable + resultant)*/
function FBD({ forces, setForces }: { forces: ForceInput[];
setForces: (f: ForceInput[]) => void }) {
  const svgRef = useRef<SVGSVGElement | null>(null);
  const [dragIndex, setDragIndex] = useState<number |
null>(null);

  // Convert forces to vectors (math coords; y positive up)

```

```

const vectors = forces
  .map((f) => {
    const m = parseFloat(f.magnitude);
    const a = parseFloat(f.angle);
    if (isNaN(m) || isNaN(a)) return null;
    const rad = (a * Math.PI) / 180;
    return {
      x: m * Math.cos(rad),
      y: m * Math.sin(rad),
    };
  })
  .filter(Boolean) as { x: number; y: number }[];

// Determine scale so arrows fit nicely
const maxMag = Math.max(1, ...vectors.map((v) =>
Math.hypot(v.x, v.y)));
const scale = 80 / maxMag; // dynamic scale

const screenPointToSvg = (clientX: number, clientY: number) =>
{
  const svg = svgRef.current;
  if (!svg) return null;
  const pt = svg.createSVGPoint();
  pt.x = clientX;
  pt.y = clientY;
  const ctm = svg.getScreenCTM();
  if (!ctm) return null;
  return pt.matrixTransform(ctm.inverse());
};

const handleMouseMove = (e: React.MouseEvent) => {
  if (dragIndex === null) return;

  const svg = svgRef.current;
  if (!svg) return;

  const pt = svg.createSVGPoint();
  pt.x = e.clientX;
  pt.y = e.clientY;

  // convert cursor to SVG coordinates
  const cursor =
pt.matrixTransform(svg.getScreenCTM()?.inverse());
  const x = cursor.x - 150;
  const y = cursor.y - 150;

```

```

const newAngle = (Math.atan2(-y, x) * 180) / Math.PI;

const newForces = [...forces];

newForces[dragIndex] = {
  ...newForces[dragIndex],
  angle: newAngle.toFixed(3), // Only angle changes
};

setForces(newForces);
};

const stopDrag = () => setDragIndex(null);

// compute resultant in math coords
const sum = vectors.reduce((acc, v) => ({ x: acc.x + v.x, y:
acc.y + v.y }), { x: 0, y: 0 });
const Rx = sum.x * scale;
const Ry = -sum.y * scale; // svg y inverted

return (
  <svg
    ref={svgRef}
    width="300"
    height="300"
    className="border rounded-lg bg-white shadow"
    style={{ background: "white" }}
    onMouseMove={handleMouseMove}
    onMouseUp={stopDrag}
    onMouseLeave={stopDrag}
  >
    <g transform="translate(150,150)">
      { /* Axes */ }
      <line x1={-140} y1={0} x2={140} y2={0} stroke="gray"
strokeWidth="1" />
      <line x1={0} y1={-140} x2={0} y2={140} stroke="gray"
strokeWidth="1" />

```



```

    { /* Force vectors */
    {vectors.map((v, i) => {
      const x = v.x * scale;
      const y = -v.y * scale; // invert for svg
      return (
        <g key={i}>
          <line
            x1={0}
            y1={0}
            x2={x}
            y2={y}
            stroke="#1848a0"
            strokeWidth="3"
            markerEnd="url(#arrow)"
            className="cursor-pointer"
            onMouseDown={() => setDragIndex(i)}
          />
          <text x={x * 0.55} y={y * 0.55} fontSize="14"
fill="black">
            F{i + 1}
          </text>
        </g>
      );
    })}

    { /* Arrow definitions */
    <defs>
      <marker id="arrow" markerWidth="10" markerHeight="10"
refX="5" refY="3" orient="auto">
        <polygon points="0 0, 6 3, 0 6" fill="#1848a0" />
      </marker>

      <marker id="arrowR" markerWidth="12" markerHeight="12"
refX="6" refY="3" orient="auto">
        <polygon points="0 0, 7 3, 0 6" fill="#009900" />
      </marker>
    </defs>
  </g>
</svg>

);
}

/* MAIN COMPONENT */
export default function Solver2D() {
  const [forces, setForces] = useState<ForceInput[]>([
    { magnitude: "", angle: "" }]);

```

```

const [result, setResult] = useState<ForceResult | null>(null);

const handleInputChange = (index: number, field: "magnitude" |
"angle", value: string) => {
  const newForces = [...forces];
  newForces[index][field] = value;
  setForces(newForces);
};

const calculateResultant = () => {
  const system = new ForceSystem2D();

  forces.forEach((f) => {
    const mag = parseFloat(f.magnitude);
    const ang = parseFloat(f.angle);
    if (!isNaN(mag) && !isNaN(ang)) system.addForce(mag, ang);
  });

  setResult(system.stepByStepSolution());
};

return (
  <div className="flex flex-col min-h-screen bg-gray-50
text-gray-900 text-[18px]">
    <Header />

    <main className="flex-grow flex flex-col items-center px-4
py-10">
      <h1 className="text-[32px] font-bold mb-6">2D Resultant
Force Calculator</h1>

      <div className="mb-8">
        <h2 className="text-[20px] font-semibold text-center
mb-2">Real-Time Free Body Diagram</h2>
        <FBD forces={forces} setForces={setForces} />
      </div>

      <div className="w-full max-w-xl bg-white rounded-2xl
shadow p-6 space-y-6">

```

```

        <h2 className="text-[20px] font-semibold">Force
setup</h2>

        <div className="grid grid-cols-2 gap-4">
          {forces.map((f, i) => (
            <div key={i} className="col-span-2 flex gap-4
items-end">
              <div className="flex-1">
                <label className="block font-medium
text-[18px]">
                  Force {i + 1} (kN)
                </label>
                <input
                  type="number"
                  value={f.magnitude}
                  onChange={ (e) =>
                    handleInputChange(i, "magnitude",
e.target.value)
                  }
                  placeholder="Magnitude (kN)"
                  className="w-full mt-1 rounded-lg
border-gray-300 text-[18px] p-2"
                />
              </div>
              <div className="flex-1">
                <label className="block font-medium
text-[18px]">
                  Angle {i + 1} (°)
                </label>
                <input
                  type="number"
                  value={f.angle}
                  onChange={ (e) =>
                    handleInputChange(i, "angle",
e.target.value)
                  }
                  placeholder="Angle (deg)"
                  className="w-full mt-1 rounded-lg
border-gray-300 text-[18px] p-2"
                />
              </div>
              {forces.length > 1 && (
                <button
                  onClick={ () =>
                    setForces(forces.filter((_, idx) => idx !==
i))
                  }
                  className="px-3 py-1 bg-red-500 text-white
rounded-lg hover:bg-red-600 text-[18px]"
                >

```

```

        -
        </button>
    })
</div>
    )})
</div>
    <button onClick={() => setForces([...forces, {
magnitude: "", angle: "" }])} className="w-full bg-[#008409]
text-white py-3 rounded-lg hover:bg-[#15711b] transition
text-[18px]">
        + Add Force
    </button>

    <button onClick={calculateResultant} className="w-full
bg-[#1848a0] text-white py-3 rounded-lg hover:bg-[#163d8a]
transition text-[18px]">
        Calculate
    </button>
</div>

    { /* Output */ }
    {result && (
        <div className="w-full max-w-xl mt-6 bg-white
rounded-2xl shadow p-6 space-y-4">
            <h2 className="text-[20px] font-semibold">Resultant
Force (kN)</h2>
            <div>
                <label className="block font-medium
text-[18px]">Horizontal component (Fx)</label>
                <input type="text"
value={` ${result.sumFx.toFixed(3)} kN` } readOnly
className="w-full mt-1 rounded-lg border-gray-300 text-[18px]
p-2" />
            </div>

            <div>
                <label className="block font-medium
text-[18px]">Vertical component (Fy)</label>
                <input type="text"
value={` ${result.sumFy.toFixed(3)} kN` } readOnly
className="w-full mt-1 rounded-lg border-gray-300 text-[18px]
p-2" />
            </div>

            <div>
                <label className="block font-medium
text-[18px]">Magnitude of resultant force (R)</label>

```

```

        <input type="text" value={`\${result.R.toFixed(3)}
kN`} readOnly className="w-full mt-1 rounded-lg border-gray-300
text-[18px] p-2" />
      </div>

```

```

    <div>
      <label className="block font-medium
text-[18px]">Direction of resultant force ( $\theta$ )</label>
      <input type="text"
value={`\${result.theta.toFixed(2)}^\circ`} readOnly className="w-full
mt-1 rounded-lg border-gray-300 text-[18px] p-2" />
    </div>
  </div>
)}

```

```

  {/* Step-by-Step Solution */}
  {result && (
    <div className="w-full max-w-xl mt-6 bg-white
rounded-2xl shadow p-6">
      <h2 className="text-[20px] font-semibold
mb-2">Step-by-Step Solution</h2>

```

```

    <div className="space-y-4">
      {result.steps.map((line, i) =>
        line.startsWith("Step") ? (
          <p key={i} className="font-medium text-[18px]">
            {line}
          </p>
        ) : (
          <div key={i} className="text-[18px]">
            <BlockMath>{line}</BlockMath>
          </div>
        )
      )}
    </div>

```

```

  {/* Step 4 */}
  <div className="mt-8">
    <p className="font-medium text-[18px] mb-2">
      Step 4: Final Free Body Diagram
    </p>
    <ResultantFBD forces={forces} result={result} />
  </div>
</div>
)}

```

```
</main>
```

```
<Footer />
```

```
</div>
```

```
);
```

```
}
```

Footer

Footer

```
"use client";

import Link from "next/link";

type NavLink = { label: string; href: string };

interface FooterProps {
  links?: NavLink[];
}

export default function Footer({
  links = [
    { label: "About", href: "/about" },
    { label: "References", href: "/reference" },
    { label: "Contact", href: "/contact" },
    { label: "Developer", href: "/developers" },
  ],
}: FooterProps) {
  return (
    <footer className="bg-white border-t mt-auto">
      <div className="max-w-7xl mx-auto px-6 py-4 text-center text-gray-700 text-[18px]">
        {/* Desktop: horizontal links with | separator */}
        <div className="hidden sm:flex justify-center flex-wrap gap-4">
          {links.map((link, idx) => (
            <span key={link.href} className="flex items-center text-[18px]">
              <Link href={link.href}
                className="hover:text-blue-600">
                {link.label}
              </Link>
              {idx < links.length - 1 && (
                <span className="mx-2 text-gray-400">|</span>
              )}
            </span>
          )
        )}
        </div>

        {/* Mobile: stacked links */}
        <div className="flex flex-col sm:hidden gap-2">
          {links.map((link) => (
            <Link
```



```
        key={link.href}
        href={link.href}
        className="hover:text-blue-600 text-[18px]"
      >
        {link.label}
      </Link>
    )}
  </div>
</div>
</footer>
);
}
```

Header

Header

```
"use client";

import Link from "next/link";

export default function Header() {
  return (
    <header className="bg-white shadow">
      <div className="max-w-7xl mx-auto px-6 py-4 flex flex-col
sm:flex-row sm:items-center sm:justify-between gap-4">

        {/* Logo + Title */}
        <Link
          href="/"
          className="flex items-center gap-3 justify-center
sm:justify-start hover:text-[#1848a0] transition"
        >
          <div className="w-10 h-10 border-2 border-black
rounded-full" />
          <span className="font-bold text-[30px]
text-black">Statics Calculator</span>
        </Link>

        {/* Desktop Navigation */}
        <nav className="hidden sm:flex items-center space-x-6
text-gray-700 relative text-[18px]">
          <Link href="/"
            className="hover:text-[#1848a0]">Home</Link>
          <span>|</span>

          {/* Topics Dropdown */}
          <div className="group relative">
            <button className="hover:text-[#1848a0]">Topics
          </button>
          <div

            className="absolute left-1/2 -translate-x-1/2 mt-2
w-56 bg-white border rounded-lg shadow-lg
opacity-0 group-hover:opacity-100 invisible
group-hover:visible transition text-[18px]"
          >
            <div className="flex flex-col p-2 text-gray-700">
              <Link href="/Introduction"
                className="hover:text-[#1848a0] p-2">
                Chapter 1: Introduction to Statics
              </Link>
            </div>
          </div>
        </nav>
      </div>
    </header>
  );
}
```

```

        <Link href="/2D-solver"
className="hover:text-[#1848a0] p-2">
            Chapter 2: Force Systems
        </Link>
        <Link href="/Equilibrium"
className="hover:text-[#1848a0] p-2">
            Chapter 3: Equilibrium
        </Link>
        <Link href="/Structures"
className="hover:text-[#1848a0] p-2">
            Chapter 4: Structures
        </Link>
        <Link href="/Distributed-Loads"
className="hover:text-[#1848a0] p-2">
            Chapter 5: Distributed Loads
        </Link>
    </div>
</div>
</div>

```

```

        <span>|</span>
        <Link href="/about"
className="hover:text-[#1848a0]">About</Link>
    </nav>

```

```

    { /* Mobile Navigation */ }
    <nav className="flex flex-col sm:hidden items-center
gap-2 text-gray-700 text-[18px]">
        <Link href="/"
className="hover:text-[#1848a0]">Home</Link>

```

```

        <details className="w-full">
            <summary className="cursor-pointer text-center
hover:text-[#1848a0]">Topics</summary>
            <div className="flex flex-col mt-2 gap-2">
                <Link href="/Introduction"
className="hover:text-[#1848a0] p-2">
                    Chapter 1: Introduction to Statics
                </Link>
                <Link href="/2D-solver"
className="hover:text-[#1848a0] p-2">
                    Chapter 2: Force Systems
                </Link>
                <Link href="/Equilibrium"
className="hover:text-[#1848a0] p-2">
                    Chapter 3: Equilibrium
                </Link>

```

```
        <Link href="/Structures"
className="hover:text-[#1848a0] p-2">
            Chapter 4: Structures
        </Link>
        <Link href="/Distributed-Loads"
className="hover:text-[#1848a0] p-2">
            Chapter 5: Distributed Loads
        </Link>
    </div>
</details>
```

```
        <Link href="/about"
className="hover:text-[#1848a0]">About</Link>
    </nav>
</div>
</header>
);
}
```

About page

About page

```
import Header from "<Ian>/components/Header";
import Footer from "<Ian>/components/Footer";

export default function AboutPage() {
  return (
    <div className="min-h-screen flex flex-col bg-gray-50">
      <Header />

      {/* Main Content */}
      <main className="flex flex-1 items-center justify-center
px-6 py-12">
        <div className="max-w-3xl text-center">
          <h1 className="text-2xl font-semibold text-gray-800
mb-6">
            About{" "}
            <span className="text-[#1848a0]">StatiCalcs</span>
          </h1>

          <p className="text-[18px] text-gray-700 leading-relaxed
mb-6">
            <span className="font-bold">
              Stati<span className="text-[#1848a0]">Calcs</span>
            </span>{" "}
            is an interactive web-based learning tool created to
support
            engineering students in their study of Statics of
Rigid Bodies. It
            combines essential concepts with integrated
calculators to help
            users practice problem-solving more effectively.
          </p>

          <p className="text-[18px] text-gray-700
leading-relaxed">
            Designed specifically for engineering students of MSU-Gensan,{" "
            <span className="font-bold">
              Stati<span className="text-[#1848a0]">Calcs</span>
            </span>{" "}
            serves as a supplementary academic tool that enhances classroom
learning,
            encourages independent study, and fosters a deeper
understanding of
            statics principles.
          </p>
```

```
    </div>  
</main>
```

```
    <Footer />  
  </div>  
);  
}
```


Page Contact

```
import Header from "<Ian>/components/Header";
import Footer from "<Ian>/components/Footer";

export default function ContactPage() {
  return (
    <div className="min-h-screen flex flex-col bg-gray-50">
      <Header />

      {/* Main Content */}
      <main className="flex flex-1 items-center justify-center
px-6 py-12">
        <div className="max-w-3xl text-center">
          <h1 className="text-2xl font-semibold text-gray-800
mb-4">
            Contact
          </h1>
          <p className="text-[18px] text-gray-700 mb-8">
            For feedback or inquiries, please reach out through
the following:
          </p>

          <div className="space-y-8 text-left">
            {/* First Contact */}
            <div>
              <h2 className="font-semibold text-[18px]
text-gray-800">
                Ian Carl P. Cona
              </h2>
              <p className="text-[18px] text-gray-700">
                Email: iancarl.cona@msugensan.edu.ph
              </p>
              <p className="text-[18px] text-gray-700">
                Mindanao State University - General Santos
              </p>

              <p className="text-[18px] text-gray-700">
                Fatima, General Santos City, Philippines
              </p>
            </div>

            {/* Second Contact */}
            <div>
              <h2 className="font-semibold text-[18px]
text-gray-800">
                Sophia Daphne C. Faelnar
```

```
</h2>
<p className="text-[18px] text-gray-700">
  Email: sophiadaphne.faelnar@msugensan.edu.ph
</p>
<p className="text-[18px] text-gray-700">
  Mindanao State University - General Santos
</p>
<p className="text-[18px] text-gray-700">
  Fatima, General Santos City, Philippines
</p>
</div>
</div>
</div>
</main>
```

```
<Footer />
</div>
);
}
```