FOSSEE Semester Long Internship Report

On

Development of Chemical PFD Builder

Submitted by

Vaibhav Kumar

3rd Year B.Tech Student, CSE

VIT Bhopal University

Under the Guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

Indian Institute of Technology Bombay

Mentor:

Chirag Koyande

July 4, 2026

Acknowledgments

I would like to express my sincere gratitude to the FOSSEE team at IIT Bombay for the opportunity to participate in the Semester Long Internship program. This experience has been intellectually rewarding, allowing me to contribute to a significant technical project alongside a dedicated team.

As a member of the Web Frontend Team for the Chemical PFD Builder, I contributed to developing the browser-based diagram editor built with React.js and Konva.js, focusing primarily on the connection-routing and pathfinding engine, AI-assisted diagram generation, and the interactive component-resizing system. Beyond feature development, I wrote unit tests for the routing module, documented the web-frontend, and iterated on pull requests through code review. These responsibilities gave me a holistic understanding of building and maintaining a production-grade, open-source frontend.

I am deeply grateful to Prof. Prabhu Ramachandran for his leadership and vision in promoting technical excellence through the FOSSEE initiative.

Special thanks to my mentor, Chirag Koyande, for his invaluable guidance, timely feedback on pull requests, and technical clarity throughout the internship. I also appreciate the support and collaboration of my fellow team members.

This internship has been a defining chapter in my journey, equipping me with the skills and clarity for a career in software engineering.

Contents

Chapter 1

Introduction

1.1 FOSSEE Project

FOSSEE (Free/Libre and Open Source Software for Education) is a project based at IIT Bombay that promotes the use of FLOSS tools in academia and research. It operates under the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education, Government of India.

1.1.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- Textbook Companion: Porting solved examples from textbooks using FLOSS.
- Lab Migration: Facilitating the migration of proprietary labs to FLOSS alternatives.
- Niche Software Activities: Specialized activities to promote niche software tools.
- Forums: Providing a collaborative space for users.
- Workshops and Conferences: Organizing events to train and inform users.

For more details, visit the official FOSSEE website

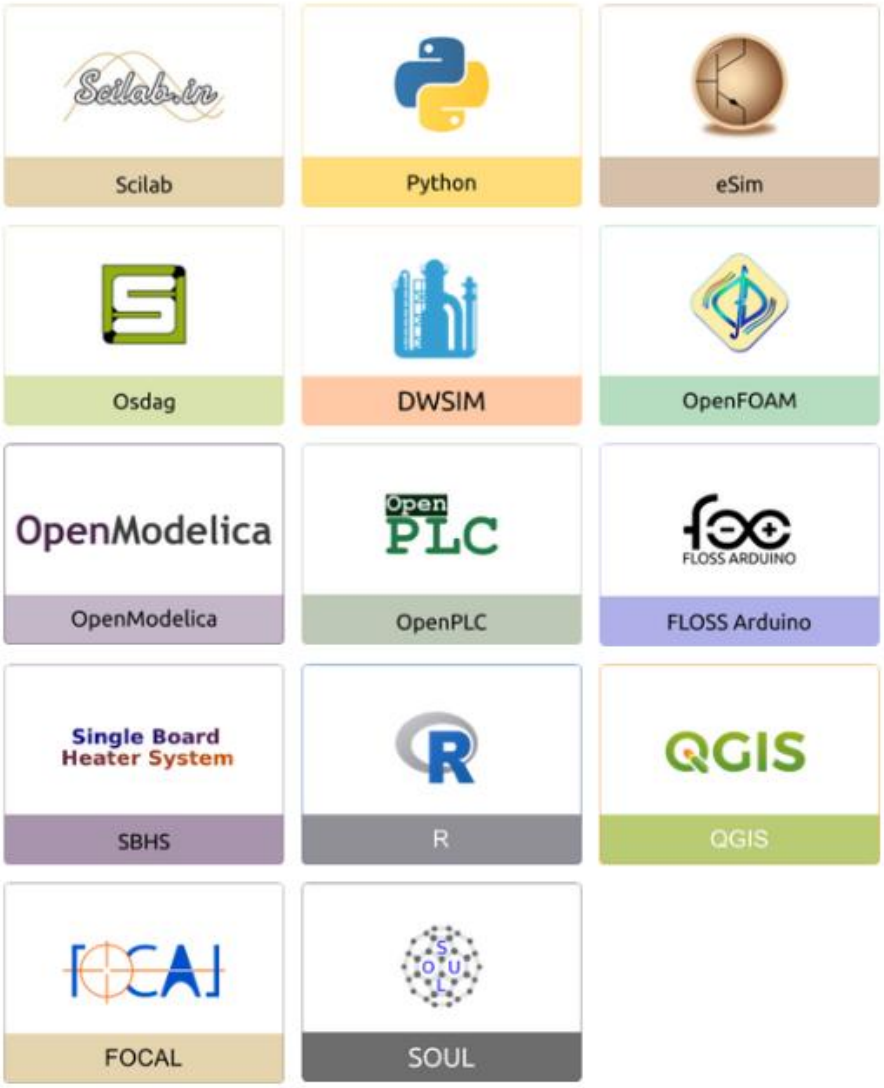

Figure 1.1: FOSSEE Projects and Activities

1.2 Chemical PFD Builder

Chemical PFD Builder is a cross-platform system developed under FOSSEE for creating, editing, and managing Process Flow Diagrams (PFDs) used in chemical engineering. It targets both academic and industrial use, offering a professional diagramming environment with cloud-based project storage and access from both web and desktop clients.

1.2.1 Overview and Purpose

Process Flow Diagrams are central to chemical engineering for visualising equipment, processes, and material flows. Commercial PFD tools are often expensive and closed, which limits collaboration in an academic setting. Chemical PFD Builder is FOSSEE's free, open-source answer to that gap, built to run on both the browser and the desktop.

1.2.2 System Architecture

The application follows a hybrid architecture: a single Django backend serves both a React web frontend and a desktop frontend. The backend exposes REST APIs for authentication, project management, and diagram storage; both frontends share a common data schema, which keeps a project portable between the web and desktop editors.

1.2.3 Key Features

Users build diagrams through drag-and-drop placement of chemical engineering symbols, with automatic component labelling, orthogonal auto-routed connections, PDF/Excel report generation, cloud-synced projects, and export to PNG/JPEG/PDF/SVG.

1.2.4 Future Directions

Future development plans for Chemical PFD Builder include integration with simulation engines, support for additional diagram types such as P&ID, enhanced collaboration features, and deployment on cloud platforms for improved scalability

1.2.5 Conclusion

Chemical PFD Builder represents a significant step forward in providing a free, open source, and feature-rich tool for chemical engineering diagramming. By leveraging modern web technologies and a shared backend, it offers flexibility, collaboration, and accessibility to students, educators, and professionals. The project exemplifies the goals of the FOSSEE initiative to promote open-source software in education.

For more information, visit the GitHub repository: https://github.com/FOSSEE/Chemical-PFD-Web-Desktop

Chapter 2

Screening Task

2.1 Problem Statement

The screening task required building a hybrid web and desktop application for chemical equipment data visualization and analytics.

2.1.1 Project Overview

The internship screening task required building a hybrid Web + Desktop application — the "Chemical Equipment Parameter Visualizer" — that lets a user upload a CSV of chemical equipment (Equipment Name, Type, Flowrate, Pressure, Temperature). A Django backend parses the data, computes summary statistics, and serves them over a REST API; both a React web client and a PyQt5 desktop client consume the same API to render tables, charts, and summaries.

2.1.2 Key Features Required

- **CSV Upload:** Both interfaces upload a CSV to the backend.
- **Data Summary API:** Returns total count, per-field averages, and equipment-type distribution.
- **Visualization:** Charts using Chart.js (Web) and Matplotlib (Desktop).
- **History Management:** Store last 5 uploaded datasets.
- **PDF Report Generation:** Generate analysis reports.
- **Basic Authentication:** User login/registration

- **Fixed tech stack:** Django + DRF + SQLite backend, Pandas for analysis, React.js + Chart.js web frontend, PyQt5 + Matplotlib desktop frontend.

2.2 Tasks Done

The completed solution demonstrates seamless integration between Django backend, React web frontend, and PyQt5 desktop application, and is accessible at the GitHub Repository: https://github.com/Vaibhav-Kumar10/FOSSEE_2026-Web_Based_Application_Screening_Task_Submission

2.2.1 System Architecture

- **Backend:** Django REST Framework handling CSV upload, parsing and analysis with Pandas, and history storage in SQLite.
- **Web Frontend:** React.js with Chart.js, configured against the backend via a REACT_APP_API_URL environment variable.
- **Desktop Frontend:** PyQt5 with Matplotlib, configured against the backend via an APP_URL environment variable.

2.2.2 Features Implemented

All required features were successfully implemented:

- CSV upload supporting the provided sample_equipment_data.csv on both clients.
- Analytics: average Flowrate, Pressure, and Temperature computed by the backend.
- Visualization: bar charts of equipment-type distribution on both clients.
- History: automatic pruning that keeps only the last 5 uploaded datasets.

- Reporting: PDF report generation with embedded charts.

2.2.3 Deployment

The backend and web frontend were deployed together and are reachable at: https://fossee-2026-web-based-application-s.vercel.app. Setup is also scripted locally via build.sh and a requirements.txt for the backend.

2.2.4 Known Limitation

Token-based authentication was implemented but disabled in the final deployment, as Render's free tier uses an ephemeral filesystem that does not reliably persist the SQLite auth store between restarts — a deliberate trade-off documented in the submission's README to keep the public demo stable.

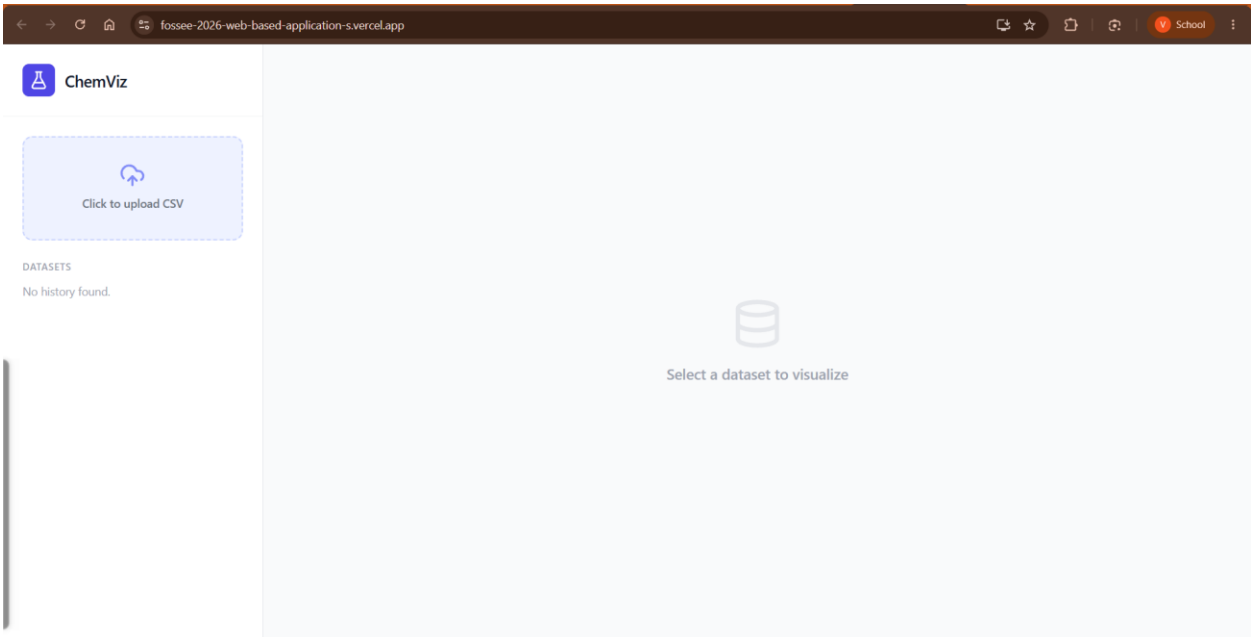

Figure 2.1: Web Application Dashboard

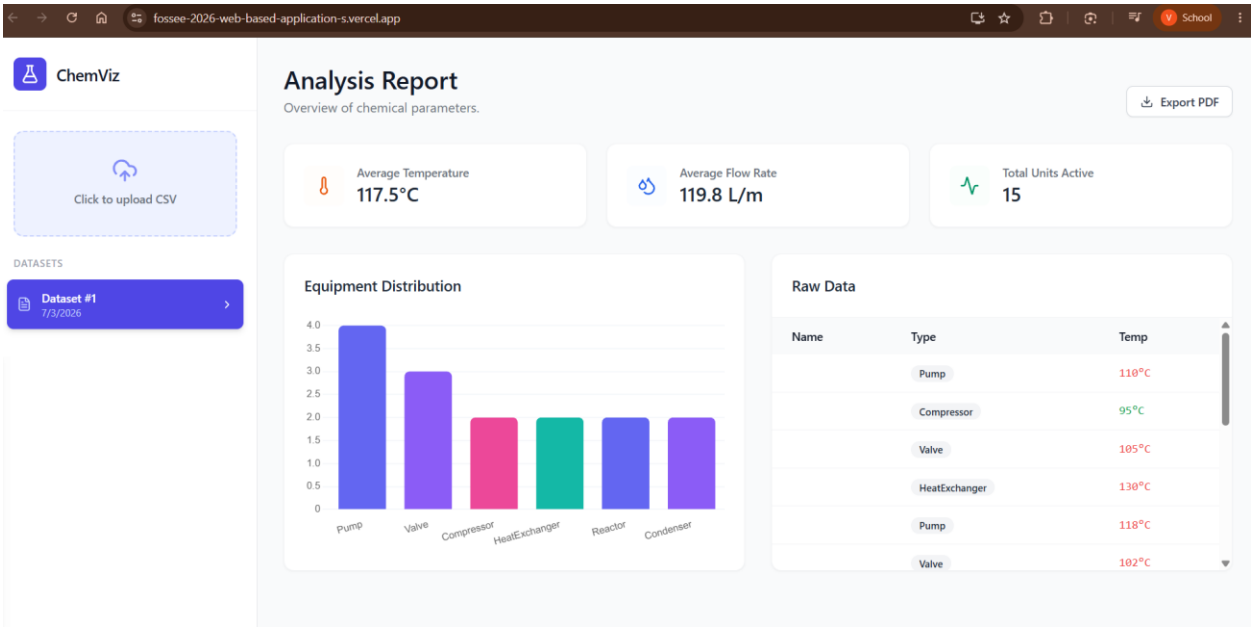

Figure 2.2: Analysis and Visualization Tab

2.2.5 Conclusion

The screening task resulted in a working hybrid application meeting the required feature set, demonstrating working proficiency in Django/DRF, React, PyQt5, and cross-client API integration ahead of the internship proper.

Chapter 3

Task 1: Connection Routing & Pathfinding Engine

3.1 Problem Statement

Early in the internship, connections between components were drawn manually and the resulting lines frequently cut straight through other equipment, or lost their orthogonal shape entirely once a connected component was moved. The mentor asked for connection-drawing to behave like professional diagramming software: a live preview while drawing, automatic obstacle-avoiding orthogonal routing, and correct re-routing whenever a component moves — flagged at one point as a fix needed “ASAP.”

3.2 Tasks Done

This task was completed and iterated on across seven pull requests (#110, #113, #124, #133, #137, #138, #140), progressively replacing the routing engine and finally adding a dedicated test suite.

3.2.1 Live Preview & Smart Routing (PR #110)

Manual waypoint placement was replaced with a live, dotted-path preview while a connection is being dragged, and the underlying router was rebuilt to avoid other components.

- Added a ConnectionPreview component (dotted path + optional arrowhead) for real-time visual feedback.

- Replaced manual waypoint-filtering with a new smartRoute implementation and helpers (getItemRects, segmentHitsRect) that generate obstacle-avoiding bend candidates.
- The router evaluates horizontal, vertical, and dogleg candidate paths and picks the first that does not intersect another component.

3.2.1.1 The Problem:

Drawing connections between components currently requires manually placing waypoints, and the resulting auto-routed lines often cut straight through other equipment on the canvas. This creates messy, overlapping diagrams and forces the user to do the router's job manually.

ScreenShot:

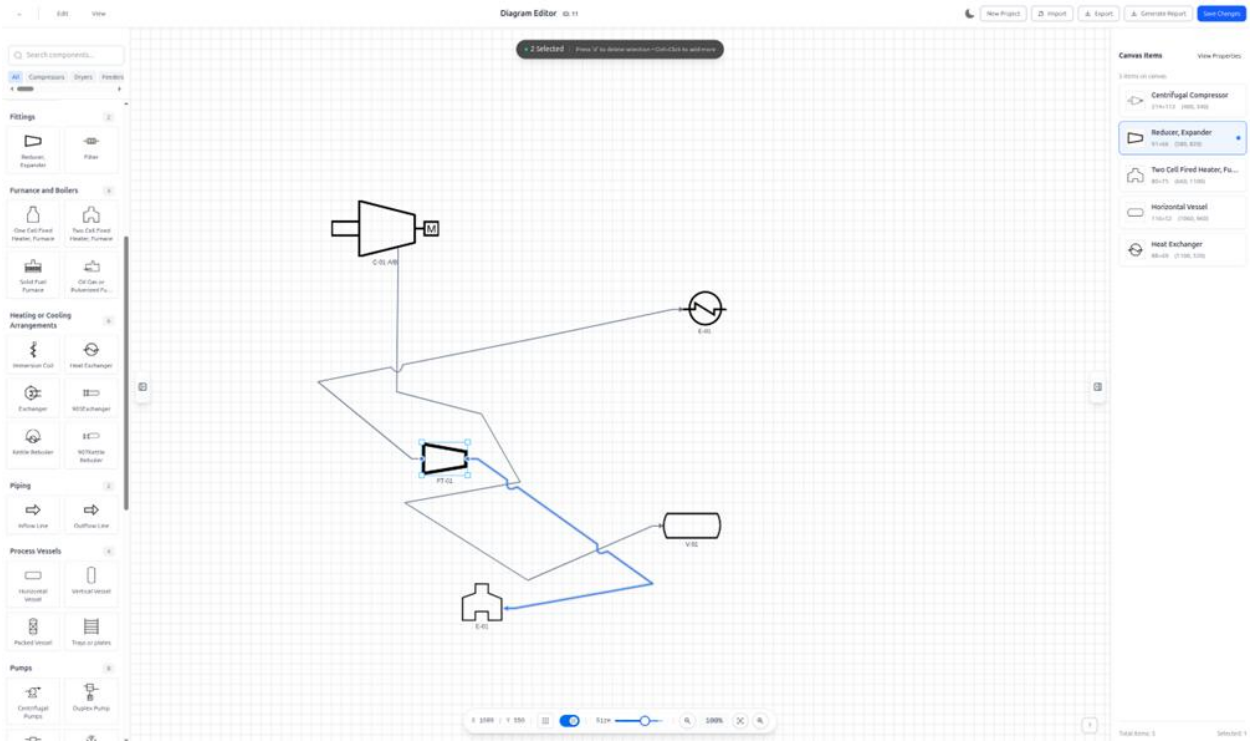

Fig 3.2.1.1: Inconsistent routing lines

3.2.1.2 Expected Behavior:

The connection tool should behave like professional diagramming software (Visio, draw.io):

1. Give a real-time, live preview of the final wire path while dragging.
2. Automatically avoid cutting through components by making smart orthogonal bends (L and Z shapes).
3. Do all of this with zero manual waypoint clicks.

3.2.1.3 Solution:

Introduce a visual connection preview and smarter auto-routing during connection drawing.

- Add ConnectionPreview component (dotted path + optional arrow head) for temporary visual feedback while drawing.
- Integrate preview into Editor: compute previewPathData using calculateManualPathsWithBridges with a fake target, render ConnectionPreview when drawing, and use handleCancelDrawing when clicking empty space (previous behavior of adding a waypoint was removed).
- Replace manual waypoint-filtering logic in routing.ts with a new smartRoute implementation and helpers (getItemRects, segmentHitsRect) that generate obstacle-avoiding bend candidates.
- The router tests candidate orthogonal paths (horizontal, vertical, and dogleg) and picks the one that doesn't intersect other nodes.

These changes improve the user experience when creating connections and make routing more robust against overlapping items.

ScreenShot:

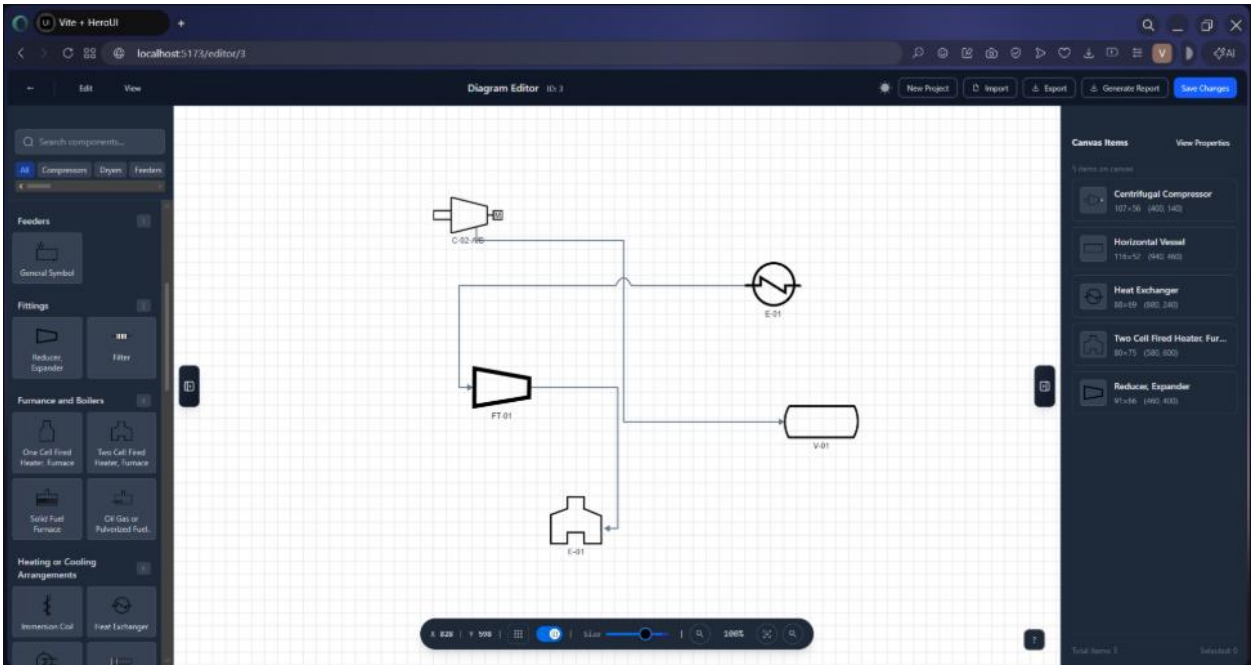

Fig 3.2.1.2: The resolved routing lines

3.2.2 Editable Waypoints (PR #113)

Connections moved from static, auto-routed-only paths to persistently editable geometry.

- Added a waypoints field on Connection to persist wire geometry between sessions.
- New connections still auto-route on creation, but the generated bends are stored so users can drag individual segments afterwards while orthogonal routing is preserved.

ScreenShot:

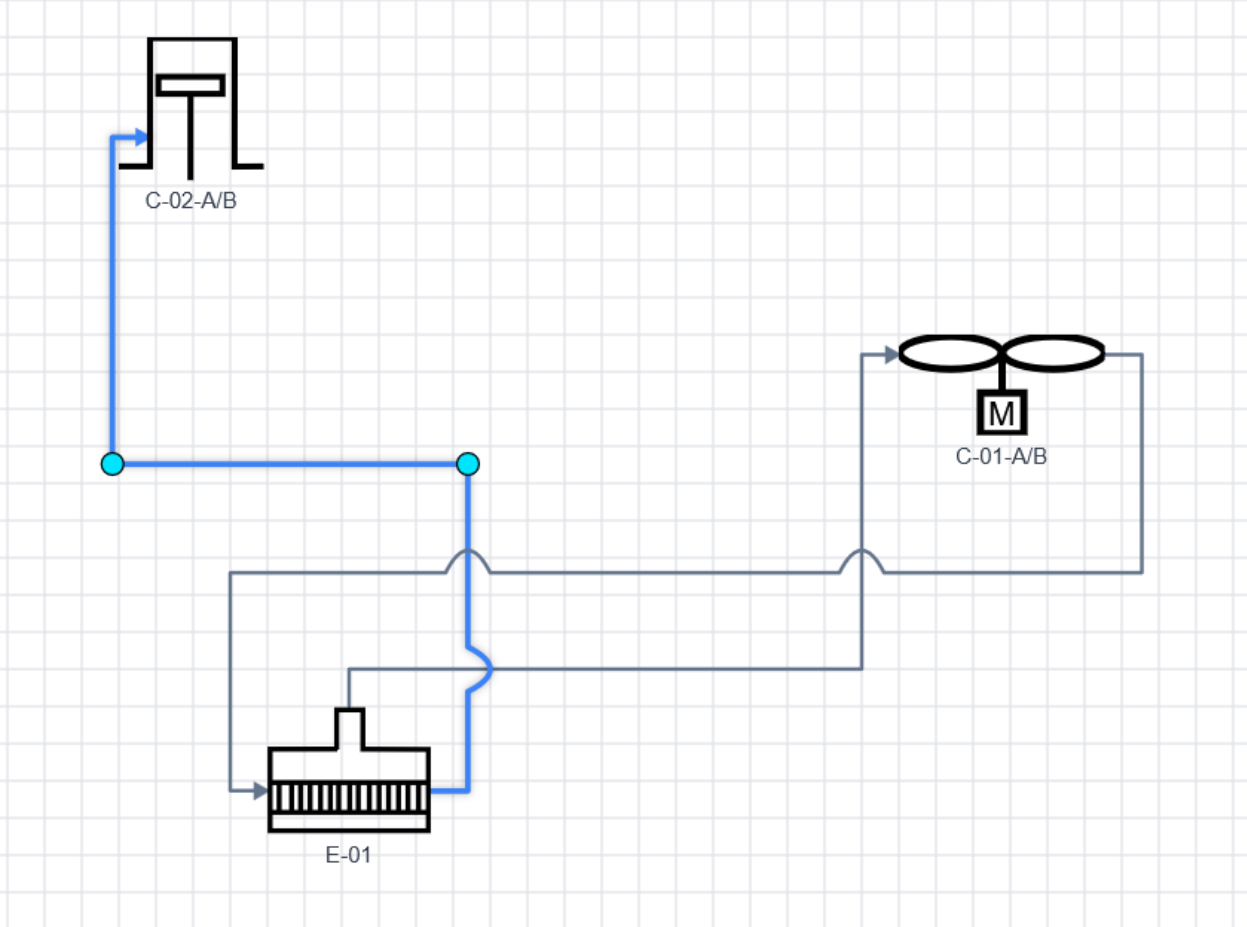

Fig 3.2.2.1: The editable routing lines

3.2.2.1 Code implementation:

```
{/* 4. DRAG HANDLES */}
{isSelected && points && points.map((p, i) => (
 <Circle
  key={i}
  x={p.x}
  y={p.y}
  radius={6}
  fill="#00e5ff"
  stroke="#000"
```

```
        strokeWidth={1}
        draggable
        onDragMove={(e) => {
          const dx = Math.abs(e.target.x() - p.x);
          const dy = Math.abs(e.target.y() - p.y);

          // Lock axis based on which way they drag more for clean 90-degree wires
          if (dx > dy) {
            e.target.y(p.y);
          } else {
            e.target.x(p.x);
          }

          onWaypointDrag?.(i, {
            x: e.target.x(),
            y: e.target.y()
          });
        }}
        onMouseEnter={(e) => {
          const stage = e.target.getStage();
          if (stage) stage.container().style.cursor = "move";
        }}
        onMouseLeave={(e) => {
          const stage = e.target.getStage();
          if (stage) stage.container().style.cursor = "default";
        }}
      />
    ))}
```

3.2.3 A* Pathfinding (PR #124)

The heuristic router was replaced with a proper A* search.

- Implemented grid-based A* with a Manhattan heuristic for obstacle-avoiding paths.
- Blocked grid cells are generated automatically from component bounding boxes.

ScreenShot:

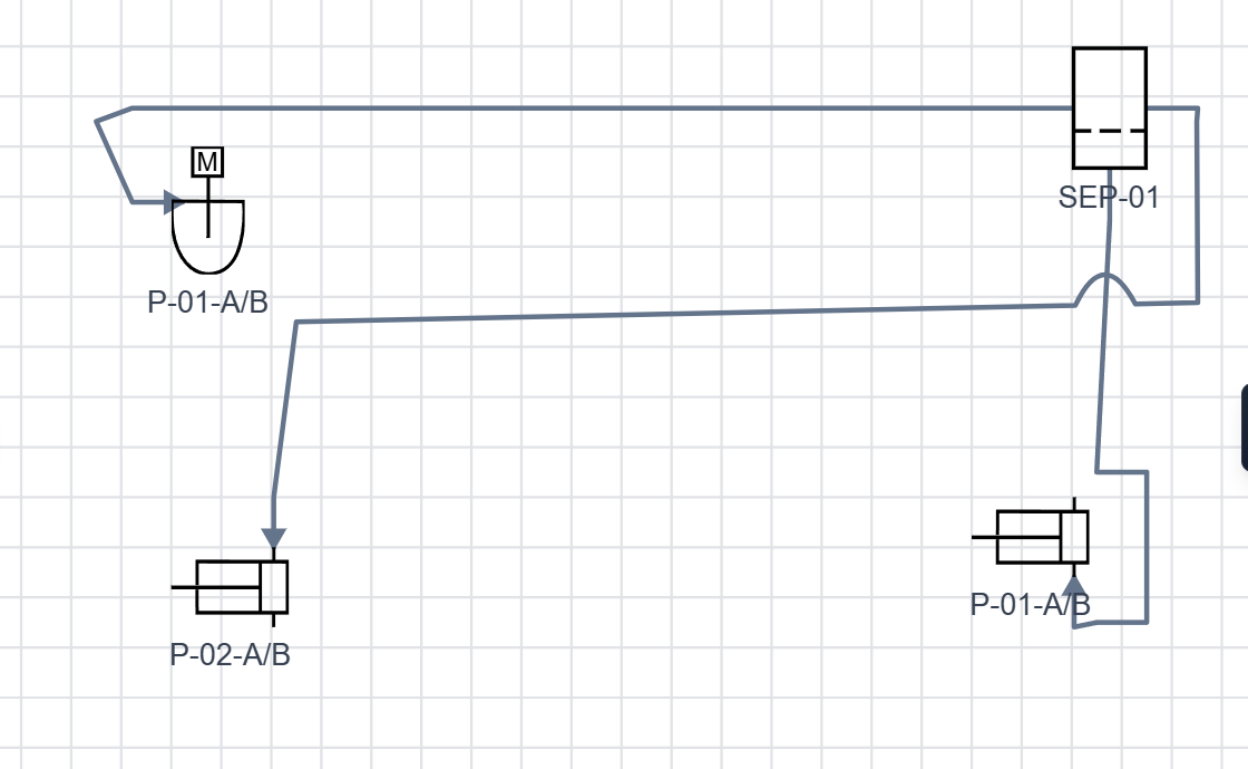

Fig 3.2.3: Routing lines using A* algorithm

3.2.3.1 Code implementation:

```
/ src/utils/pathfinding/aStar.ts
import { GridPoint, PathResult, AStarNode } from "./types";
```

```
/**
 * Priority queue implementation for A* algorithm
 */
class PriorityQueue<T> {
  private items: { item: T; priority: number }[] = [];

  enqueue(item: T, priority: number): void {
    this.items.push({ item, priority });
    this.items.sort((a, b) => a.priority - b.priority);
  }

  dequeue(): T | undefined {
    return this.items.shift()?.item;
  }

  isEmpty(): boolean {
    return this.items.length === 0;
  }

  contains(item: T, comparator: (a: T, b: T) => boolean): boolean {
    return this.items.some(({ item: queueItem }) =>
      comparator(item, queueItem),
    );
  }
  find(comparator: (item: T) => boolean): T | undefined {
    return this.items.find(({ item }) => comparator(item))?.item;
  }
}
```

```
/**
 * Manhattan distance heuristic for orthogonal movement
 */
function heuristic(a: GridPoint, b: GridPoint): number {
  return Math.abs(a.x - b.x) + Math.abs(a.y - b.y);
}

/**
 * Get neighboring grid points (4-directional orthogonal movement)
 */
function getNeighbors(
  point: GridPoint,
  grid: boolean[][],
  bounds: { width: number; height: number },
): GridPoint[] {
  const neighbors: GridPoint[] = [];
  const directions = [
    { x: 1, y: 0 }, // Right
    { x: -1, y: 0 }, // Left
    { x: 0, y: 1 }, // Down
    { x: 0, y: -1 }, // Up
  ];
  for (const dir of directions) {
    const nx = point.x + dir.x;
    const ny = point.y + dir.y;

    // Check bounds
```

```
    if (nx >= 0 && nx < bounds.width && ny >= 0 && ny < bounds.height) {
      // Check if cell is not blocked
      if (!grid[ny][nx]) {
        neighbors.push({ x: nx, y: ny });
      }
    }
  }
  return neighbors;
}

/**
 * Check if two grid points are equal
 */
function pointsEqual(a: GridPoint, b: GridPoint): boolean {
  return a.x === b.x && a.y === b.y;
}

/**
 * A* pathfinding algorithm for orthogonal movement
 */
export function aStar(
  start: GridPoint,
  goal: GridPoint,
  grid: boolean[][],
  bounds: { width: number; height: number },
): PathResult {
  const openSet = new PriorityQueue<AStarNode>();
  const closedSet = new Set<string>();
```

```
// Key function for tracking visited nodes
const key = (point: GridPoint) => `${point.x},${point.y}`;

// Initialize start node
const startNode: AStarNode = {
  point: start,
  g: 0,
  h: heuristic(start, goal),
  f: heuristic(start, goal),
  parent: null,
};

openSet.enqueue(startNode, startNode.f);
while (!openSet.isEmpty()) {
  const current = openSet.dequeue()!;

  // Check if we reached the goal
  if (pointsEqual(current.point, goal)) {
    // Reconstruct path
    const path: GridPoint[] = [];
    let node: AStarNode | null = current;

    while (node) {
      path.unshift(node.point);
      node = node.parent;
    }
```

```
  return { path, found: true };
}

const currentKey = key(current.point);
if (closedSet.has(currentKey)) {
  continue;
}
closedSet.add(currentKey);

// Explore neighbors
const neighbors = getNeighbors(current.point, grid, bounds);

for (const neighbor of neighbors) {
  const neighborKey = key(neighbor);

  if (closedSet.has(neighborKey)) {
    continue;
  }
  const g = current.g + 1; // Orthogonal movement cost = 1
  const h = heuristic(neighbor, goal);
  const f = g + h;
  const neighborNode: AStarNode = {
    point: neighbor,
    g,
    h,
    f,
    parent: current,
  };
```

```
      // Check if this neighbor is already in open set with better cost
      const existingNode = openSet.find((node) =>
        pointsEqual(node.point, neighbor),
      );

      if (!existingNode || g < existingNode.g) {
        openSet.enqueue(neighborNode, f);
      }
    }
  }
  // No path found
  return { path: [], found: false };
}
```

3.2.4 Spatial-Graph A* Attempt (PR #133 — superseded)

An intermediate attempt replaced the grid-based A* with a spatial-graph-based A* (generateSpatialGraph, direction-aware aStarSpatial with a turn penalty) and added draggable-segment UI support. It was closed without merging once the approach below (PR #137) carried the idea to completion three days later.

3.2.5 Obstacle Avoidance & Turn-Restriction Pipeline (PR #137)

This is the direct fix for the mentor's priority bug report: whenever a component moves, connected edges must drop their old path, re-run routing, and produce a new, strictly orthogonal path.

- Established a fixed routing pipeline (see below), with a hardcoded 2-turn cap in the A* search.

- Added stricter obstacle-intersection checks, wired into both initial routing and later segment editing, so a connection can no longer be drawn over a component.

A* (max 2 turns hardcoded) | optimizePath() -- remove collinear points | enforceManhattanShape() -- collapse to L/straight | STORE PATH (waypoints = single source of truth) | render segments | drag -> findSegmentIndex -> moveSegment -> snap -> normalize -> update

The finalised routing pipeline established in PR #137.

3.2.5.1 Code implementation:

```
/**
 * Check if an orthogonal line segment intersects with any obstacle
 * Uses stricter orthogonal check instead of coarse bounding box overlap
 */
export function orthogonalSegmentHitsObstacle(
  p1: Point,
  p2: Point,
  obstacles: Rect[]
): boolean {
  for (const r of obstacles) {
    if (Math.abs(p1.x - p2.x) < 1) { // practically vertical
      const x = p1.x;
      const minY = Math.min(p1.y, p2.y);
      const maxY = Math.max(p1.y, p2.y);
      // Strict intersection: x is inside horizontally, and vertical line spans the rect y bounds
      if (
        x > r.x &&
```

```
      x < r.x + r.width &&
      maxY > r.y &&
      minY < r.y + r.height
    ) {
      return true;
    }
  } else {
    // horizontal line
    const y = p1.y;
    const minX = Math.min(p1.x, p2.x);
    const maxX = Math.max(p1.x, p2.x);
    if (
      y > r.y &&
      y < r.y + r.height &&
      maxX > r.x &&
      minX < r.x + r.width
    ) {
      return true;
    }
  }
 }
 return false;
}

/**
 * Validate an entire path against obstacles
 */
```

```
export function pathHitsObstacle(path: Point[], obstacles: Rect[]): boolean {
  for (let i = 0; i < path.length - 1; i++) {
    if (orthogonalSegmentHitsObstacle(path[i], path[i + 1], obstacles)) {
      return true;
    }
  }
  return false;
}
```

3.2.6 Import Cleanup (PR #138)

A quick clean-up removing duplicate/unused imports in Editor.tsx (calculateManualPathsWithBridges, smartRoute, getGripPosition, getStandoff, and a duplicate useComponents import) left behind by the routing rewrite.

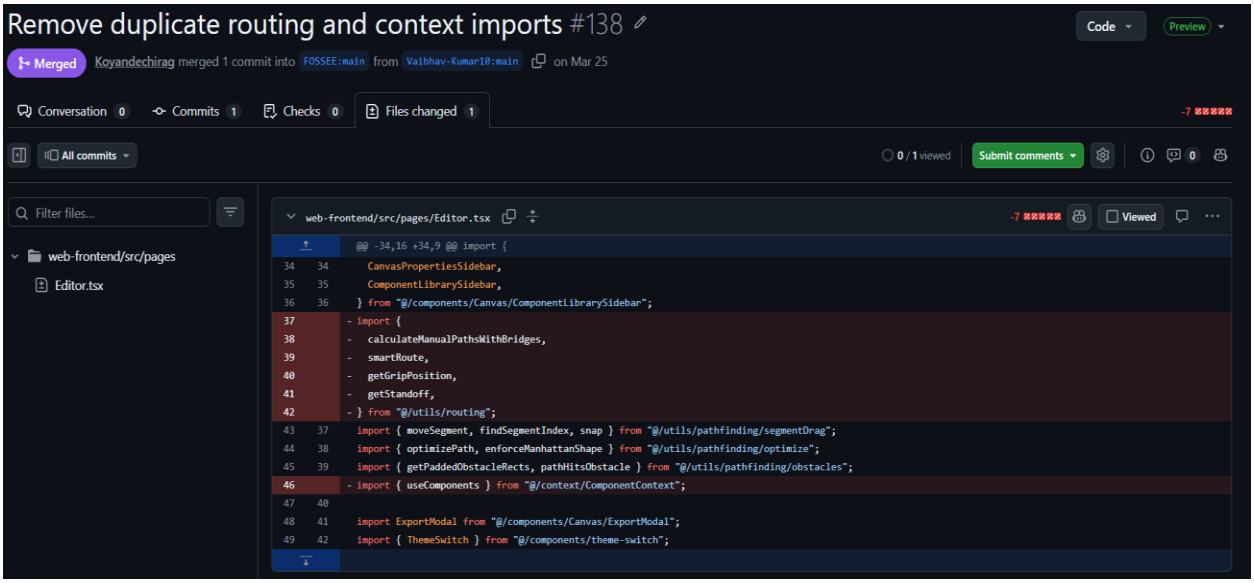

Fig. 3.2.6.1: Screenshot of GitHub commit.

3.2.7 Routing Test Suite (PR #140)

The first dedicated unit tests for the routing engine, added under web-frontend/tests/routing/: **connectionLine.test.tsx**, **obstacles.test.ts**, **routing.test.ts**, and **segmentDrag.test.ts**, with mocks for react-konva and layout, with more than 300 lines of code for each test file. The tests cover dragBoundFunc kinematics, padded obstacle rectangles, collision detection, smartRoute heuristics, standoff/grip math, and manual path generation.

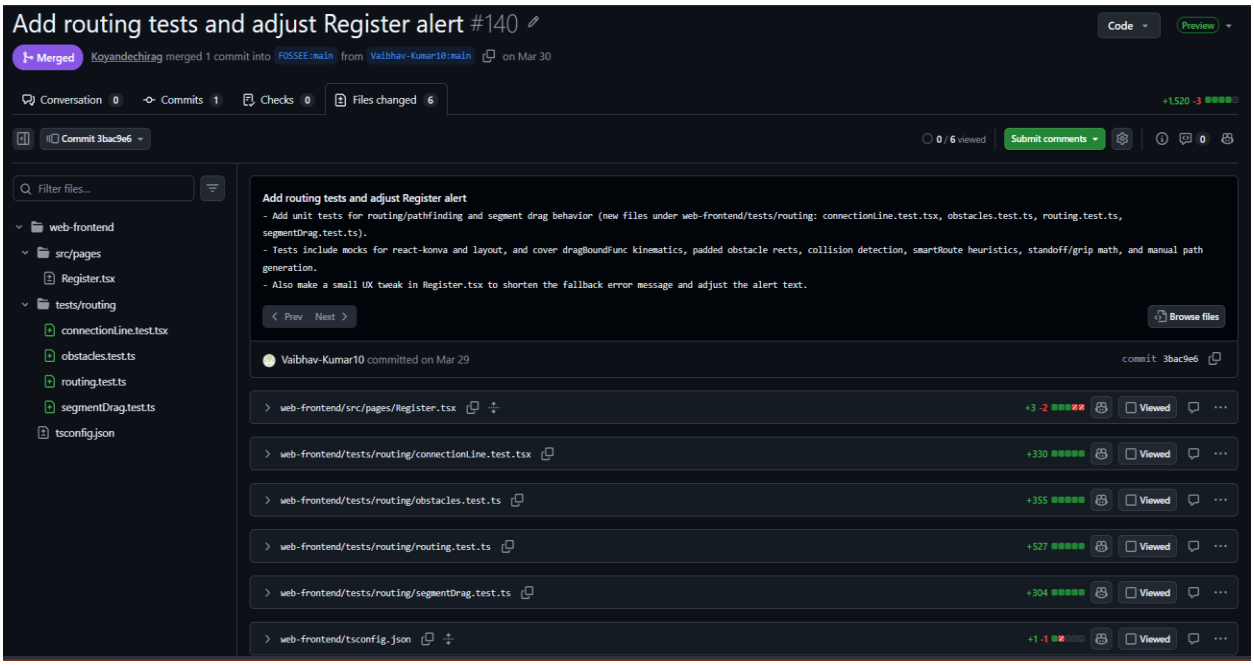

Fig. 3.2.7.1: Screenshot of GitHub commit.

3.3 Outcome

The routing engine moved from manual waypoint placement, through three successive pathfinding strategies, to a single documented pipeline with **strict obstacle avoidance,** a hardcoded **turn limit**, and its own **test suite** — closing out what had been flagged as the most urgent bug of the early internship.

Chapter 4

Task 2: “Add New Component” Modal — Help & UX

4.1 Problem Statement

The “Add New Component” modal, used to register new symbols into the library, needed an in-context help section explaining the required PNG/SVG assets, field meanings, and the auto-generated label format, so contributors did not have to guess the correct inputs.

4.2 Tasks Done

Pull Requests: #145

Delivered in PR #145, merged exactly on the deadline set by the mentor.

- Added an inline help popover using Popover / PopoverTrigger / PopoverContent, with a new isHelpOpen UI state.
- Updated the ModalHeader layout with an info (**i**) button that opens a “Component Requirements Guide” covering file requirements (PNG for the toolbar, SVG for canvas rendering), the Legend-Count-Suffix label format, and grips/connectors.
- Set the modal to isDismissable=false so the help panel cannot be closed accidentally mid-read.

4.2.1 Code Implementation

The following code snippet illustrates the intened logic used in the dashboard:

```
<ModalHeader className="flex justify-between items-center w-full pr-8">
  <div className="flex flex-col gap-1">
    {editingComponent
    ? `Edit ${editingComponent.name}`
    : "Add New Component"}
  </div>
  <Popover isOpen={isHelpOpen} onOpenChange={setIsHelpOpen}
placement="bottom-end">
    <PopoverTrigger>
      <Button isIconOnly variant="light" className="text-gray-500 hover:text-
primary transition-colors text-xl"
        aria-label="Toggle Component Help">
        i
      </Button>
    </PopoverTrigger>
    <PopoverContent className="w-[500px]">
      <div className="p-4 bg-white dark:bg-gray-900 rounded-xl text-sm
relative">
        <Button isIconOnly size="sm" variant="light"
          className="absolute top-2 right-2 text-gray-400 hover:text-gray-600
dark:hover:text-gray-200"
          onPress={()=> setIsHelpOpen(false)}
          >
          ×
        </Button>
        <h4
          className="font-semibold text-blue-800 dark:text-blue-300 mb-4 flex
items-center gap-2 text-base w-11/12">
          Component Requirements Guide
        </h4>
        <div
          className="grid grid-cols-1 md:grid-cols-2 gap-x-6 gap-y-4 text-
gray-600 dark:text-gray-300 leading-relaxed">
          <div>
            <strong className="block text-gray-800 dark:text-gray-100 mb-
1">File Requirements</strong>
            <ul className="list-disc pl-4 space-y-1 text-xs">
              <li><b>PNG:</b> Displayed as the thumbnail in sidebars.</li>
              <li><b>SVG:</b> Rendered on the piping canvas.</li>
            </ul>
```

```
                </div>
                <div>
                  <strong className="block text-gray-800 dark:text-gray-100 mb-
1">Label Generation</strong>
                  <p className="text-xs">Format: <code>Legend-Count-
Suffix</code> (e.g., <code>P-01-A</code>)
                    representing type and chronological order.</p>
                </div>
                <div>
                  <strong className="block text-gray-800 dark:text-gray-100 mb-
1">Field Definitions</strong>
                  <ul className="list-disc pl-4 space-y-1 text-xs">
                    <li><b>Legend:</b> The base prefix (e.g., 'HX').</li>
                    <li><b>Suffix:</b> An optional trailing specifier (e.g., 'A').</li>
                  </ul>
                </div>
                <div>
                  <strong className="block text-gray-800 dark:text-gray-100 mb-
1">Grips & Connectors</strong>
                  <p className="text-xs">Anchor points that snap pipes and lines to
the boundaries of this
                    component.</p>
                </div>
              </div>
            </div>
        </PopoverContent>
    </Popover>
</ModalHeader>
```

4.3 Outcome

New and existing contributors adding components to the library now have a self-contained reference inside the modal itself, reducing back-and-forth about file formats and label conventions.

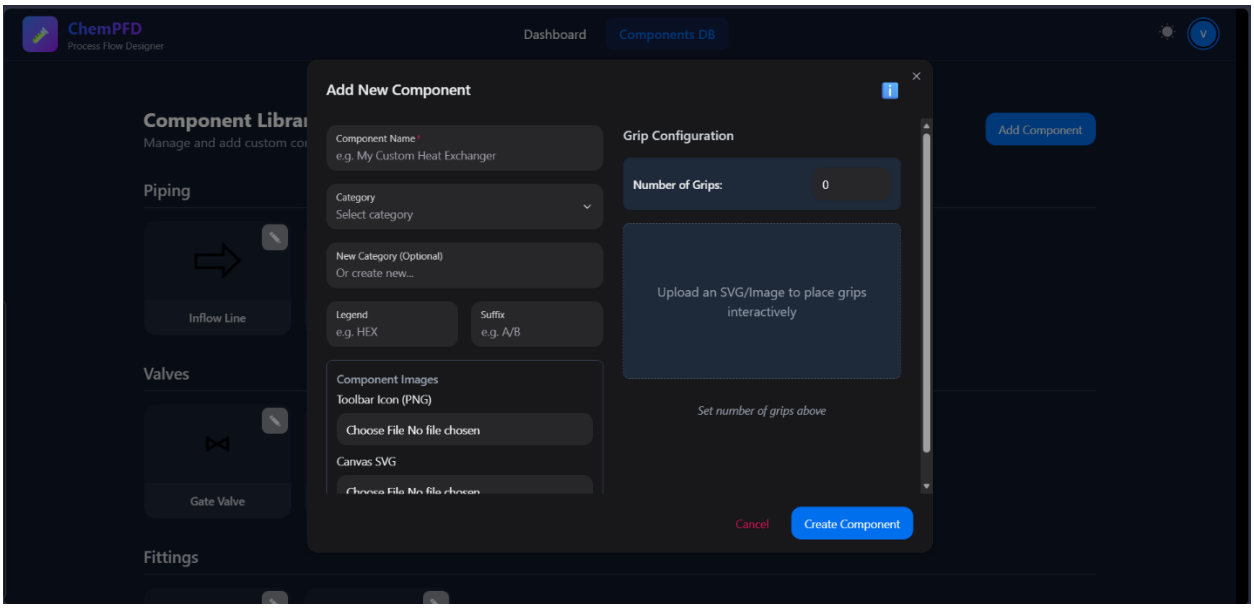

Figure 4.1: Add New Component Interface

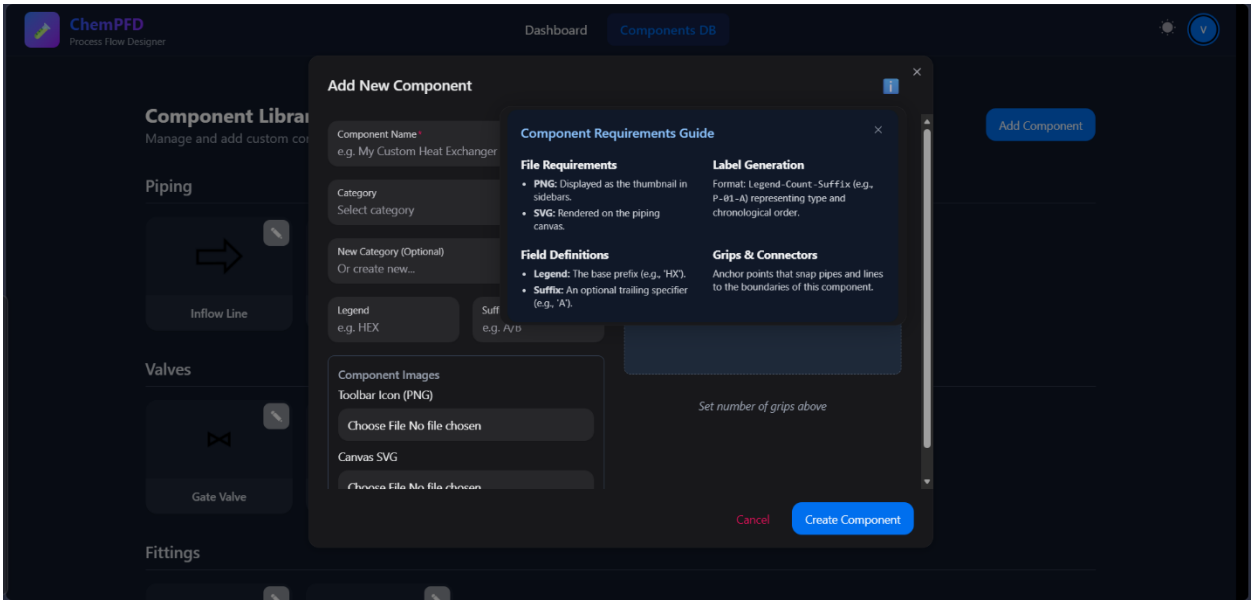

Figure 4.2: Help Interface

Chapter 5

Task 3: AI-Powered Diagram Generation

5.1 Problem Statement

The team was asked to let a user describe a process in plain text (for example, “Pump → Heat Exchanger → Tank”) and have the editor generate the corresponding diagram automatically, including correctly typed components and their connections.

5.2 Tasks Done

Pull Requests: #151, #152, #164

5.2.1 Gemini Backend Integration (PR #151)

- Added backend/core/gemini_service.py, calling Google's google-genai SDK to convert a prompt into a strict JSON schema of components (pump, tank, valve, heat_exchanger) and their connections.
- Exposed POST /api/ai-generate/ with input validation and error handling, and added GEMINI_API_KEY configuration.

5.2.1.1 Code Implementation

backend/api/views.py

```
# ============= AI Endpoints =============
@api_view(["POST"])
@permission_classes([AllowAny]) # No login required for now to facilitate easy frontend testing
```

```
def ai_generate(request):
    """
    POST /api/ai-generate/
    Accepts: { "prompt": "Pump connected to tank..." }
    Returns: Structured JSON for the Canvas UI or an Error message.
    """
    prompt = request.data.get("prompt")

    # 1. Basic validation
    if not prompt or not isinstance(prompt, str):
        return Response(
            {"error": "Prompt string is required."},
            status=status.HTTP_400_BAD_REQUEST
        )

    try:
        # 2. Call the dedicated Gemini service
        result = generate_diagram(prompt)

        # 3. Handle graceful LLM rejections
        if "error" in result:
            return Response(result, status=status.HTTP_400_BAD_REQUEST)

        # 4. Return success data
        return Response(result, status=status.HTTP_200_OK)

    except ValueError as ve:
        # Configuration errors (like missing API key)
        return Response(
            {"error": str(ve)},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )
```

```
    except Exception as e:
        # General LLM or runtime errors
        return Response(
            {"error": str(e)},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )
```

backend/core/gemini_service.py

```
import json
from google import genai
from google.genai import types
from django.conf import settings

def generate_diagram(user_input: str) -> dict:
    """
    Takes a natural language prompt and returns a structured JSON dictionary
    representing the components and connections for a Chemical PFD.
    """
    # 1. Validate API Key
    api_key = getattr(settings, "GEMINI_API_KEY", None)
    if not api_key:
        raise ValueError("LLM API key is not configured in settings/environment variables.")

    client = genai.Client(api_key=api_key)

    # 2. Strict System Prompt combining original requirements and Antigravity's instructions
    system_prompt = """
    You are an expert chemical engineering assistant. Your task is to process user descriptions
    of chemical process flow diagrams and return a structured JSON representation.
```

```
    Extract the components and the connections between them.

    STRICT RULES:
    - Use ONLY these component types: pump, tank, valve, heat_exchanger.
    - If a user mentions a synonym (like 'exchanger'), map it to 'heat_exchanger'.

    The JSON MUST match this exact schema:
    {
      "components": [
        { "type": "pump | tank | valve | heat_exchanger", "id": "string (unique string/uuid)", "label": "string" }
      ],
      "connections": [
        { "from": "id1", "to": "id2" }
      ]
    }

    If the user input is completely unrelated to chemical components or process flows, return exactly this JSON:
    { "error": "Invalid input. Please describe a process flow involving components like pumps, tanks, valves, etc." }
    """

    try:
        # 3. Use gemini-1.5-pro with native JSON formatting
        # 3. Use gemini-3-flash-preview with native JSON formatting
        response = client.models.generate_content(
            model="gemini-3-flash-preview",
            # model="gemini-1.5-pro",
            contents=user_input,
            config=types.GenerateContentConfig(
                system_instruction=system_prompt,
```

```
            response_mime_type="application/json"
        )
    )

    # 4. Generate content
    output_text = response.text.strip()

    # 5. Parse and return the guaranteed JSON
    parsed_data = json.loads(output_text)
    return parsed_data

except json.JSONDecodeError:
    raise RuntimeError("LLM returned malformed JSON.")
except Exception as e:
    raise RuntimeError(f"LLM API Error: {str(e)}")
```

5.2.2 Wiring Generation into the Editor (PR #152)

- Added a controlled aiPrompt textarea and a modal action to trigger generation, with loading and error states.
- On success, mapped AI-returned components to existing component definitions, added them to the editor store at spaced-out positions, and created connections between the newly added items.

5.2.3 Robustness & Whitelisting (PR #164)

Testing surfaced three problems: limited component variety (the model defaulted to only 3–4 types), no variant selection within a component group, and incomplete connections once three or more components were requested.

- Accepted an available_components whitelist end-to-end: the frontend now collects and sends the currently available component set, and the backend forwards it to gemini_service.
- The backend builds a stricter system prompt around the whitelist, falls back to sensible defaults, and validates the returned JSON (component ids/types and connection references) before returning structured errors.

5.2.3.1 System prompt used in backend/core/gemini_service.py

```
# 2. SYSTEM PROMPT
system_prompt = f"""
  You are a system that converts process descriptions into STRICT JSON.

  RULES:
  1. Use ONLY these component types exactly:
  {components_str}

  2. DO NOT use variations or synonyms.

  3. Generate EXACTLY the number of UNIQUE components implied by the input. If a component type is mentioned again (e.g. "tank connected back to pump"), DO NOT create a duplicate component unless explicitly requested. Reuse the existing component's ID to create a feedback loop or complex connection.

  4. Each component must have:
  - id (string, e.g. "c1", "c2"...)
  - type (string, from allowed list)
  - label (string)
  - x (integer, horizontal placement. VARY THIS.)
  - y (integer, vertical placement. CRITICAL: You MUST use 2D space. DO NOT place all components on the same Y-axis. Arrange them in a cycle, zig-zag, or branched pattern by mixing Y values like 100, 300, 500.)
```

```
    5. Connections must accurately represent the flow. Feedback loops and multiple connections to the same component are allowed.

    6. Every component must be connected. Do NOT skip components.

    7. Output STRICT JSON only.

    FORMAT:
    {{
      "components": [
        {{ "id": "c1", "type": "tank", "label": "Main Tank", "x": 100, "y": 300 }},
        {{ "id": "c2", "type": "pump", "label": "Pump", "x": 300, "y": 100 }}
      ],
      "connections": [
        {{ "from": "c1", "to": "c2" }}
      ]
    }}

    ERROR:
    If invalid input:
    {{ "error": "Invalid process description" }}
  """
```

5.3 Outcome

Diagram generation moved from a single-shot integration to a validated pipeline that reliably draws from the full component catalogue and produces complete connection sets, closing out repeated rounds of mentor-reported generation bugs.

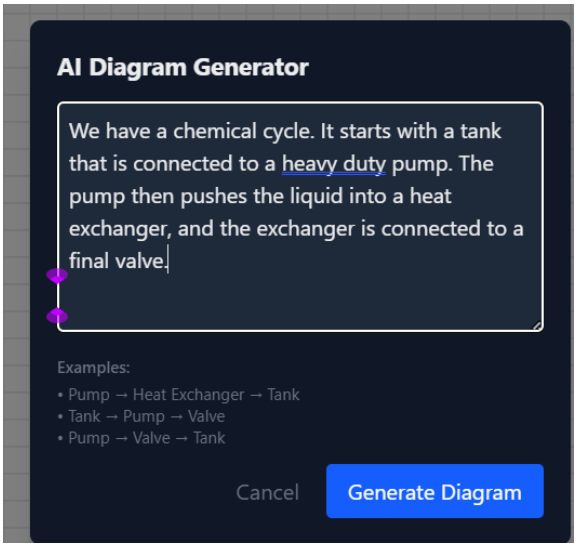

Fig. 5.1: AI prompt text area

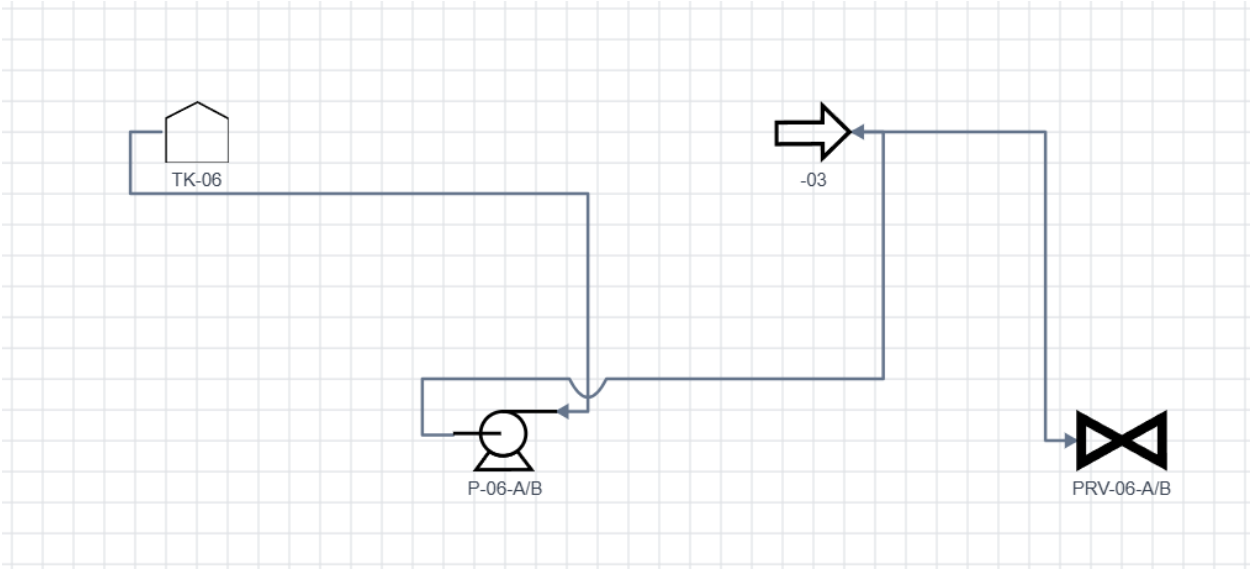

Fig. 5.2: Generated output diagram

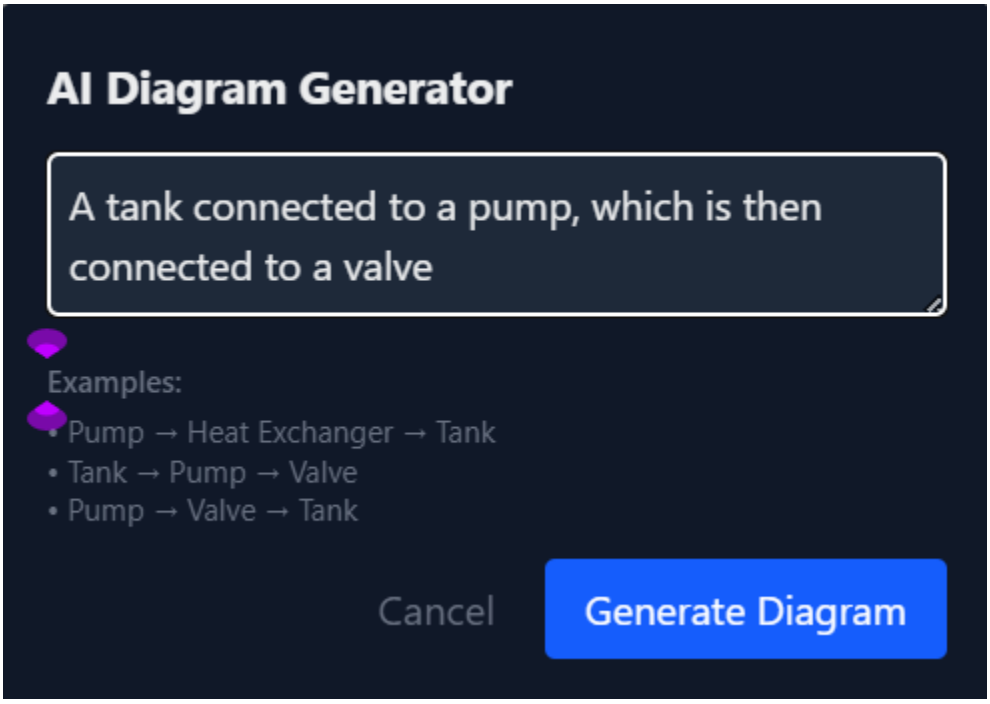

Fig. 5.3: Another prompt

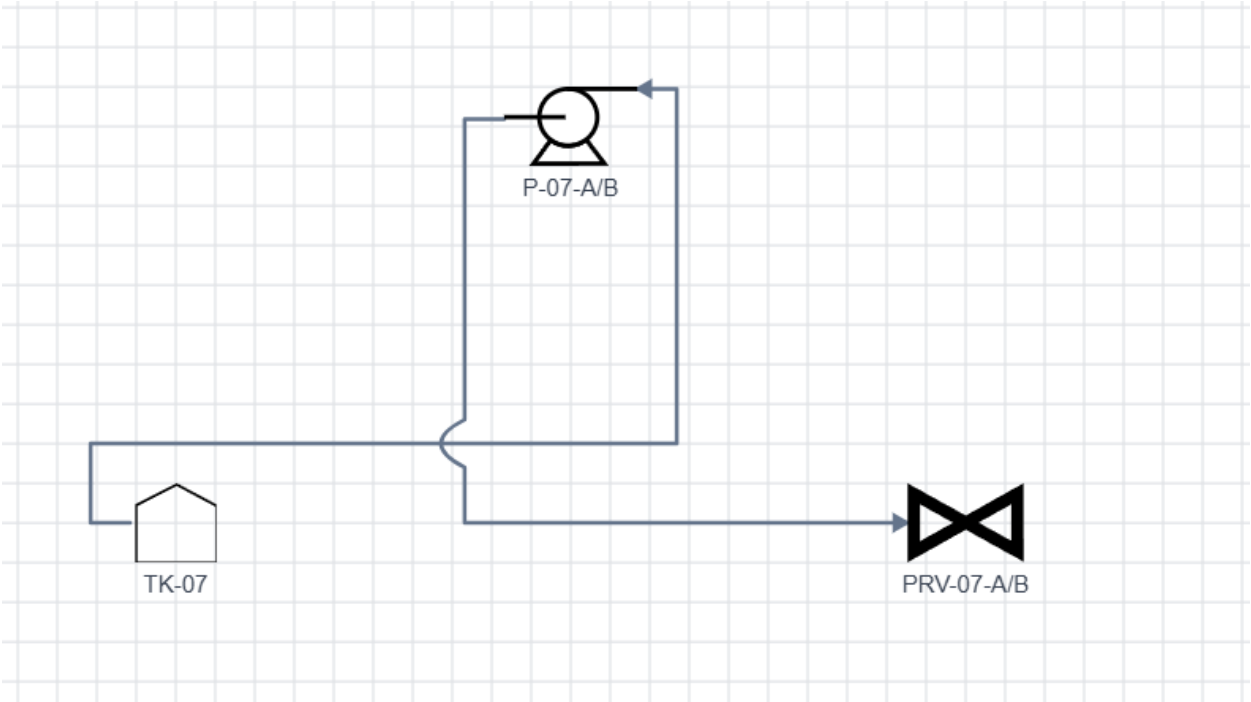

Fig. 5.4: Generated output diagram

Chapter 6

Task 4: Export Footer & Zoom-Aware Resizing

6.1 Problem Statement

Two separate requests landed around the same time: exported PDFs/images/reports needed a footer identifying the project, author, and date; and the existing resize handles behaved incorrectly at non-default zoom levels and generated many small, hard-to-undo history entries.

6.2 Tasks Done

Pull Requests: #169, #170

6.2.1 Export Footer / Title Block (PR #169)

- Added web-frontend/src/utils/exportFooter.ts with appendFooterToImage (draws a project/title block onto exported raster images) and getReportFooterHTML (an HTML title block for reports).
- ExportReportModal now accepts a projectName prop and injects the footer into generated report markup; the Editor's image-export path calls appendFooterToImage so PDF/PNG exports carry the same footer.

6.2.1.1 Code Implementation

```
    const projectName = projectMetadata?.name || "Untitled Project";
    const createdBy = localStorage.getItem("username") || "Unknown User";
    const exportDate = new Date().toLocaleDateString("en-US", { month: "long",
day: "numeric", year: "numeric" });
```

```
    dataUrl = await appendFooterToImage(dataUrl, projectName, createdBy, exportDate, backgroundFill, pixelRatio);
```

web-frontend/src/utils/exportFooter.ts

```
export async function appendFooterToImage(
   dataUrl: string,
   projectName: string,
   createdBy: string,
   date: string,
   backgroundColor: string,
   scale: number = 2
): Promise<string> {
   return new Promise((resolve, reject) => {
      const img = new Image();

      img.onload = () => {
         const canvas = document.createElement("canvas");
         const ctx = canvas.getContext("2d");

         if (!ctx) {
            return reject(new Error("Failed to get 2d context"));
         }

         const blockWidth = Math.min(360 * scale, Math.max(250 * scale, img.width * 0.34));
         const blockHeight = 92 * scale;
         const margin = 24 * scale;

         // Keep original image dimensions; draw footer inside the existing bounds
         canvas.width = img.width;
         canvas.height = img.height;
```

```
        // Fill background
        ctx.fillStyle = backgroundColor;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Draw original diagram image
        ctx.drawImage(img, 0, 0);

        // Calculate position for the footer block (bottom-right of existing image bounds)
        const x = canvas.width - blockWidth - margin;
        const y = canvas.height - blockHeight - margin;

        // Draw footer block background and border
        ctx.fillStyle = "#FFF8F2";
        ctx.strokeStyle = "#C97B5A";
        ctx.lineWidth = 1.5 * scale;

        // Rounded rectangle
        ctx.beginPath();
        ctx.roundRect(x, y, blockWidth, blockHeight, 8 * scale);
        ctx.fill();
        ctx.stroke();

        // Text setup
        ctx.fillStyle = "#6B4A3B";
        const labelFontSize = 11 * scale;
        const valueFontSize = 12 * scale;
        const lineGap = 24 * scale;
        const textX = x + 16 * scale;
        const labelY = y + 26 * scale;
```

```
        const rows = [
            { label: "Project Name:", value: projectName },
            { label: "Created By:", value: createdBy },
            { label: "Date:", value: date },
        ];

        rows.forEach((row, index) => {
            const rowY = labelY + index * lineGap;

            ctx.font = `bold ${labelFontSize}px sans-serif`;
            ctx.fillText(row.label, textX, rowY);

            ctx.font = `normal ${valueFontSize}px sans-serif`;
            ctx.fillText(row.value, textX + 110 * scale, rowY);
        });

        resolve(canvas.toDataURL("image/png", 1.0));
      };
      img.onerror = reject;
      img.src = dataUrl;
    });
}

export function getReportFooterHTML(
    projectName: string,
    createdBy: string,
    date: string
): string {
    return `
    <div class="title-block" style="
      width: 320px;
      background-color: #FFF8F2;
```

```
    border: 2px solid #C97B5A;
    border-radius: 8px;
    padding: 14px 16px;
    margin-left: auto;
    margin-top: 40px;
    page-break-inside: avoid;
    box-sizing: border-box;
  ">
    <table style="width: 100%; border-collapse: collapse; border: none;">
      <tr style="border: none;">
        <td style="padding: 5px 0; font-weight: bold; color: #6B4A3B; font-size: 10pt; width: 110px; white-space: nowrap; border: none;">Project Name:</td>
        <td style="padding: 5px 0; color: #6B4A3B; font-size: 10pt; border: none;">${projectName}</td>
      </tr>
      <tr style="border: none;">
        <td style="padding: 5px 0; font-weight: bold; color: #6B4A3B; font-size: 10pt; width: 110px; white-space: nowrap; border: none;">Created By:</td>
        <td style="padding: 5px 0; color: #6B4A3B; font-size: 10pt; border: none;">${createdBy}</td>
      </tr>
      <tr style="border: none;">
        <td style="padding: 5px 0; font-weight: bold; color: #6B4A3B; font-size: 10pt; width: 110px; white-space: nowrap; border: none;">Date:</td>
        <td style="padding: 5px 0; color: #6B4A3B; font-size: 10pt; border: none;">${date}</td>
      </tr>
    </table>
  </div>
 `;
}
```

Screenshots

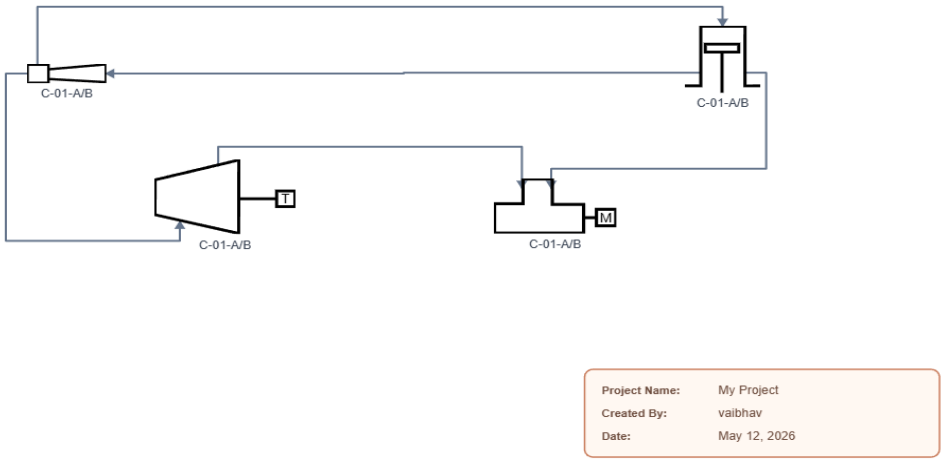

Fig. 6.2.1: Export sample

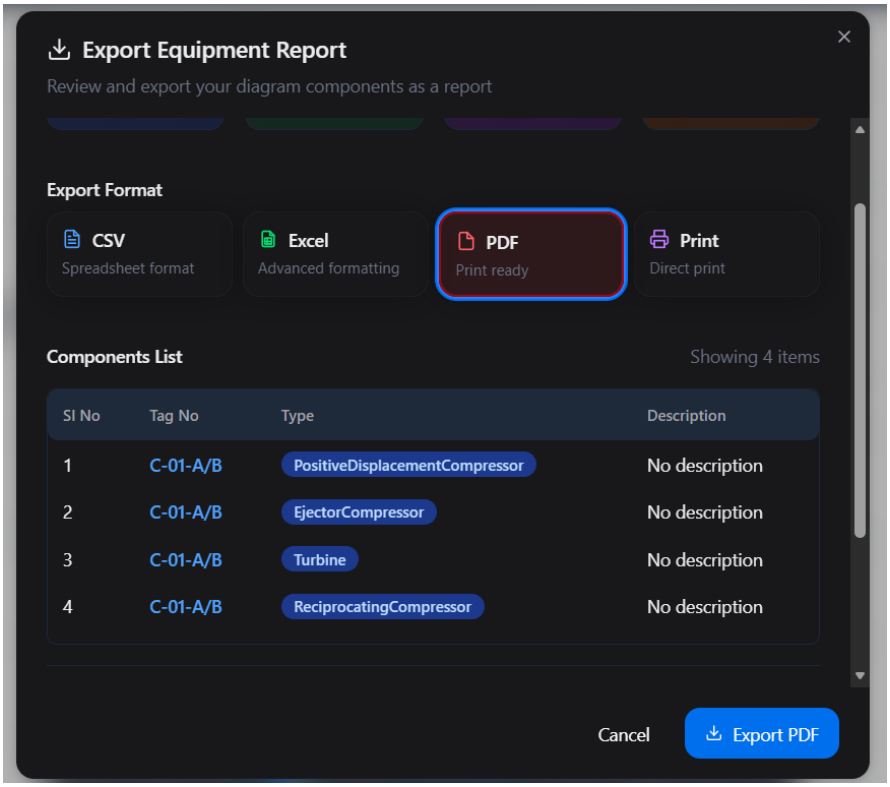

Fig. 6.2.2: Export Equipment Report Interface

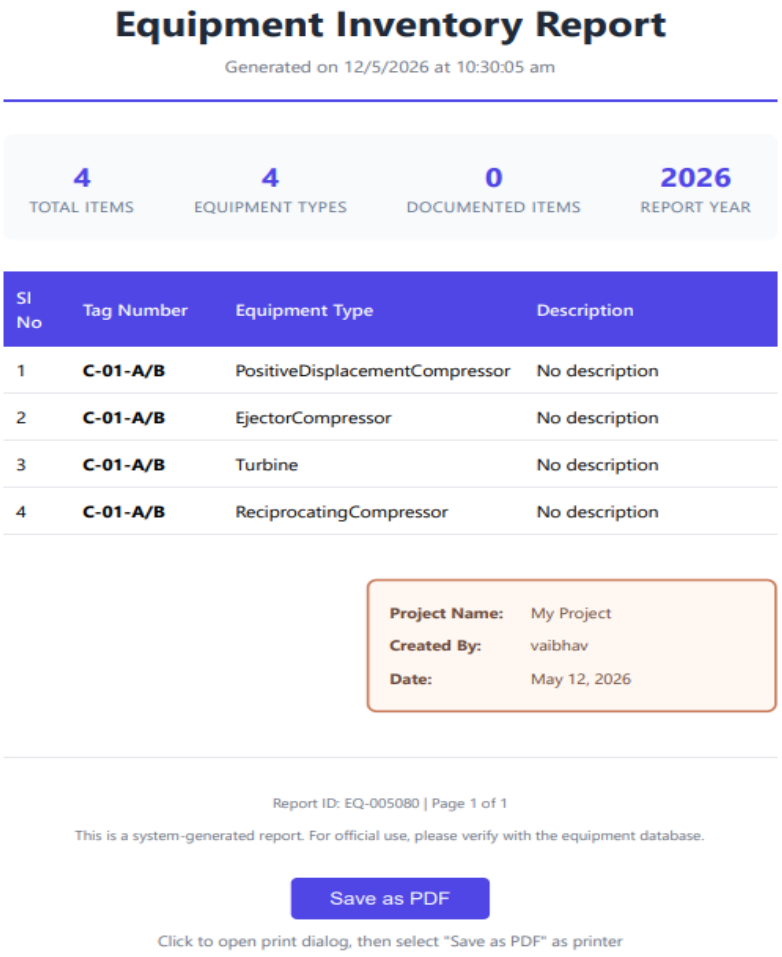

Fig. 6.2.3: Export Equipment Report

6.2.2 Zoom-Aware, Single-Commit Resize (PR #170)

- Added a stageScale prop to CanvasItemImage so resizing accounts for canvas zoom, and enforced a 20px logical minimum scaled by that zoom.
- Reset Konva's internal scale after each transform so stored dimensions stay in real pixels, and added Shift-key aspect-ratio locking via the transform boundBoxFunc.

- Introduced an onTransformEnd handler so the editor commits a single undo snapshot per resize instead of many intermediate ones.

6.2.2.1 Code Implementation

web-frontend/src/pages/Editor.tsx

```
/**
 * Called ONCE when the user finishes a resize gesture (Konva onTransformEnd).
 * Writes the final dimensions to the store in a single atomic update so only
 * ONE undo snapshot is pushed — mirroring the desktop ResizeCommand pattern.
 * Also resets waypoints on connected pipes so they re-route around the new bounds.
 */
const handleResizeEnd = (item: CanvasItem) => {
  if (!projectId) return;

  // Single atomic store write = single undo entry
  editorStore.updateItem(projectId, item.id, {
    x: item.x,
    y: item.y,
    width: item.width,
    height: item.height,
    rotation: item.rotation,
  });

  // Reset waypoints for all pipes attached to this component so they re-route
  const relatedConnections = connections.filter(
    (conn) => conn.sourceItemId === item.id || conn.targetItemId === item.id,
  );
```

```
  relatedConnections.forEach((conn) => {
    editorStore.updateConnection(projectId, conn.id, { waypoints: [] });
  });
};
```

6.3 Outcome

Every exported artefact now self-identifies with a project/author/date footer, and resizing became predictable at any zoom level with one clean undo step per action — an interim fix later superseded by the full rewrite in Chapter 8.

Chapter 7

Task 5: Web-Frontend Documentation

7.1 Problem Statement

The web-frontend README was minimal and did not reflect the system that had since been built — new contributors had no single reference for architecture, setup, or how to run the test suite.

7.2 Tasks Done

Pull Requests: #174

- Replaced the minimal README with a comprehensive document: framework badges, a table of contents, an overview, and an architecture/dependencies section.
- Documented the project structure and an environment-variable example (VITE_API_BASE_URL).
- Added setup and run instructions, development and testing guidance, and full descriptions of the core systems: the canvas editor, orthogonal routing, Zustand undo/redo, and the export system, along with test expectations and operational notes/limitations.

7.3 Outcome

The web-frontend now has a single, current reference document covering both architecture and day-to-day development workflow.

Chapter 8

Task 6: Component Resizing — Final Rewrite

8.1 Problem Statement

The zoom-aware fix in Chapter 6 still only supported uniform scaling. The requirement was independent control of width and height — users needed to be able to stretch a component in one axis only, with corner handles for diagonal resize — which meant replacing Konva's built-in Transformer with a purpose-built resize control.

8.2 Tasks Done

Pull Requests: #179, #180

An initial version of this change (PR #179) was opened and closed without merging within the same hour, in favour of PR #180 raised immediately after to resolve merge conflicts — both carry the same description below.

- Removed the Konva Transformer logic from CanvasItemImage and introduced a new HTML-based ResizableWrapper component with draggable resize handles, aspect-ratio-preserving resizing, collision checks, and minimum-size clamping.
- Integrated ResizableWrapper into the Editor: added a resizingItem state and computed displayItems to reflect in-flight bounds during a drag, used for both path recalculation and rendering.
- The wrapper overlay mounts only when a single item is selected, and unit tests were added alongside the new component.

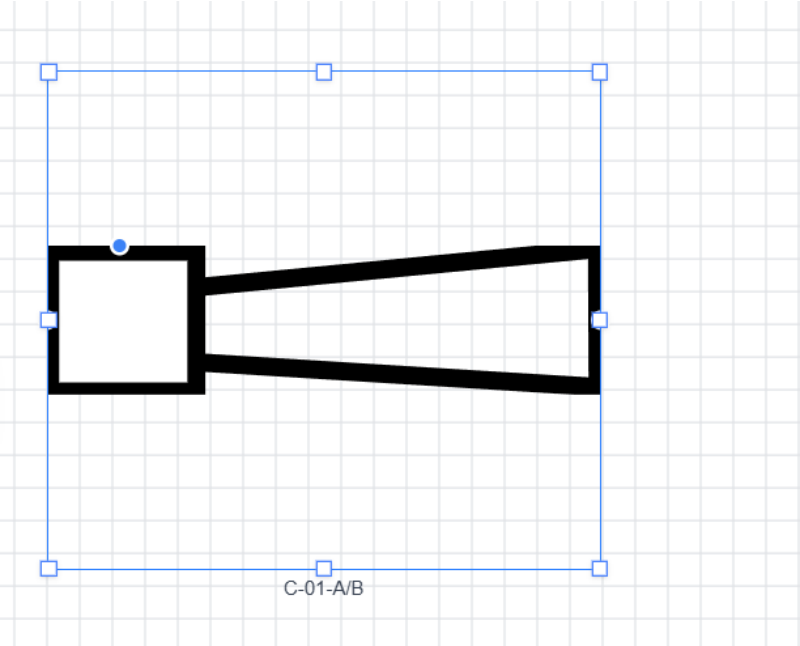

Fig. 8.1: Original component

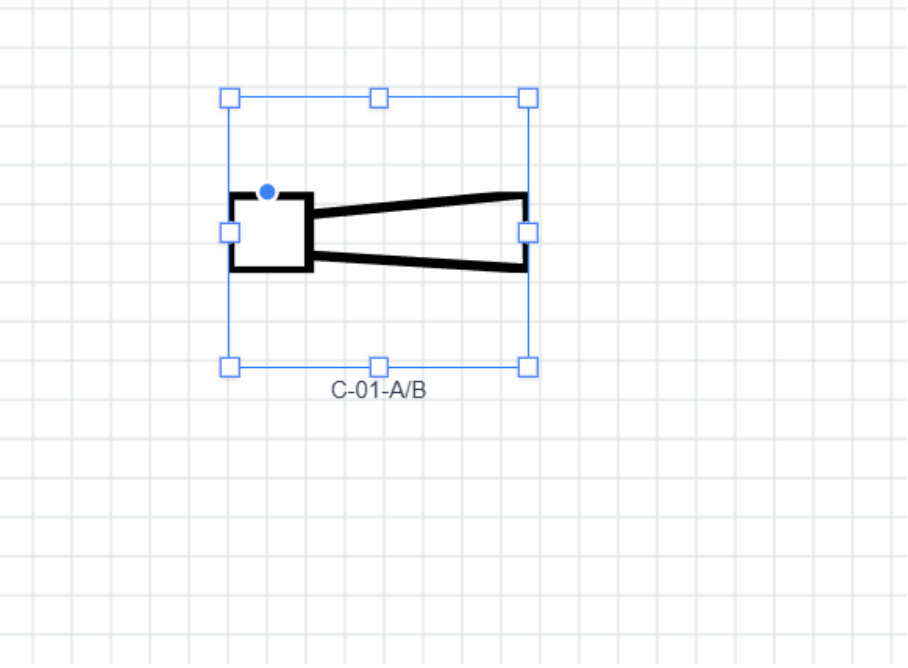

Fig. 8.2: Component resized diagonally

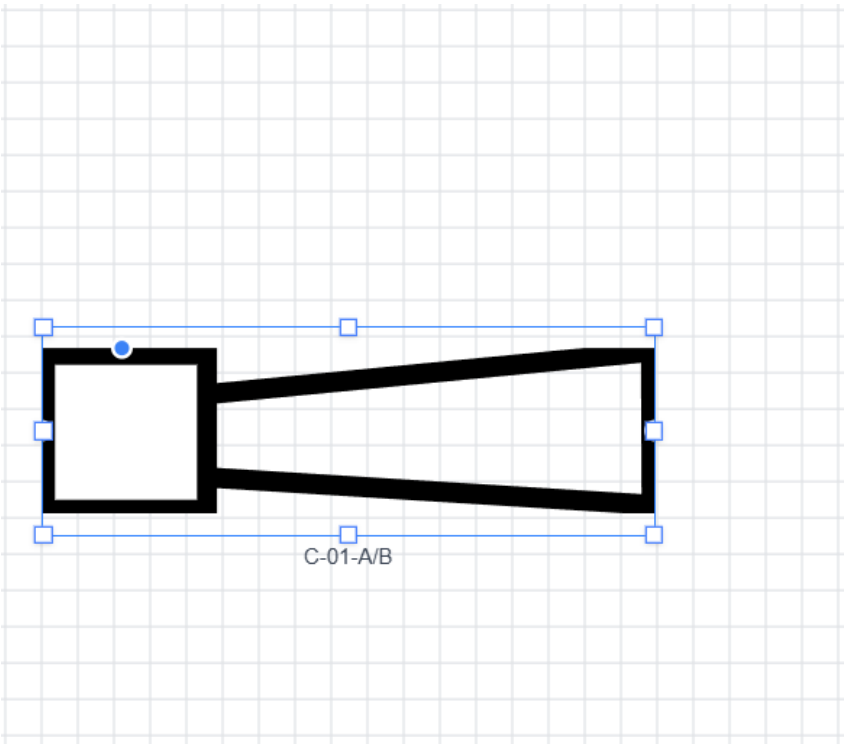

Fig. 8.3: Component resized vertically

8.3 Outcome

Components can now be resized independently along width and height (or both, via corner handles), rendered through a dedicated HTML overlay rather than Konva's generic transformer, with test coverage for the new behaviour.

Chapter 9

Conclusions

9.1 Tasks Accomplished

Across the internship, I raised 16 pull requests against the Chemical PFD Builder web-frontend (28 February – 8 June 2026), of which 14 were merged. Key accomplishments include:

- **Routing Engine:** rebuilt connection-drawing from manual waypoint placement through grid A* and spatial-graph A* to a final, documented, obstacle-avoiding pipeline, backed by a dedicated test suite.
- **AI Diagram Generation:** stood up the Gemini-based backend service, wired it into the editor, and hardened it with a component whitelist and stricter validation.
- **Component Resizing:** replaced Konva's built-in transformer with a custom HTML ResizableWrapper supporting independent width/height control, after an interim zoom-aware fix.
- **Supporting work:** an in-modal component-creation help guide, export/report footers, and a full rewrite of the web-frontend README.
- **Screening Task:** built and deployed a hybrid Django + React + PyQt5 application ahead of the internship, covering CSV analytics, charting, history management, and PDF reporting.

9.2 Skills Developed

9.2.1 Technical Skills

This internship strengthened my practical knowledge of modern web development using React.js, TypeScript, and Vite, and deepened my understanding of pathfinding and computational geometry while building and iterating on the A*-based routing engine. Working with Konva.js for interactive canvas development, I implemented graph-based algorithms such as Depth First Search (DFS) and Breadth First Search (BFS) for process validation, before eventually moving past Konva's built-in tooling to build a custom HTML-based interaction layer for resizing. I also gained practical experience integrating a third-party LLM (Google Gemini) behind a validated API contract for AI-assisted diagram generation, wrote unit tests using Vitest (including mocking react-konva dependencies), and improved my understanding of scalable frontend architecture, state management, and debugging complex frontend issues. Documenting a non-trivial frontend system for other contributors rounded out this technical growth.

9.2.2 Professional Skills

Working inside an existing codebase alongside a mentor and teammates sharpened my code-review habits — several of my pull requests were revised in response to specific review comments, such as fixing a nested-button HTML violation and a line-over-component rendering bug flagged in review. I learned to treat a closed, unmerged pull request as a normal part of iterating toward a working solution rather than as a failure, and to respond quickly to priority bug reports from the mentor. More broadly, being part of the FOSSEE development team improved my collaboration and communication skills through regular discussions, pull requests, code reviews, and mentor feedback. I gained experience understanding existing codebases, implementing production-ready features, debugging real-world issues,

documenting technical work, and following structured software development practices — all of which sharpened my ability to break problems into manageable tasks and deliver reliable solutions while maintaining code quality and readability.

9.3 Future Scope

- Extend the obstacle-avoidance router to support diagonal or curved connection styles where orthogonal routing is visually cluttered.
- Add automated evaluation of AI-generated diagrams (e.g. connection-completeness checks) as a CI gate rather than a manual testing pass.
- Support real-time collaborative editing using WebSocket-based synchronization.
- Bring the resize interaction to the desktop (PyQt5) frontend for parity with the web editor.
- Expand the export footer system to support custom templates and organisation branding.
- Add advanced export options, including CAD-compatible formats and customizable engineering reports.
- Integrate process simulation software for automatic process verification and analysis.
- Enhance performance for handling larger and more complex industrial process diagrams.

The FOSSEE Semester Long Internship provided hands-on experience in iterating a real, shared codebase from first bug report to merged, tested feature, and gave me valuable exposure to open-source software development, modern frontend

engineering, and collaborative development workflows. Working on the Chemical PFD Builder let me apply theoretical knowledge to practical engineering software while significantly improving both my technical and professional skills — an experience that has strengthened my confidence in contributing to large-scale software projects and laid a solid foundation for my future career in software development.

Chapter 10

Bibliography

[1] FOSSEE Project, FOSSEE Website, https://fossee.in/.

[2] Chemical PFD Builder repository, https://github.com/FOSSEE/Chemical-PFD-Web-Desktop.

[3] Vaibhav Kumar, Chemical Equipment Parameter Visualizer (Screening Task submission), https://github.com/Vaibhav-Kumar10/FOSSEE_2026-Web_Based_Application_Screening_Task_Submission.

[4] Pull requests by Vaibhav-Kumar10 https://github.com/FOSSEE/Chemical-PFD-Web-Desktop/pulls?q=is%3Apr+is%3Aclosed+author%3AVaibhav-Kumar10.