FOSSEE Semester Long Internship Report

On

Development of Chemical PFD Builder

Submitted by
Jayraj Sawant

3rd Year B.Tech Student, EXCS Department,

Vidyalankar Institute of Technology,

Wadala

Under the Guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

Indian Institute of Technology Bombay

Mentors: Chirag Koyande

July 3, 2026

Acknowledgments

I would like to express my sincere gratitude to the FOSSEE Project at IIT Bombay for providing me with the opportunity to be a part of the internship program. This internship has been an enriching learning experience, allowing me to contribute to the development of an open-source software project while working alongside an experienced and supportive team.

As a member of the **Web Frontend Team** for the **Chemical PFD Builder**, I worked on implementing new features, resolving bugs, improving the editor's functionality, and enhancing the overall user experience. My contributions included graph-based process validation, AI-assisted diagram generation features, canvas interaction improvements, dashboard enhancements, state management, and testing. Through these tasks, I gained valuable experience in **React.js, TypeScript, Konva.js, Zustand, Git, GitHub, debugging, testing, and collaborative open-source development**.

I sincerely thank **Prof. Prabhu Ramachandran** for providing students with such an excellent platform to learn, contribute, and grow through the FOSSEE initiative.

I would also like to express my heartfelt gratitude to my mentor, **Chirag Koyande**, for his continuous guidance, encouragement, and technical support throughout the internship. His valuable feedback and mentorship played a significant role in helping me understand the project and improve my problem-solving skills. I am equally thankful to my fellow team members for their cooperation, discussions, and collaborative spirit during the course of the internship.

This internship has been a significant milestone in my learning journey, strengthening both my technical knowledge and professional skills. The experience of contributing to a real-world open-source project has greatly increased my confidence and prepared me for future opportunities in software development.

Contents

CHAPTER 1: INTRODUCTION 3
1.1 FOSSEE Project 3
1.2 Chemical PFD Builder 4
1.3 System Overview 4
1.4 Key Features 5
1.5 Internship Objectives 6
1.6 Conclusion 6
CHAPTER 2: SCREENING TASK 7
2.1 Problem Statement 7
2.2 Project Overview 7
2.3 Key Features Required 8
2.4 System Architecture 8
2.5 Features Implemented 9
2.6 Deployment 9
CHAPTER 3: 10
TASK 1: Project Setup & Codebase Familiarization 10
3.1 Problem Statement 10
3.2 Tasks Done 10
3.3 Outcome 11
CHAPTER 4: 12
TASK 2: Fix Initial Component Scaling Issue on First Drop 12
4.1 Problem Statement 12
4.2 Tasks Completed 12
4.3 Implementation 12
4.4 Code Snippet 1 (New Applied) 13
4.5 Code Snippet 2 (Previous Version) 13
4.6 Outcome 13

CHAPTER 5: 15
TASK 3: Graph-Based Process Validation Engine 15
5.1 Problem Statement 15
5.2 Tasks Completed 15
5.3 Implementation 15
5.4 Code Snippet 1 16
5.5 Code Snippet 2 17
5.6 Code Snippet 3 18
5.7 Code Snippet 4 19
5.8 Outcome 20
CHAPTER 6: 21
TASK 4: Draggable Waypoint Adjustment for Orthogonal Connection Lines 21
6.1 Problem Statement 21
6.2 Tasks Completed 21
6.3 Code Snippet 1 21
6.4 Code Snippet 2 22
6.5 Code Snippet 3 24
6.6 Outcome 25
CHAPTER 7: 26
TASK 5: Dashboard Interaction Improvements 26
7.1 Problem Statement 26
7.2 Tasks Completed 26
7.3 Outcome 26
CHAPTER 8: 28
TASK 6: Resolve Project Opening Issue from Dashboard 28
8.1 Problem Statement 28
8.2 Tasks Completed 28
8.3 Code Snippet 28
8.4 Explanation 29
8.5 Outcome 29

CHAPTER 9: 30
TASK 7: AI Diagram Generation – Initial User Interface 30
9.1 Problem Statement 30
9.2 Tasks Completed 30
9.3 Code Snippets 30
9.4 Explanation 31
9.5 Outcome 31
CHAPTER 10: 32
TASK 8: Enhanced AI Generate Interface 32
10.1 Problem Statement 32
10.2 Tasks Completed 32
10.3 Code snippet 1 32
10.4 Explanation 33
10.5 Code snippet 2 33
10.6 Explanation 33
10.7 Code snippet 3 34
10.8 Explanation 34
10.9 Outcome 34
CHAPTER 11: 36
TASK 9: Fix AI Component Mapping, Variant Handling, and Connection Generation 36
11.1 Problem Statement 36
11.2 Tasks Completed 36
11.3 Code Snippet 1 37
11.4 Explanation 37
11.5 Code snippet 2 37
11.6 Explanation 39
11.7 Outcome 39
CHAPTER 12: TASK 10: AI Smart Component Matching 40
12.1 Problem Statement 40
12.2 Tasks Completed 40

12.3 Code snippet 1 40
12.4 Explanation 41
12.5 Code snippet 2 41
12.6 Explanation 42
12.7 Outcome 42
CHAPTER 13: 43
TASK 11: Preserve Undo/Redo History for Component Addition 43
13.1 Problem Statement 43
13.2 Tasks Completed 44
13.3 Code snippet 1 44
13.4 Explanation: 44
13.5 Code snippet 2 45
13.6 Explanation 45
13.7 Outcome 45
CHAPTER 14: 46
TASK 12: Component Image Resizing with Bounding Box 46
14.1 Problem Statement 46
14.2 Tasks Completed 47
14.3 CODE SNIPPET 1 47
14.4 Explanation: 47
14.5 Outcome 47
CHAPTER 15: Conclusion 48
15.1 Tasks Accomplished 48
15.2 Skills Developed 50
CHAPTER 16: Bibliography 52

CHAPTER 1: INTRODUCTION

1.1 FOSSEE Project

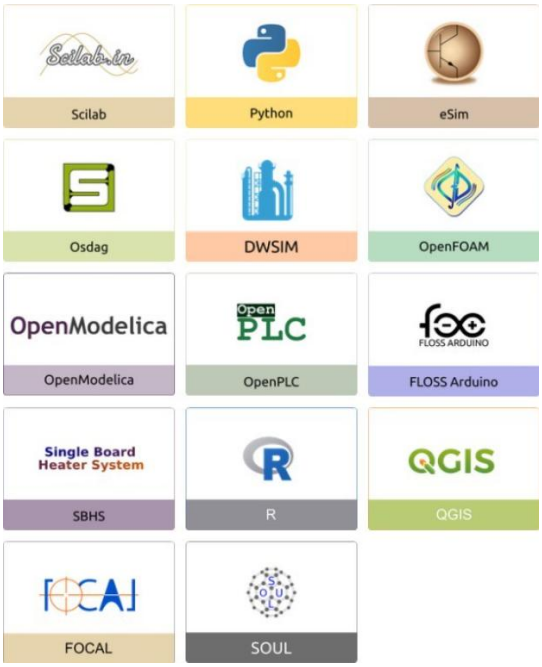
The Free/Libre and Open Source Software for Education (FOSSEE) project is an initiative under the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India. The project is coordinated by IIT Bombay with the objective of promoting the adoption of Free and Open Source Software (FOSS) in education, research, and industry.

FOSSEE encourages students, educators, researchers, and professionals to contribute to open-source software through collaborative development, internships, workshops, and community-driven projects. By promoting freely accessible engineering and scientific software, the initiative aims to reduce dependence on proprietary tools while improving accessibility to high-quality technical resources.

The project supports numerous activities including:

- Textbook Companion projects
- Lab Migration
- Niche Software Development
- Forums and Community Support
- Workshops and Training Programs
- Open Source Internship Programs

These initiatives provide students with practical software development experience while contributing to software that benefits the wider engineering community.



1.2 Chemical PFD Builder

The Chemical PFD Builder is an open-source software application developed under the FOSSEE Project for creating and managing Process Flow Diagrams (PFDs) used in chemical engineering.

Process Flow Diagrams are fundamental engineering documents used to represent process equipment, material flow, and process operations in chemical plants. Traditionally, engineers rely on expensive proprietary software to create these diagrams. Chemical PFD Builder offers a free and cross-platform alternative while maintaining professional functionality.

The application is available for both desktop and web platforms, allowing users to design, edit, save, and manage PFDs efficiently through a modern user interface.

1.3 System Overview

The Chemical PFD Builder follows a hybrid architecture consisting of a shared backend with separate web and desktop frontends.

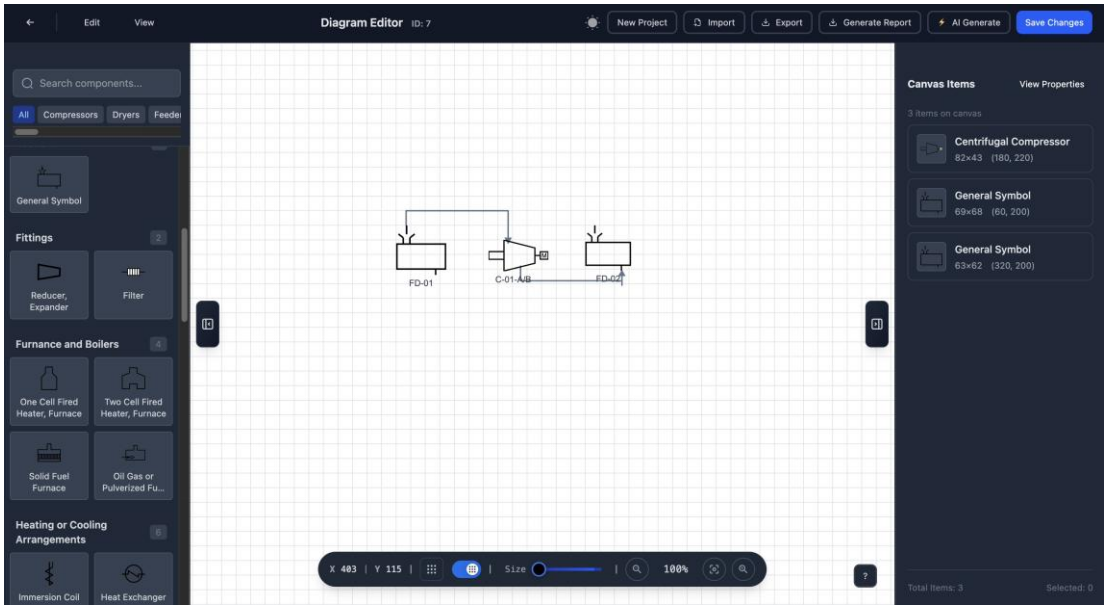
The backend provides REST APIs responsible for user authentication, project management, component storage, report generation, and cloud synchronization. The frontend is developed using modern web technologies and provides an interactive canvas where users can create engineering diagrams using drag-and-drop operations.

This architecture enables users to seamlessly continue working across different platforms while maintaining a consistent project structure.

1.4 Key Features

Some of the major features provided by the Chemical PFD Builder include:

- Interactive drag-and-drop canvas for creating Process Flow Diagrams
- Large library of standard chemical engineering components
- Automatic routing of connections between components
- AI-assisted diagram generation using natural language prompts
- Graph-based process validation
- Automatic component labeling
- Cloud-based project management
- PDF and Excel report generation
- Export support for multiple engineering formats
- Cross-platform support for Web and Desktop



1.5 Internship Objectives

The objective of this internship was to contribute to the frontend development of the Chemical PFD Builder while gaining practical experience in collaborative open-source software development.

During the internship, I worked primarily on enhancing the web application's functionality by implementing new features, resolving bugs, improving user interactions, and contributing to the overall reliability of the editor. My work involved technologies such as React.js, TypeScript, Konva.js, Zustand, and GitHub, with contributions spanning graph algorithms, AI-assisted workflows, canvas rendering, state management, and application testing.

The experience provided valuable exposure to real-world software engineering practices including debugging, code reviews, version control, feature development, testing, and collaborative development within an open-source project.

1.6 Conclusion

The Chemical PFD Builder is a feature-rich, open-source platform for creating and managing Process Flow Diagrams (PFDs). During this internship, I contributed to the web frontend by implementing new features, enhancing existing functionality, and resolving critical issues related to validation, AI-assisted diagram generation, canvas interactions, and undo/redo functionality. This experience strengthened my technical skills in React, TypeScript, Konva.js, Zustand, Git, and collaborative open-source development, making it a valuable learning experience.

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.

2.2 Project Overview

Create a Web + Desktop application that allows users to upload a CSV file containing chemical equipment data (Equipment Name, Type, Flowrate, Pressure, Temperature). The Django backend parses the data, performs analysis, and provides summary statistics via API. Both React (Web) and PyQt5 (Desktop) frontends consume this API to display data tables, charts, and summaries.

2.3 Key Features Required

- **CSV Upload:** Both interfaces allow CSV upload to backend
- **Data Summary API:** Returns count, 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

Task Completed:

The screening task involved developing **VizoChem**, a web-based application for chemical data visualization and analysis. The project was built using a React.js frontend and a

Django REST Framework backend, enabling users to upload datasets, analyze chemical information, visualize results through interactive charts, and generate reports.

The objective of the project was to demonstrate proficiency in full-stack web development, REST API integration, data visualization, and deployment of a production-ready application.

2.4 System Architecture

Backend:
Developed using **Django** and **Django REST Framework**, responsible for handling CSV uploads, data processing, REST APIs, authentication, and PDF report generation.

Frontend:
Built with **React.js** and **Vite**, providing an intuitive interface for dataset upload, visualization, report generation, and interaction with backend services through REST APIs.

2.5 Features Implemented

The following features were successfully implemented during the screening task:

- CSV dataset upload and validation
- Interactive chemical data visualization
- Dynamic charts and statistical summaries
- REST API integration between frontend and backend
- PDF report generation
- Responsive user interface
- Error handling and user feedback
- Cloud deployment for public accessibility

2.6 Deployment

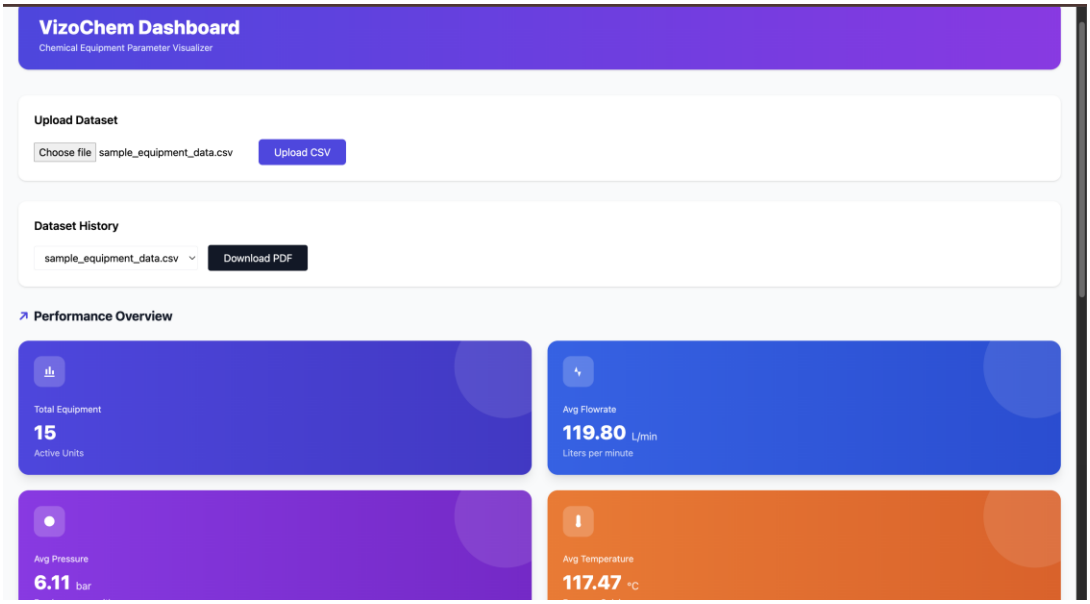
The application was deployed using modern cloud platforms to demonstrate real-world deployment practices.

Frontend (Vercel):
https://vizo-chem.vercel.app

Backend (Render):
https://vizochem-backend.onrender.com

Source Code (GitHub):
https://github.com/Fb-JAYRAJ/VizoChem

The successful deployment verified the complete end-to-end workflow, including frontend-backend communication, API integration, and report generation in a production environment.



CHAPTER 3:

TASK 1: Project Setup & Codebase Familiarization

3.1 Problem Statement

Before contributing to the Chemical PFD Builder, it was essential to understand the existing project architecture, development workflow, and code organization. A proper local development environment also had to be configured to enable feature development and debugging.

3.2 Tasks Done

The following tasks were completed during the initial phase of the internship:

- Cloned the Chemical PFD Builder repository from GitHub.
- Set up the frontend development environment using React, TypeScript, and Vite.
- Installed and configured all required project dependencies.
- Explored the project structure and understood the organization of components, pages, utilities, state management, and APIs.
- Studied the application's workflow, including canvas rendering, routing, state management, and backend communication.
- Successfully ran the project locally and verified the development setup.

3.3 Outcome

A fully functional local development environment was established, and a thorough understanding of the project architecture was gained. This foundation enabled efficient debugging, feature implementation, and contribution to subsequent development tasks throughout the internship.

CHAPTER 4:

TASK 2: Fix Initial Component Scaling Issue on First Drop

4.1 Problem Statement

When a component was added to the canvas for the first time, it appeared with an incorrect size due to improper initialization of the scaling reference. The issue occurred only during the initial component drop, resulting in inconsistent rendering and an unreliable user experience. Since component placement is a core operation of the editor, it was essential to ensure that newly added components retained their intended dimensions from the moment they were created.

4.2 Tasks Completed

The following improvements were implemented to resolve the issue:

- Investigated the cause of inconsistent component scaling during the initial drop.
- Identified that the reference used to track previous component size (prevComponentSizeRef) was not initialized correctly.
- Initialized the reference using the current component size during the first render.
- Verified that all subsequently added components were rendered with consistent dimensions.

4.3 Implementation

The issue originated because the previous component size reference did not contain the correct initial value during the first render. As a result, the scaling calculation used an incorrect reference, causing the first dropped component to appear at an unexpected size.

The implementation initialized the reference with the current component size, ensuring that the scaling logic always had a valid starting point before any resize calculations were performed.

4.4 Code Snippet 1 (New Applied)

```
const prevComponentSizeRef = useRef(componentSize);
```

Explanation

The previous component size reference is initialized using the current component size instead of an undefined value. This ensures that the scaling calculations begin with the correct dimensions.

4.5 Code Snippet 2 (Previous Version)

```
const prevComponentSizeRef = useRef(1500);
```

Explanation

The scaling logic now compares the current size against a properly initialized reference, preventing incorrect scaling during the initial component placement.

4.6 Outcome

The component scaling issue was successfully resolved. Newly added components now maintain their correct dimensions during the first placement, providing a consistent editing experience and improving the reliability of the canvas rendering system.

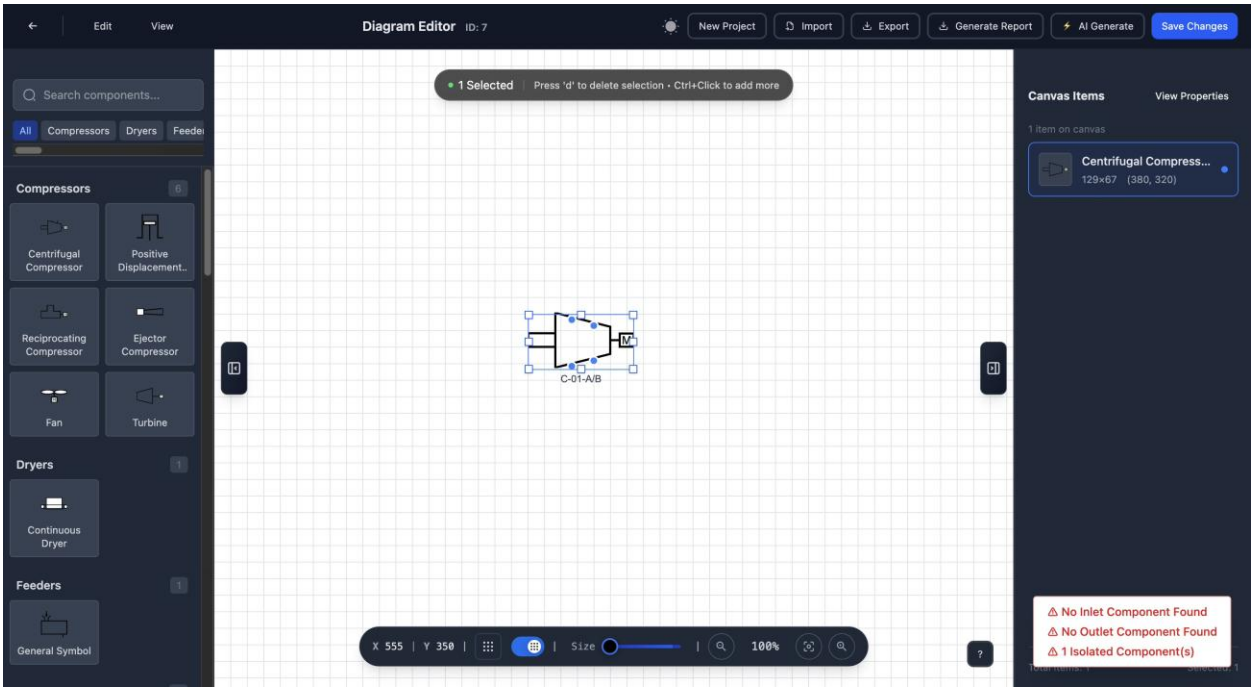


Figure 4.1: Correct rendering of a newly dropped component after fixing the initial scaling issue.

CHAPTER 5:

TASK 3: Graph-Based Process Validation Engine

5.1 Problem Statement

The Chemical PFD Builder previously allowed users to create process flow diagrams without validating their structural correctness. This made it possible to create invalid diagrams containing disconnected components, missing inlets or outlets, isolated equipment, or circular process loops. A validation mechanism was therefore required to ensure that diagrams satisfied fundamental process flow rules and to provide immediate feedback during editing.

5.2 Tasks Completed

The following validation features were implemented:

- Converted Process Flow Diagrams into an adjacency list graph representation.
- Implemented Depth First Search (DFS) for circular loop detection.
- Implemented Breadth First Search (BFS) for process flow validation.
- Added inlet validation.
- Added outlet validation.
- Implemented isolated component detection.
- Integrated validation results into the editor interface for real-time feedback.

5.3 Implementation

Each component in the Process Flow Diagram was represented as a graph node, while every connection between components was represented as an edge. This graph representation enabled the application of graph traversal algorithms to validate the logical correctness of the process flow.

5.4 Code Snippet 1

```
nodes.forEach((node) => {
  const id = String(node.id);

  adjacencyList[id] = [];
  inDegree[id] = 0;
  outDegree[id] = 0;
});

connections.forEach((conn) => {
  const source = String(conn.sourceItemId);
  const target = String(conn.targetItemId);

  if (!adjacencyList[source] || !adjacencyList[target]) return;

  adjacencyList[source].push(target);
  outDegree[source]++;
  inDegree[target]++;
});
```

Explanation

The Process Flow Diagram is transformed into an adjacency list representation, where each equipment component acts as a graph node and every connection acts as a directed edge. During graph construction, the in-degree and out-degree of each node are also calculated. These values are later used for inlet detection, outlet detection, and process validation.

5.5 Code Snippet 2

```
nodes.forEach((node) => {
  if (inDegree[node] === 0 && outDegree[node] === 0) {
    isolated.push(node);
  }
});


const inletNodes = nodes.filter(
  (node) => inDegree[node] === 0 && outDegree[node] > 0,
);


const outletNodes = nodes.filter(
  (node) => outDegree[node] === 0 && inDegree[node] > 0,
);
```

Explanation

The validation engine first identifies isolated components by checking whether a node has neither incoming nor outgoing connections. It then determines inlet and outlet nodes using the calculated in-degree and out-degree values. This ensures that every valid process flow begins with at least one inlet and ends with at least one outlet.

5.6 Code Snippet 3

```
const visited = new Set<string>();

const recursionStack = new Set<string>();
```

```
function dfs(node: string) {
  if (recursionStack.has(node)) {
    circular.add(node);
    return;
  }

  if (visited.has(node)) return;

  visited.add(node);
  recursionStack.add(node);

  for (const neighbor of adjacencyList[node]) {
    dfs(neighbor);
  }

  recursionStack.delete(node);
}
```

Explanation

Depth First Search (DFS) is used to identify circular dependencies within the Process Flow Diagram. A recursion stack keeps track of the current traversal path. If a node is encountered that already exists in the recursion stack, a circular loop is detected and reported to the user.

5.7 Code Snippet 4

```
const [validationResult, setValidationResult] = useState<any>(null);



// Rebuild graph and validate process structure whenever components or connections change

useEffect(() => {

  if (!droppedItems || droppedItems.length === 0) return;



  const graph = buildGraph(droppedItems, connections);

  const result = validateGraph(graph);



  setValidationResult(result);

}, [droppedItems, connections]);
```

Explanation

This code integrates the graph validation engine with the editor to provide real-time feedback. Whenever a component is added, removed, moved, or a connection is modified, the useEffect hook is triggered. The current Process Flow Diagram is first converted into a graph representation using the buildGraph() function, after which the validateGraph() function performs structural validation, including cycle detection, flow continuity checks, inlet and outlet validation, and isolated component detection. The validation results are then stored in the component state using setValidationResult(), enabling the user interface to immediately display warnings or validation messages. This ensures that users receive continuous feedback while constructing the diagram, helping them identify and correct structural issues without requiring a separate validation step.

5.8 Outcome

The validation engine provides immediate feedback whenever an invalid process flow is detected. Users are informed about disconnected components, missing inlets or outlets, isolated equipment, and circular loops, significantly improving diagram correctness before report generation or export.

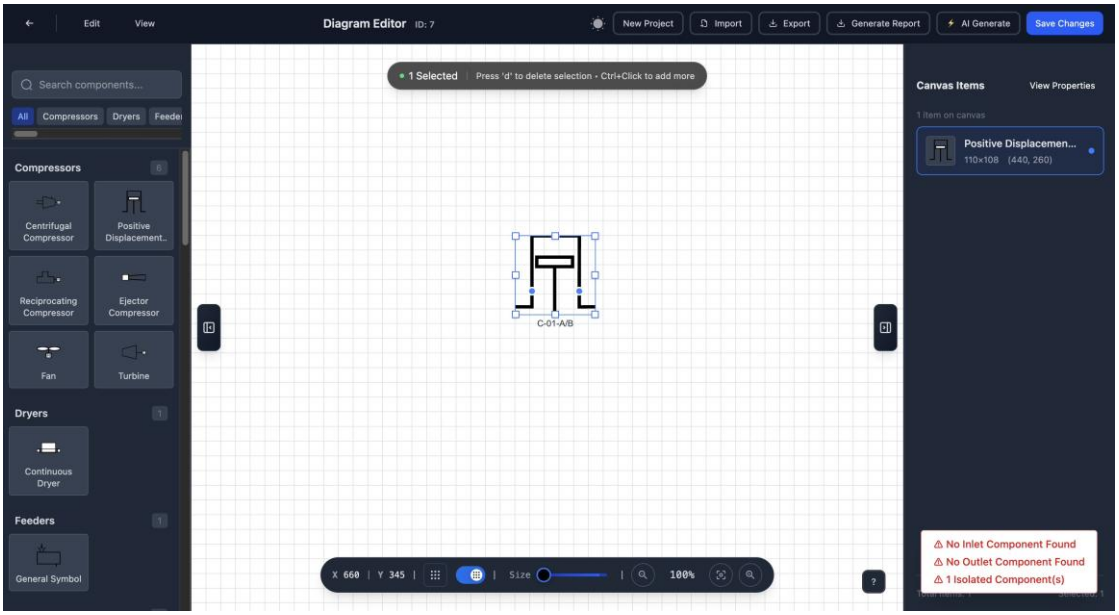


Figure 5.1: Real-time validation messages displayed for an invalid Process Flow Diagram.

CHAPTER 6:

TASK 4: Draggable Waypoint Adjustment for Orthogonal Connection Lines

6.1 Problem Statement

The existing orthogonal connection lines were automatically routed between components, but users had no control over adjusting the path manually. This made it difficult to organize complex Process Flow Diagrams where multiple connection lines overlapped or required custom routing.

6.2 Tasks Completed

- Added draggable waypoint support for orthogonal connection lines.
- Allowed users to manually reposition intermediate waypoints.
- Updated connection routing dynamically while preserving obstacle avoidance.
- Maintained orthogonal (horizontal and vertical) routing after waypoint movement.
- Implemented the feature as an experimental enhancement for future improvements.

6.3 Code Snippet 1

```
if (dx > dy) {
  newX = e.target.x();
  e.target.y(p.y);
} else {
  newY = e.target.y();
  e.target.x(p.x);
```

```
}


onWaypointDrag?.(i, {
  x: newX,
  y: newY,
});
```

Explanation

This logic enables users to drag waypoints while preserving orthogonal routing. During dragging, movement is constrained to either the horizontal or vertical direction based on the dominant drag distance. Once the waypoint position is updated, the new coordinates are passed to the parent component through the onWaypointDrag() callback, allowing the connection path to be recalculated in real time.

6.4 Code Snippet 2

```
const handleWaypointDrag = (
  connectionId: number,
  index: number,
  pos: { x: number; y: number },
) => {
  setConnections((prev) =>
    prev.map((conn) => {
      if (conn.id !== connectionId) return conn;
```

```
    const waypoints = conn.waypoints ? [...conn.waypoints] : [];

    waypoints[index] = pos;

    return {
      ...conn,
      waypoints,
    };
  }),
 );
};

// If the component moved, reset connection waypoints so routing recalculates
if (updates.x !== undefined || updates.y !== undefined) {
 const relatedConnections = connections.filter(
  (conn) => conn.sourceItemId === itemId || conn.targetItemId === itemId,
 );

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

Explanation

The editor maintains the updated waypoint positions within the connection state whenever a waypoint is dragged. Additionally, if a connected component is moved, all manually placed waypoints associated with that component are cleared. This allows the routing algorithm to recalculate a fresh path based on the new component position, preventing invalid or distorted connection layouts.

6.5 Code Snippet 3

```
const waypoint = conn.waypoints[0];


const first = smartRoute(startStandoff, waypoint, items);
const second = smartRoute(waypoint, endStandoff, items);


bends = [...first, waypoint, ...second];
```

Explanation

The routing engine incorporates user-defined waypoints into the connection path by dividing the route into two segments. The first segment is calculated from the source component to the waypoint, while the second segment connects the waypoint to the destination component. The resulting path preserves obstacle avoidance while respecting the manually adjusted waypoint position, providing users with greater flexibility in organizing complex Process Flow Diagrams.

6.6 Outcome

The implemented waypoint system provides interactive editing of connection paths while preserving orthogonal routing behavior. Users can adjust connection layouts

without affecting existing component relationships, resulting in cleaner and more readable Process Flow Diagrams.

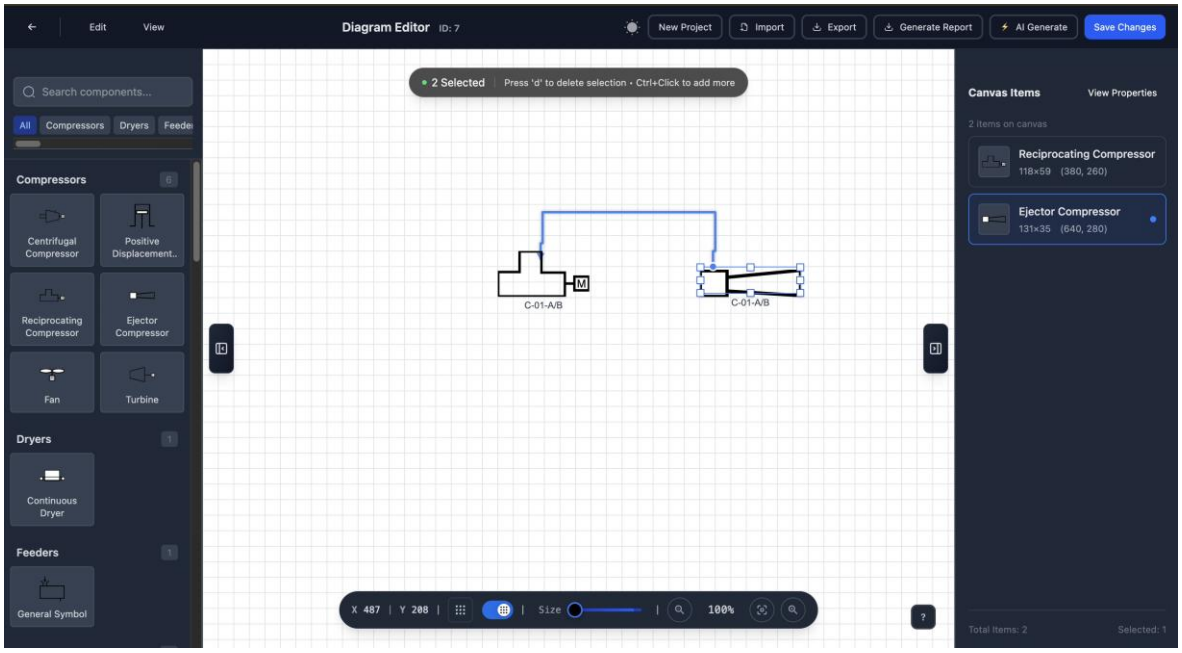


Figure 6.1: Manual adjustment of orthogonal connection using draggable waypoint.

CHAPTER 7:

TASK 5: Dashboard Interaction Improvements

7.1 Problem Statement

The Dashboard contained nested interactive elements, resulting in invalid HTML structure and inconsistent click behavior. This occasionally caused navigation conflicts and reduced accessibility.

7.2 Tasks Completed

- Refactored Dashboard card interaction logic.
- Removed nested button rendering.
- Simplified component hierarchy.
- Improved click handling.
- Enhanced accessibility using semantic HTML.
- Verified functionality using the Vite development server.

7.3 Outcome

The refactoring improved dashboard responsiveness and accessibility while eliminating rendering conflicts caused by nested interactive elements. Project cards now respond consistently to user interactions across supported browsers.

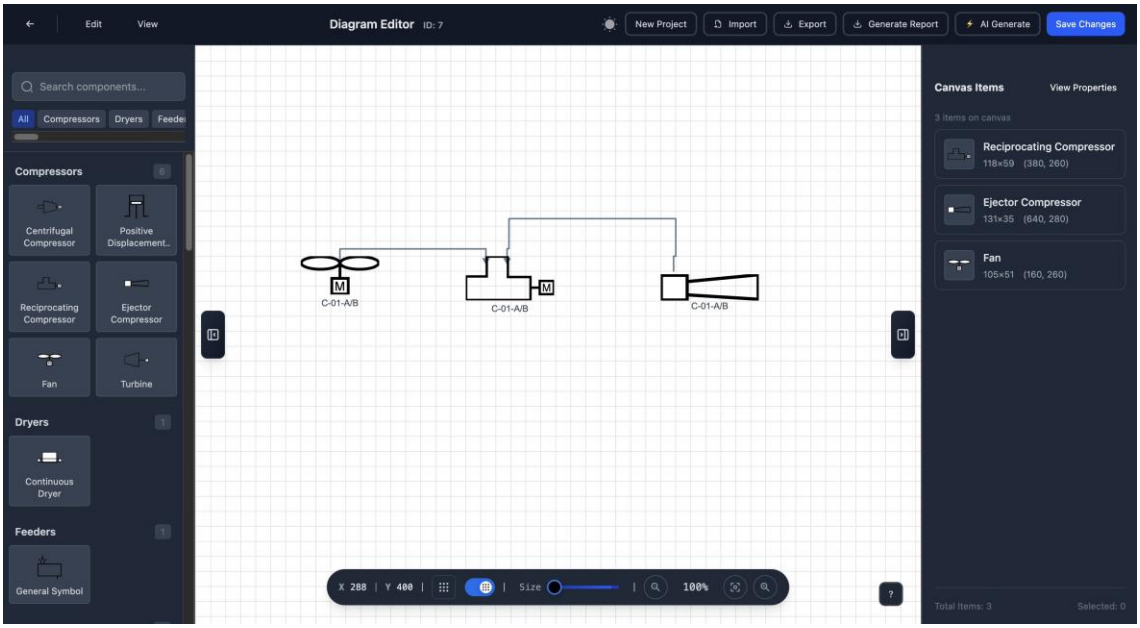


Figure 7.1: Dashboard after interaction and accessibility improvements.

CHAPTER 8:

TASK 6: Resolve Project Opening Issue from Dashboard

8.1 Problem Statement

Users were unable to reliably open projects directly from the dashboard due to issues in the navigation flow and interaction handling. The existing implementation affected usability and reduced accessibility.

8.2 Tasks Completed

- Restored project navigation using **React Router**.
- Replaced non-interactive text elements with semantic button elements.
- Improved click handling to eliminate interaction conflicts.
- Enhanced accessibility by following semantic HTML practices.
- Ensured projects open consistently from the dashboard.

8.3 Code Snippet

```
<button

  onClick={() => navigate(`/editor/${proj.id}`)}

  className="text-primary text-sm font-medium cursor-pointer bg-transparent border-none p-0 hover:underline"

  type="button"

>
```

8.4 Explanation

This implementation restores project navigation by using React Router to open the selected project in the editor. Replacing the previous non-interactive element with a semantic <button> improves accessibility, provides better keyboard support, and ensures consistent click behavior across browsers.

8.5 Outcome

The dashboard now allows users to open projects reliably with a single click. The updated implementation improves usability, accessibility, and overall navigation consistency within the application.

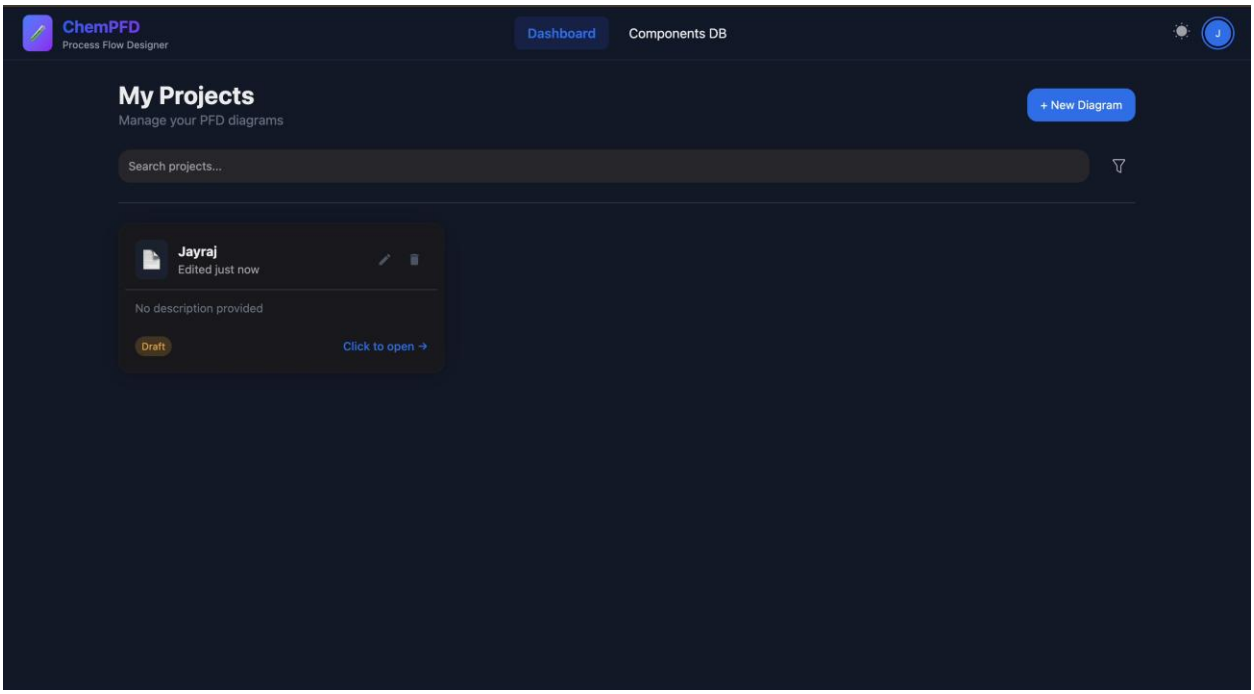


Figure 8.1: *Opening a project directly from the dashboard using the updated navigation flow.*

CHAPTER 9:

TASK 7: AI Diagram Generation – Initial User Interface

9.1 Problem Statement

Creating Process Flow Diagrams manually can be time-consuming. To prepare the editor for AI-assisted diagram generation, a dedicated interface was required where users could provide natural language prompts describing the desired process.

9.2 Tasks Completed

- Added an **AI Generate** button to the editor toolbar.
- Implemented a modal for AI prompt input.
- Added a text area for entering process descriptions.
- Added a **Generate** button inside the modal.
- Integrated the UI with the editor for future backend support.

9.3 Code Snippet

```
<Button
 className="border-gray-300 dark:border-gray-700 text-gray-700 dark:text-gray-300"
 size="sm"
 startContent={<span> ⚡ </span>}
 variant="bordered"
 onPress={() => setShowAIModal(true)}>
```

AI Generate

</Button>

9.4 Explanation

An AI Generate button was added to the editor toolbar to provide users with quick access to AI-assisted diagram generation. When clicked, it opens a dedicated modal where users can describe a process using natural language.

9.5 Outcome

The editor now includes a dedicated user interface for AI-assisted Process Flow Diagram generation. Although backend functionality was not yet available, the frontend workflow was successfully established and made ready for future integration.

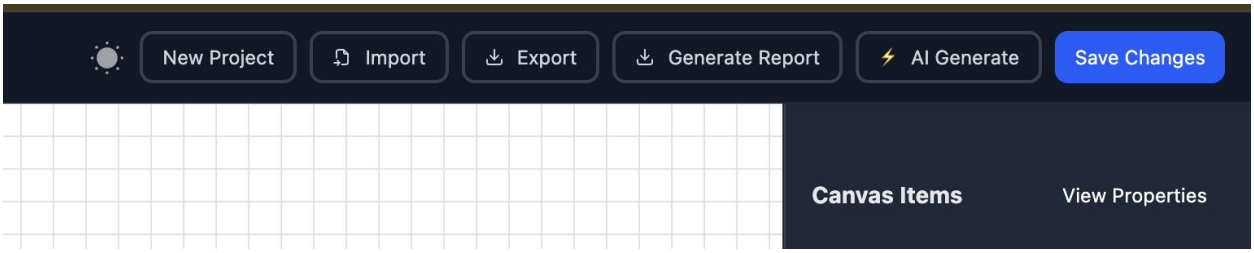


Figure 9.1: *AI Generate interface integrated into the Chemical PFD Builder editor.*

CHAPTER 10:

TASK 8: Enhanced AI Generate Interface

10.1 Problem Statement

After introducing the initial AI Generate interface, additional improvements were required to make the feature more user-friendly and ready for backend integration. The interface needed better user guidance, loading indicators, input validation, and error handling to provide a smoother user experience.

10.2 Tasks Completed

- Enhanced the AI Generate modal interface.
- Added a text area for entering natural language prompts.
- Added a **Generate Diagram** button inside the modal.
- Implemented loading state while the diagram generation request is processed.
- Disabled the Generate button during processing to prevent duplicate requests.
- Added basic error handling for invalid or failed requests.
- Included example prompts to guide users in writing effective AI instructions.
- Updated the empty canvas message to promote AI-assisted diagram generation.
- Prepared the frontend workflow for future backend API integration.

10.3 Code Snippet 1

```
<Button

  className="border-gray-300 dark:border-gray-700 text-gray-300"

  size="sm"

  startContent={<span> ⚡ </span>}
```

```
  variant="bordered"
  onPress={() => setShowAIModal(true)}
>
  AI Generate
</Button>
```

10.4 Explanation

An AI Generate button was added to the editor toolbar, allowing users to access the AI-assisted diagram generation feature directly from the editor interface.

10.5 Code Snippet 2

```
<div className="text-xs text-gray-500 mb-3">
 <p className="font-semibold mb-1">Examples:</p>
 <p>• Pump → Heat Exchanger → Tank</p>
 <p>• Tank → Pump → Valve</p>
 <p>• Pump → Valve → Tank</p>
</div>
```

10.6 Explanation

Example prompts were introduced to guide users in writing valid natural language instructions. This helps improve usability and makes the AI feature easier to understand for first-time users.

10.7 Code Snippet 3

```
<button
   disabled={isGenerating}
   onClick={() => {
      setAiError("");
      setIsGenerating(true);


      ...
   }}
>
   {isGenerating ? "Generating..." : "Generate Diagram"}
</button>
```

10.8 Explanation

Loading and error handling were implemented to improve the overall user experience. The Generate button is disabled while processing requests, preventing duplicate submissions, and meaningful error messages are displayed whenever generation fails.

10.9 Outcome

The AI generation interface was significantly improved by introducing loading indicators, error handling, example prompts, and better interaction feedback. These enhancements created a complete and user-friendly frontend workflow, making the feature ready for seamless backend integration.

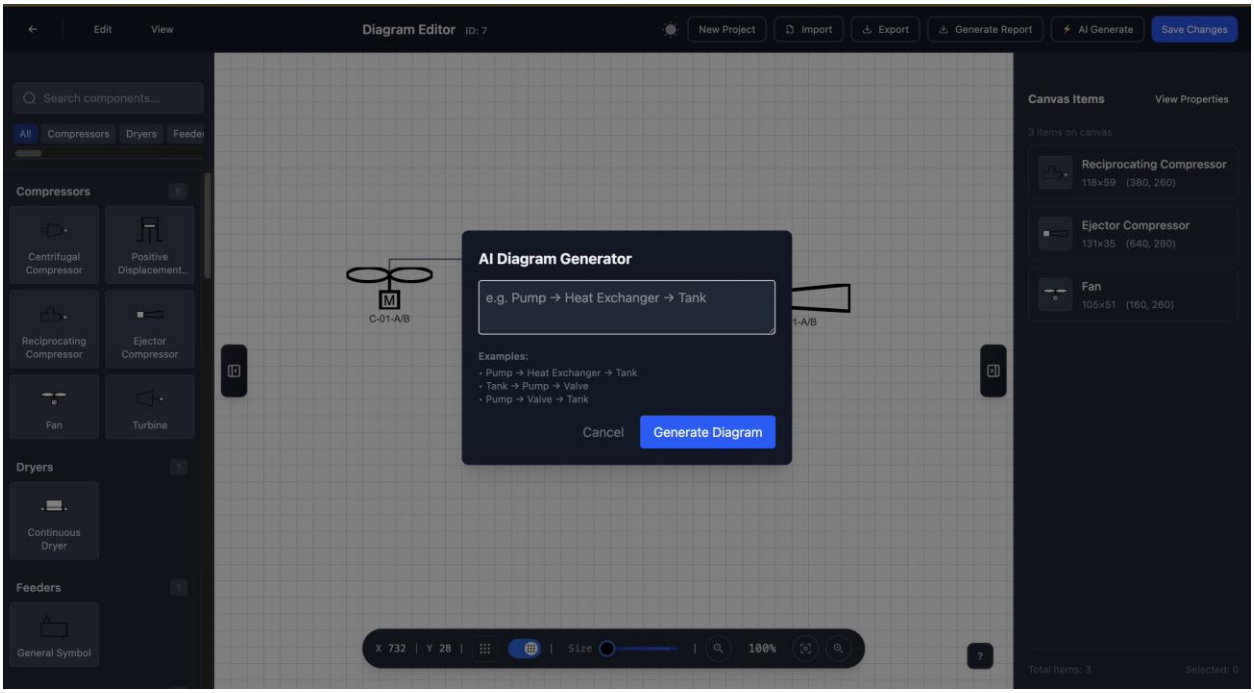


Figure 10.1: Updated empty canvas interface promoting AI-assisted diagram generation.

CHAPTER 11:

TASK 9: Fix AI Component Mapping, Variant Handling, and Connection Generation

11.1 Problem Statement

The initial AI diagram generation workflow faced issues while converting AI-generated component names into valid Process Flow Diagram (PFD) components. Different naming conventions such as “control valve” and “centrifugal pump” were not always matched correctly, resulting in missing components and incomplete connection chains. This affected the reliability of AI-generated diagrams.

11.2 Tasks Completed

The following improvements were implemented:

- Enhanced component mapping to support variant-based matching.
- Added normalization fallback to prevent components from being skipped.
- Improved connection generation for multiple components.
- Refined the AI generation workflow with better error handling.
- Ensured complete connection chains between generated components.

11.3 Code Snippet 1

```
const normalizeType = (text: string) => {
  const t = text.toLowerCase();
```

```
  if (t.includes("pump")) return "pump";
  if (t.includes("valve")) return "valve";
  if (t.includes("heat")) return "heat exchanger";
  if (t.includes("tank") || t.includes("vessel"))
    return "tank";

  return t;
};
```

11.4 Explanation

A normalization function was introduced to map different AI-generated component names to their corresponding Process Flow Diagram (PFD) components. This allowed variations such as “centrifugal pump”, “control valve”, and “vessel” to be correctly recognized and matched with the available component library.

11.5 Code Snippet 2

```
const match = allComps.find((c: any) => {
  const name = c.name?.toLowerCase() || "";
  const object = c.object?.toLowerCase() || "";

  return (
    name.includes(searchText) ||
    object.includes(searchText) ||
```

```
    name.includes(normalized) ||
    object.includes(normalized)
  );
});

...

generatedConnections.forEach((conn: any) => {
  const source = idMapping[conn.from];
  const target = idMapping[conn.to];

  if (!source || !target) return;

  editorStore.addConnection(projectId, {
    sourceItemId: source,
    targetItemId: target,
    sourceGripIndex: 0,
    targetGripIndex: 0,
    waypoints: [],
  });
});
```

11.6 Explanation

The component matching logic was enhanced to support both exact and normalized searches before placing components on the canvas. After successful matching,

connections were automatically generated between mapped components, ensuring that AI-generated process diagrams formed complete and valid process flows.

11.7 Outcome

- Improved AI component recognition.
- Reliable handling of component variants.
- Complete connection chains between generated components.
- Reduced chances of skipped components.
- Enhanced stability of AI-generated Process Flow Diagrams.

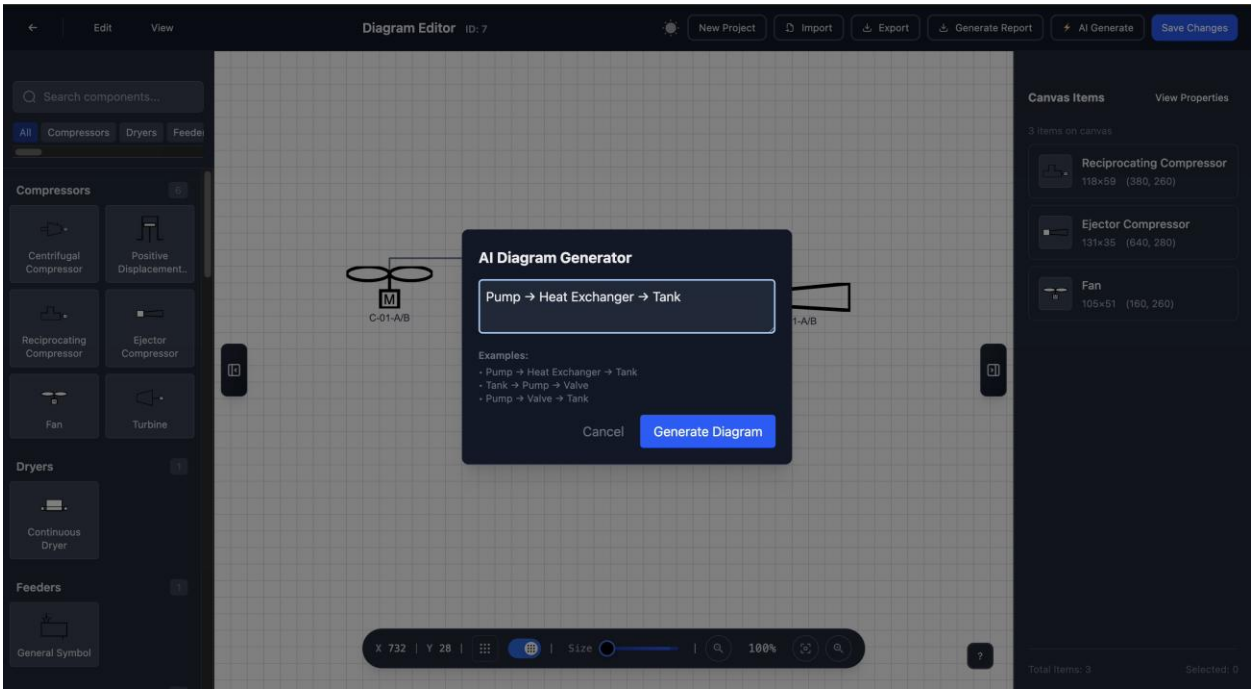


Figure 11.1: *AI Diagram Generation interface with user prompt.*

CHAPTER 12:
TASK 10: AI Smart Component Matching

12.1 Problem Statement

Improve AI-generated component selection by accurately matching user-described components with the available component library, reducing incorrect or missing mappings.

12.2 Tasks Completed

- Created a dedicated aiMatcher.ts utility for AI component matching.
- Implemented text normalization for generic component names.
- Added exact, partial, and scoring-based matching logic.
- Added fallback matching for unknown component names.
- Wrote unit tests using Vitest to verify normalization and matching behavior.

12.3 Code Snippet 1

```
export function normalizeType(text: string): string {
  const t = text.toLowerCase();


  if (t.includes("pump")) return "pump";
  if (t.includes("valve")) return "valve";
  if (t.includes("heat")) return "heat exchanger";
  if (t.includes("tank") || t.includes("vessel")) return "tank";
```

```
  return t;
}
```

12.4 Explanation

This function standardizes different user inputs into common component categories. For example, “centrifugal pump” and “reciprocating pump” are both normalized to “pump”, improving AI matching accuracy.

12.5 Code Snippet 2

```
// Exact match
if (name === searchText.toLowerCase() ||
    object === searchText.toLowerCase()) {
  score = 100;
}
// Partial match
else if (name.includes(searchText.toLowerCase()) ||
        object.includes(searchText.toLowerCase())) {
  score = 70;
}
// Normalized fallback
else if (name.includes(normalized) ||
        object.includes(normalized)) {
  score = 40;
}
```

12.6 Explanation

A scoring-based approach was introduced to prioritize exact matches while still supporting partial and normalized matches. This ensures the AI selects the most appropriate component even when the user's wording differs from the component library.

12.7 Outcome

- Improved AI component recognition accuracy.
- Reduced incorrect or missing component mappings.
- Introduced reusable matching logic for future AI enhancements.
- Added automated unit tests to improve code reliability.

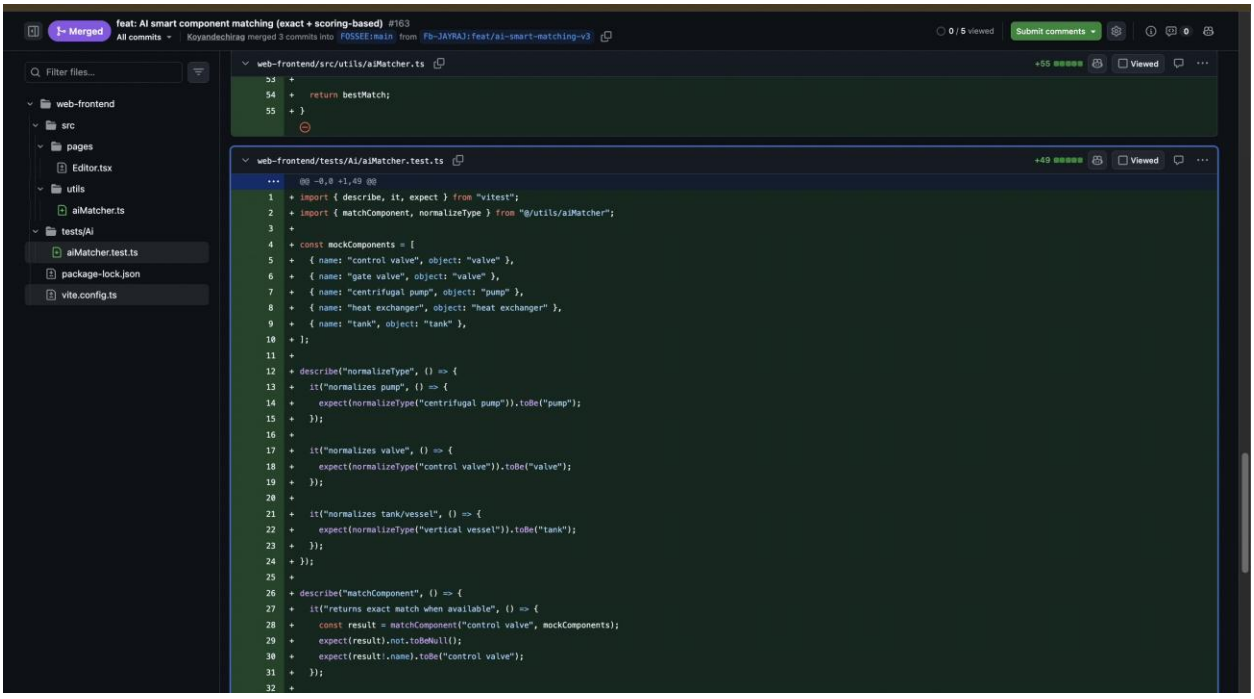


Figure 12.1: *Implementation of the AI Smart Component Matching utility and its corresponding unit tests for exact, partial, and normalized component matching.*

CHAPTER 13:

TASK 11: Preserve Undo/Redo History for Component Addition

13.1 Problem Statement

When a new component was added to the canvas, automatic image metadata updates (naturalWidth and naturalHeight) created an extra history entry. As a result, pressing Undo immediately after adding a component did not remove it correctly because the metadata update became the latest action in the history stack.

13.2 Tasks Completed

- Investigated the root cause of the incorrect undo/redo behavior.
- Added logic to distinguish metadata synchronization from actual user actions.
- Introduced an optional recordHistory flag in the editor store.
- Prevented metadata-only updates from creating history snapshots.
- Preserved undo/redo functionality for user actions such as add, move, resize, rotate, and delete.

13.3 Code Snippet 1

```
const isMetadataSync =
  "naturalWidth" in snappedUpdates ||
  "naturalHeight" in snappedUpdates;


editorStore.updateItem(
```

```
  projectId,
  itemId,
  snappedUpdates,
  !isMetadataSync,
);
```

13.4 Explanation

This logic checks whether the update only modifies image metadata. If so, the update is performed without recording a new history snapshot, preventing unnecessary undo entries.

13.5 Code Snippet 2

```
updateItem: (
  editorId,
  itemId,
  patch,
  recordHistory = true,
) => {
    ...

const snapshot = recordHistory
  ? createSnapshot(ed)
  : null;
```

13.6 Explanation

A new recordHistory parameter was introduced to control when history snapshots should be created. User actions continue to be recorded, while internal metadata synchronization skips history creation.

13.7 Outcome

The undo/redo system now behaves correctly after adding new components. Metadata synchronization no longer creates unnecessary history entries, allowing users to undo component additions in a single step while preserving complete history for all meaningful editing operations.

CHAPTER 14:

TASK 12: Component Image Resizing with Bounding Box

14.1 Problem Statement

Previously, resizing a component only changed the bounding box dimensions while the rendered image maintained its original aspect ratio. This caused the image and selection box to become misaligned, resulting in an inconsistent editing experience.

14.2 Tasks Completed

- Updated component rendering to resize the image together with the bounding box.
- Removed aspect-fit rendering during interactive resizing.
- Enabled independent width and height resizing.
- Ensured labels remain correctly positioned after resizing.
- Verified compatibility with drag-and-drop, selection, and undo/redo operations.

14.3 Code Snippet 1

```
<KonvaImage
  height={item.height}
  width={item.width}
  x={0}
  y={0}
/>
```

14.4 Explanation

The rendered image now directly uses the component's updated width and height, ensuring that the visual representation always matches the resized bounding box.

14.5 Outcome

Component resizing now behaves intuitively, with the image stretching together with the bounding box during width, height, or corner resizing. This provides a consistent editing experience while maintaining correct label positioning and compatibility with existing editor features.

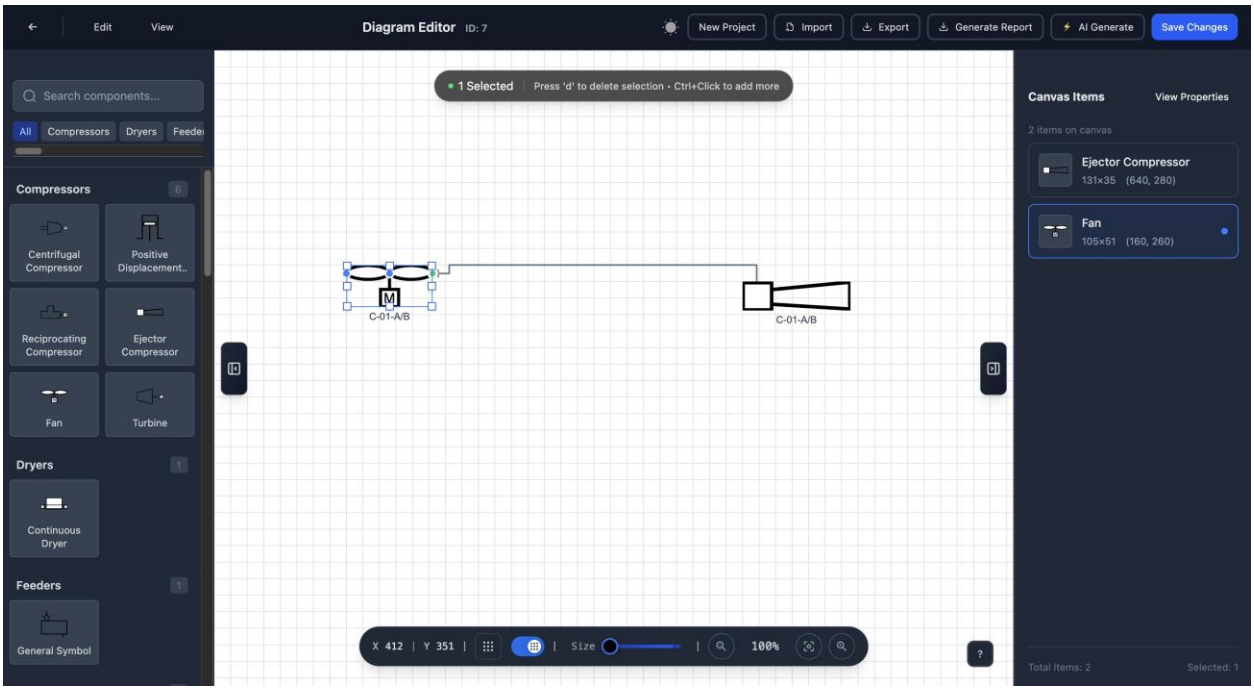


Figure 14.1: *Implementation of synchronized image and bounding box resizing for canvas components.*

CHAPTER 15: Conclusion

15.1 Tasks Accomplished

During the **FOSSEE Semester Long** Internship, I contributed to the development of the **Chemical PFD Builder**, a web-based application for creating and editing Process Flow Diagrams (PFDs). My contributions focused on improving editor functionality, usability, AI-assisted features, process validation, and overall user experience.

The major tasks completed during the internship include:

- Successfully completed the Screening Task involving a hybrid React.js and Django application for chemical equipment data visualization and analytics.
- Fixed the initial component scaling issue that occurred when dropping the first component onto the canvas.
- Designed and implemented a **Graph-Based Process Validation Engine** capable of detecting isolated components, circular loops, broken process flows, and validating inlet and outlet connectivity.
- Added support for **draggable waypoints** in orthogonal connection lines, allowing users to manually adjust routing while preserving automatic obstacle avoidance.
- Improved dashboard interaction by resolving nested button rendering issues and fixing project opening/navigation from the dashboard.
- Designed and implemented the frontend UI for **AI-based diagram generation**, including toolbar integration, input modal, loading indicators, validation messages, and user guidance.
- Enhanced AI component mapping by introducing normalized matching, variant handling, fallback strategies, and reliable connection generation.
- Developed a reusable **AI Smart Component Matching** module using exact matching, partial matching, normalized matching, and scoring-based selection, along with comprehensive unit tests.
- Improved the editor’s undo/redo mechanism by preventing internal metadata synchronization from creating unnecessary history entries.

- Updated component rendering so that images resize together with their bounding boxes, providing a smoother and more intuitive editing experience.

Throughout the internship, I actively participated in GitHub-based development, submitted multiple pull requests, collaborated with mentors during code reviews, resolved reported issues, and contributed to the continuous improvement of the Chemical PFD Builder project.

15.2 Skills Developed

Technical Skills

This internship strengthened my practical knowledge of modern web development using **React.js**, **TypeScript**, and **Vite**. I gained hands-on experience with **Konva.js** for interactive canvas development, implemented graph-based algorithms such as **Depth First Search (DFS)** and **Breadth First Search (BFS)** for process validation, and improved my understanding of scalable frontend architecture.

I also learned to integrate AI-assisted features into user interfaces, implement reusable utility modules, write unit tests using **Vitest**, manage application state effectively, debug complex frontend issues, and collaborate through Git and GitHub in a real open-source development environment.

Professional Skills

Working as 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 in understanding existing codebases, implementing production-ready features, debugging real-world issues, documenting technical work, and following structured software development practices.

The fellowship also enhanced my ability to analyze problems, break them into manageable tasks, and deliver reliable solutions while maintaining code quality and readability.

Future Scope

The Chemical PFD Builder can be further enhanced through several future improvements:

- Complete backend integration for AI-powered process flow generation.
- Introduce additional process validation rules based on chemical engineering standards.
- Improve AI component recognition using semantic search and Large Language Models.
- Support real-time collaborative editing using WebSocket-based synchronization.
- Expand the chemical component library with additional industrial symbols.
- 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 valuable hands-on experience in open-source software development, modern frontend engineering, collaborative development workflows, and real-world software problem solving. Working on the Chemical PFD Builder allowed me to apply theoretical knowledge to practical engineering software while significantly improving my technical and professional skills. The experience has strengthened my confidence in contributing to large-scale software projects and has laid a solid foundation for my future career in software development.

CHAPTER 16: Bibliography

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

[2] Chemical PFD Builder Repository. *GitHub Repository.* https://github.com/FOSSEE/Chemical-PFD-Web-Desktop