FOSSEE Winter Internship Report

On

Development of Chemical PFD Builder

Submitted by

Harshit Kandpal

2nd Year B.Tech Student,CSE

IIT Bhilai

Under the Guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

Indian Institute of Technology Bombay

Mentors:

Chirag Koyande

February 21, 2026

Acknowledgments

I would like to express my sincere gratitude to the FOSSEE team at IIT Bombay for the opportunity to participate in the Winter 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 a browser-based editor using React.js and Konva.js. Beyond frontend development, I managed GitHub repositories, created Swagger API documentation, and implemented tests for the front-end. These responsibilities provided me with a holistic understanding of full-stack architecture and the lifecycle of open-source software.

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

Special thanks to my mentor, Chirag Koyande, for his invaluable guidance and technical clarity. I also appreciate the support as well as the 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

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), 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 software system developed under the FOSSEE project for creating, editing, and managing Process Flow Diagrams (PFDs) commonly used in chemical engineering. The tool is designed to cater to both academic and industrial needs, providing a professional-grade diagramming environment with cloud-based project storage and seamless access across web and desktop platforms.

1.2.1 Overview and Purpose

Process Flow Diagrams are essential in chemical engineering for visualizing processes, equipment, and material flows. Traditional PFD creation often relies on proprietary software, which can be expensive and restrict collaboration. Chemical PFD Builder addresses

this by offering a free and open-source alternative that runs on multiple platforms and supports modern workflows.

1.2.2 System Architecture

Chemical PFD Builder follows a hybrid architecture with a common backend serving both web and desktop frontends. This design ensures consistency, scalability, and flexibility across platforms. The backend provides RESTful APIs for user authentication, project management, and diagram storage, while the web and desktop frontends offer tailored interfaces for different usage scenarios. All modules share a common data schema, enabling seamless exchange and cross-platform project continuity.

1.2.3 Key Features

The software enables users to create PFDs through an intuitive drag-and-drop interface, with support for a wide range of chemical engineering symbols. Components can be automatically labeled using CSV or Excel files, streamlining the creation of large diagrams. The system generates comprehensive design reports in PDF and Excel formats, including component properties and stream tables. With cloud synchronization, diagrams can be saved online and accessed from any device, facilitating collaboration. Export options include high-resolution images, vector graphics, and CAD formats for integration with drafting tools.

1.2.4 Evolution from Legacy Tool

Chemical PFD Builder builds upon an earlier version of the software, originally developed as a standalone desktop application. The legacy tool provided basic drag-and-drop functionality and file-based saving. The new version addresses limitations by introducing a shared backend, a web frontend, and enhanced features such as auto-labeling, report generation, and cloud storage, while preserving the desktop application's performance for complex engineering tasks.

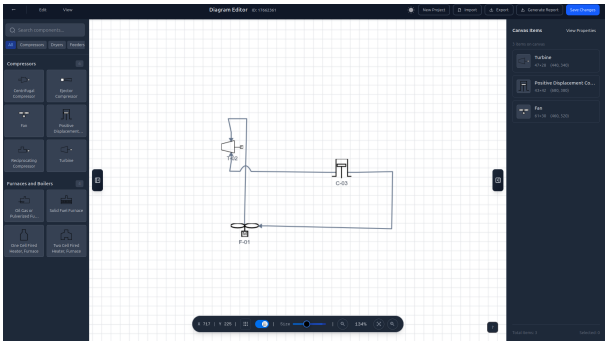

Figure 1.2: Evolution from Legacy Desktop Tool to Dekstop+Web System. The web app is shown in image

1.2.5 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.

1.2.6 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

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.

2.1.2 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

2.2 Tasks Done

The completed solution demonstrates seamless integration between Django backend, React web frontend, and PyQt5 desktop application.

2.2.1 System Architecture

- **Backend:** Django REST API handling CSV uploads, data processing with Pandas, history management in SQLite, and authentication
- **Web Frontend:** React.js with Chart.js for responsive dashboard and visualizations
- **Desktop Frontend:** PyQt5 with Matplotlib providing native desktop experience

2.2.2 Features Implemented

All required features were successfully implemented:

- CSV upload functionality in both interfaces
- RESTful endpoints returning comprehensive statistics
- Interactive charts (Chart.js) and Matplotlib plots
- History tracking of last 5 datasets
- PDF report generation with summaries and charts
- User authentication with protected routes

2.2.3 Deployment

The application can be run using:

- **Docker:** Single command deployment with Docker Compose
- **Manual:** Individual setup for backend, web, and desktop
- **Desktop Script:** Automated script for desktop app launch

The web application was deployed on Render.

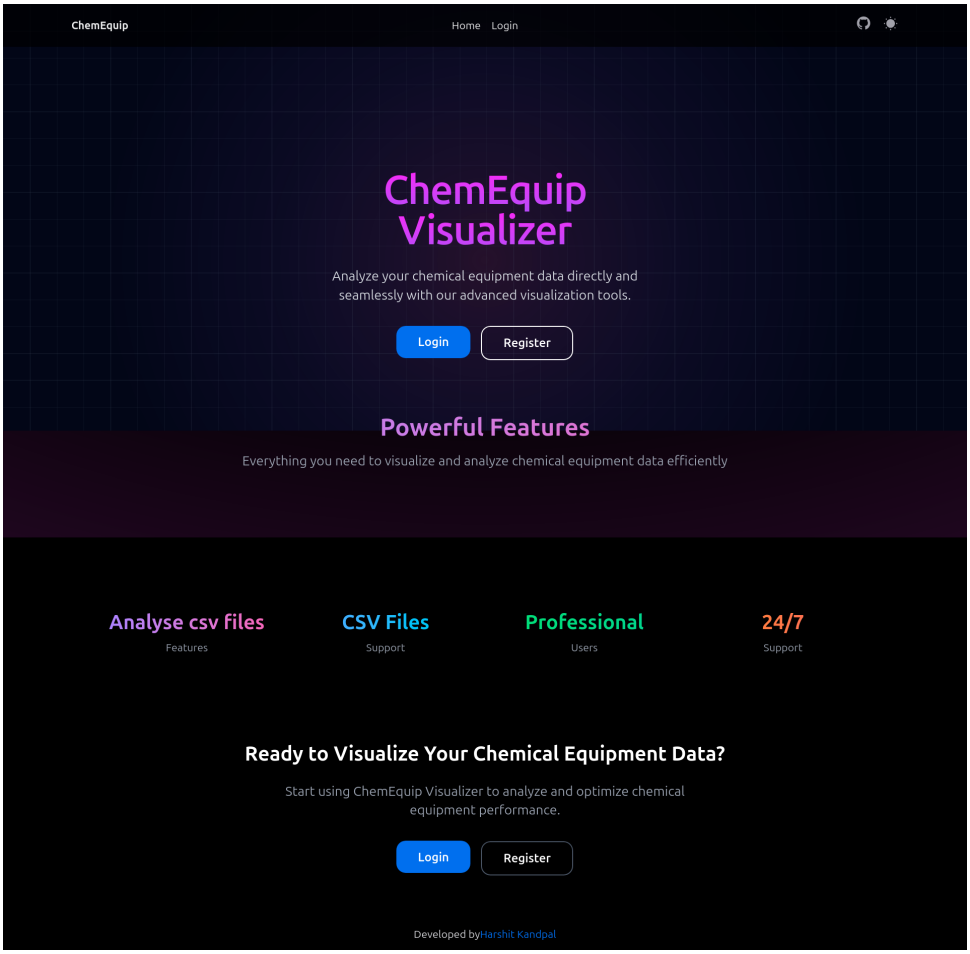

Figure 2.1: Web Application Dashboard

2.2.4 Conclusion

The screening task resulted in a fully functional hybrid application meeting all requirements, demonstrating proficiency in Django, React, PyQt5, and API integration. The complete source code, documentation, and demo video were submitted for evaluation.

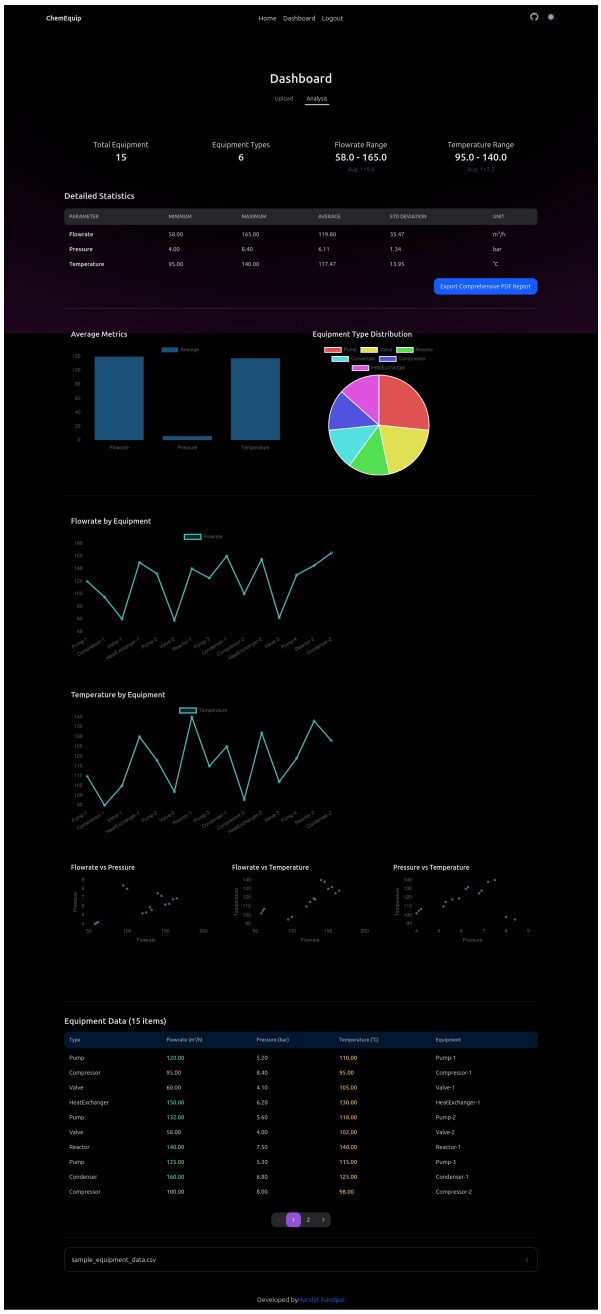

Figure 2.2: Analysis and Visualization Tab

Chapter 3

Task 1: Project Setup & Foundation

3.1 Problem Statement

Establish a modern, scalable frontend architecture for the Chemical PFD Editor. The foundation must support TypeScript, theming, routing, and a clean folder structure to enable iterative feature development.

3.2 Tasks Done

The following setup tasks were completed in `PR #7`:

- Initialised Vite + React + TypeScript project.
- Configured TailwindCSS with HeroUI plugin.
- Set up folder structure (`components`, `pages`, `store`, `utils`, `types`).
- Added basic routing placeholders (Login, Dashboard, Editor, Reports).
- Implemented light/dark theme provider with `@heroui/use-theme`.

3.2.1 Outcome

A clean foundation ready for iterative development, ensuring consistency and scalability across the application.

Chapter 4

Task 2: Dark Mode & Dashboard Search/-Filter

4.1 Problem Statement

Enhance the user interface by introducing theme switching (dark/light mode) and improving the dashboard's usability with real-time search, sort, and filter capabilities. The UI must be accessible, visually consistent, and performant.

4.2 Tasks Done

This task was completed in Pull Request #10 and includes two major improvements: a global dark mode toggle integrated into a new navbar component, and an interactive search/filter system for the project dashboard.

4.2.1 Dark Mode Toggle & Navigation

A new `CNavbar` component was created using HeroUI's `Navbar`, `Popover`, and `Avatar` components. The navbar includes:

- A `ThemeSwitch` component that toggles between light and dark themes instantly.
- Semantic color tokens (e.g., `bg-background`, `text-foreground`) instead of hard-coded values, ensuring consistency across themes.

- A popover menu for user profile and settings, enhancing the overall navigation experience.

4.2.2 Dashboard Search and Filter

The project dashboard was upgraded with:

- **Real-time search**: Projects are filtered by title as the user types, using a case-insensitive match.
- **Sort options**: Users can sort projects alphabetically or by most recent (based on a mock date field).
- **Size filter**: A simple filter to show only projects of a certain size category (small, medium, large) – implemented with mock criteria for demonstration.

To ensure performance, the filtering logic is wrapped in `useMemo`, preventing unnecessary re-computations on every render.

4.2.3 Code Implementation

The following code snippet illustrates the filtering and sorting logic used in the dashboard:

Listing 4.1: Filtering Logic with useMemo

```
const filteredProjects = useMemo(() => {
  let list = [...projects];

  // Search filter
  if (search.trim()) {
    list = list.filter(p =>
      p.title.toLowerCase().includes(search.toLowerCase())
    );
  }

  // Sort options
  if (sortBy === 'alpha') {
    list.sort((a, b) => a.title.localeCompare(b.title));
```

```
  } else if (sortBy === 'recent') {
    list.sort((a, b) => (b.updatedAt || 0) - (a.updatedAt || 0));
  }

  // Size filter (mock criteria)
  if (sizeFilter !== 'all') {
    list = list.filter(p => p.size === sizeFilter);
  }

  return list;
}, [search, sortBy, sizeFilter, projects]);
```

4.2.4 Outcome

The dark mode toggle provides a seamless theme transition across the entire application, improving accessibility and user comfort. The dashboard's search and filter features significantly enhance the user experience by allowing quick access to relevant projects, all while maintaining high performance through memoization.

Chapter 5

Task 3: Canvas Engine – Konva Integration

5.1 Problem Statement

Implement a performant diagram canvas that supports drag-and-drop of chemical engineering symbols, resizing, rotation, selection, connection grips, grid system, and keyboard shortcuts. The canvas must handle complex interactions while maintaining smooth rendering and accurate coordinate transformations.

5.2 Tasks Done

This task was completed across multiple pull requests (#20, #31, #34, #40, #51, #88), each adding critical functionality to the canvas engine.

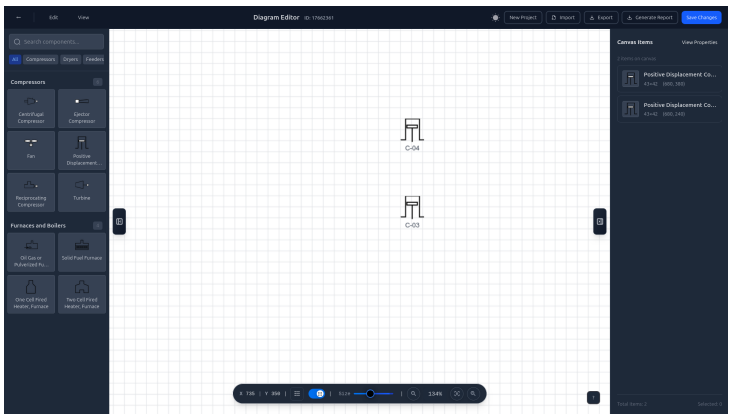

Figure 5.1: Canvas Editor Interface

5.2.1 Core CanvasItemImage Component (PR #20)

Objective

Render draggable, resizable SVG components with a selection transformer.

Technical Challenges

- Konva transformer requires attaching nodes to the transformer ref.
- Scale must be applied to width/height, not to the node itself, to maintain accurate dimensions.

Implementation

Listing 5.1: Transformer Setup for Draggable Components

```
const shapeRef = useRef<Konva.Image>(null);
const trRef = useRef<Konva.Transformer>(null);

useEffect(() => {
  if (isSelected && trRef.current && shapeRef.current) {
    trRef.current.nodes([shapeRef.current]);
    trRef.current.getLayer()?.batchDraw();
  }
}, [isSelected]);

const handleTransformEnd = () => {
  const node = shapeRef.current;
  if (!node) return;
  const scaleX = node.scaleX(), scaleY = node.scaleY();
  node.scaleX(1); node.scaleY(1);
  onChange({
    ...item,
    x: node.x(), y: node.y(),
    width: node.width() * scaleX,
```

```
    height: node.height() * scaleY,
    rotation: node.rotation()
  });
};
```

5.2.2 Grip System for Connection Points (PR #34)

Problem

Component grip definitions use bottom-left origin (Y=0 at bottom, Y=100 at top), but Konva uses top-left origin.

Solution

Invert Y coordinate during rendering.

Listing 5.2: Grip Coordinate Transformation

```
item.grips?.map((grip) => {
  const gripX = item.x + (grip.x / 100) * item.width;
  const gripY = item.y + ((100 - grip.y) / 100) * item.height;
  return <Circle x={gripX} y={gripY} radius={6} fill="#3b82f6"
     />;
});
```

5.2.3 Grid System & Snap-to-Grid (PR #51)

High-Performance Grid Rendering

Used memoised Konva `Shape` with `perfectDrawEnabled=false` for crisp lines. Grid lines extend far beyond viewport to avoid clipping.

Listing 5.3: GridLayer Component

```
const GridLayer = React.memo(({ width, height, gridSize, showGrid
    }) => {
  if (!showGrid) return null;
  return (
```

```
    <Layer listening={false}>
      <Shape
        stroke="#9ca3af"
        strokeWidth={1}
        opacity={0.3}
        perfectDrawEnabled={false}
        sceneFunc={(ctx, shape) => {
          ctx.beginPath();
          for (let x = -5000; x <= width + 5000; x += gridSize) {
            ctx.moveTo(x, -5000);
            ctx.lineTo(x, height + 5000);
          }
          // ... vertical lines
          ctx.fillStrokeShape(shape);
        }}
      />
    </Layer>
  );
});
```

Snap-to-Grid Logic

Applied to drop, single move, and batch move operations. Used per-item snapping to preserve relative positions during multi-select.

Listing 5.4: Batch Move with Snapping

```
const batchUpdates = selectedItems.map((item) => ({
  id: item.id,
  patch: {
    x: snapToGrid
      ? snapToGridPosition(item.x + deltaX, item.y + deltaY).x
      : item.x + deltaX,
    y: snapToGrid
      ? snapToGridPosition(item.x + deltaX, item.y + deltaY).y
      : item.y + deltaY,
```

```
  },
}));
```

5.2.4 Keyboard Shortcuts & Help System (PR #40)

Objective

Standardise shortcuts and improve discoverability.

Problem

Previously shortcuts were ad-hoc, conflicted with browser defaults, and used different modifier keys.

Solution

Centralised all shortcuts in a registry array. Required `Ctrl/Cmd` for every action to avoid accidental triggers. Added a floating help popover displaying all shortcuts.

Listing 5.5: Shortcut Registry

```
const shortcuts: Shortcut[] = [
  {
    key: "z",
    label: "Undo",
    display: "Ctrl+Z",
    requireCtrl: true,
    handler: undo,
  },
  {
    key: "g",
    label: "Toggle Grid",
    display: "Ctrl+G",
    requireCtrl: true,
    handler: handleToggleGrid,
  },
  // ... others
```

```
];

useEffect(() => {
  const handleKeyDown = (e: KeyboardEvent) => {
    const key = e.key.toLowerCase();
    for (const s of shortcuts) {
      if (
        (key === s.key ||
          (s.key === "delete" && (key === "delete" || key === "
             backspace"))) &&
        (!s.requireCtrl || e.ctrlKey || e.metaKey)
      ) {
        e.preventDefault();
        s.handler();
        return;
      }
    }
  };
  window.addEventListener("keydown", handleKeyDown);
  return () => window.removeEventListener("keydown",
     handleKeyDown);
}, [shortcuts]);
```

5.2.5 Enhancements: Zoom, Layout, and Export

Key features were implemented to improve the user experience and output quality:

- **Zoom Controls**: Bounds-limited scaling from 10% to 300%.
- **Center-to-Content**: Automatic viewport adjustment based on item coordinates.
- **Export System**: Support for PNG, JPG, PDF, and SVG with presets for Presentation and Print.

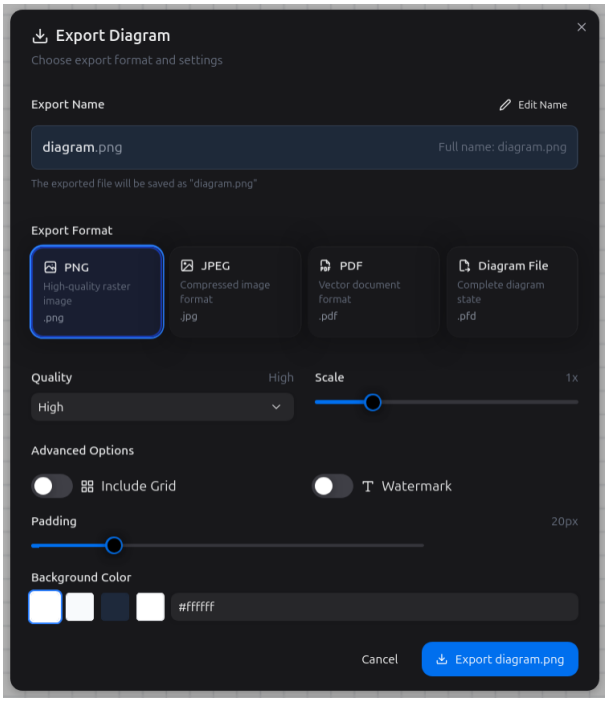

Figure 5.2: Export modal showing format options and quality presets

5.2.6 Outcome

The canvas engine now provides a professional-grade experience, successfully bridging the gap between browser-based interactions and precise engineering requirements. Key achievements include accurate grip positioning via coordinate inversion and high-performance grid rendering using memoized Konva shapes. The canvas engine now provides a professional-grade diagramming experience with:

- Smooth drag-and-drop of components
- Accurate grip positioning for connections
- High-performance grid rendering
- Snap-to-grid functionality for precise alignment
- Comprehensive keyboard shortcuts
- Multi-format export capabilities
- Responsive zoom and pan controls

Chapter 6

Task 4: Global State Management with Zustand

6.1 Problem Statement

Replace local React component state with a scalable, multi-editor-aware global store that provides undo/redo capabilities, cross-component state synchronization, and persistent history across editor instances.

6.2 Tasks Done

This task was completed in Pull Requests #34 and #51, establishing a robust state management system using Zustand.

6.2.1 Before: Fragmented Local State Management

Prior to this implementation, the editor used:

- Local `useState` hooks in `Editor.tsx`
- A custom `useHistory` hook wrapping state setters
- Prop drilling through multiple component layers
- No shared state between different editor tabs

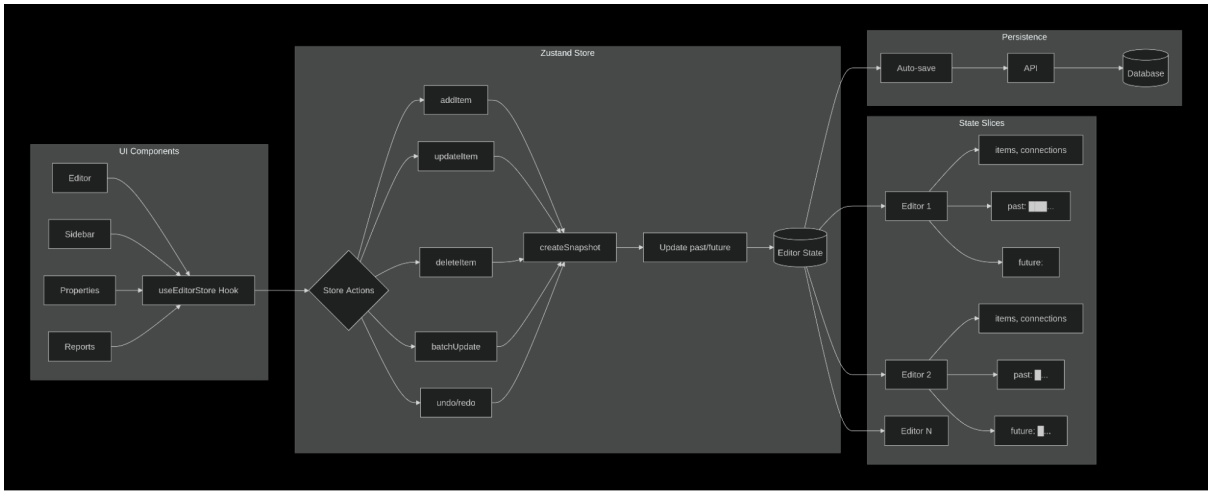

Figure 6.1: Zustand store architecture showing editor instances with independent history stacks

6.2.2 Critical Issues Identified

Problem	Impact
History lost on unmount	Switching tabs erased undo stack
Multiple tabs unsupported	Each tab had independent, non-shareable state
Batch ops not undoable	Multi-select moves couldn't be reverted
Counters desynchronized	Item counts inconsistent across components
No centralized logic	Validation, tag generation duplicated
Memory leaks	No cleanup of editor instances

6.2.3 Solution: Zustand Store with History

Store Design Pattern

The Zustand store implements a slice pattern where each editor instance maintains its own state slice with embedded history buffers.

Snapshot-Based History System

The history implementation uses immutable snapshots rather than inverse operations.

Listing 6.1: Snapshot Definition

```
type EditorSnapshot = Pick<CanvasState,
  "items" | "connections" | "counts" | "sequenceCounter"
```

```
>;
```

Mutation Pattern with Automatic History

Every mutation action follows a consistent pattern:

Listing 6.2: addItem with Automatic History

```
addItem: (editorId, component, opts) => {
  const editor = get().editors[editorId];
  if (!editor) return undefined;

  // 1. CAPTURE SNAPSHOT
  const snapshot = createSnapshot(editor);
  const newPast = [...editor.past, snapshot];

  // 2. GENERATE BUSINESS DATA
  const key = component.object?.trim() || component.name.trim();
  const currentCount = editor.counts[key] ?? 0;
  const nextCount = currentCount + 1;
  const label = `${component.legend}-${padCount(nextCount)}${
     component.suffix ? `-${component.suffix}` : ''}`;

  // 3. CREATE NEW ITEM
  const newItem: CanvasItem = {
    ...component,
    id: ++globalIdCounter,
    sequence: (editor.sequenceCounter ?? 0) + 1,
    x: opts.x ?? 100,
    y: opts.y ?? 100,
    width: opts.width ?? 80,
    height: opts.height ?? 40,
    rotation: opts.rotation ?? 0,
    label,
  };
```

```
  // 4. APPLY MUTATION + UPDATE HISTORY
  set(state => ({
    editors: {
      ...state.editors,
      [editorId]: {
        ...state.editors[editorId],
        items: [...state.editors[editorId].items, newItem],
        counts: { ...state.editors[editorId].counts, [key]:
           nextCount },
        sequenceCounter: newItem.sequence,
        past: newPast,
        future: [],
      },
    },
  }));
  return newItem;
}
```

Undo/Redo Implementation

The undo/redo actions swap snapshots between past and future stacks.

Listing 6.3: Undo Implementation

```
undo: (editorId) =>
  set((s) => {
    const ed = s.editors[editorId];
    if (!ed || ed.past.length === 0) return s;

    const previous = ed.past[ed.past.length - 1];
    const newPast = ed.past.slice(0, ed.past.length - 1);
    const currentSnapshot = createSnapshot(ed);

    return {
      editors: {
        ...s.editors,
```

```
        [editorId]: {
          ...ed,
          ...previous,
          past: newPast,
          future: [currentSnapshot, ...ed.future],
        },
      },
    };
  }),
```

Batch Operations with Single History Entry

Previously, moving 10 items would create 10 history entries. Now, one snapshot captures all changes.

Listing 6.4: batchUpdateItems

```
batchUpdateItems: (editorId, updates) =>
  set((state) => {
    const ed = state.editors[editorId];
    if (!ed) return state;

    const snapshot = createSnapshot(ed);
    const nextItems = [...ed.items];

    updates.forEach(({ id, patch }) => {
      const index = nextItems.findIndex((i) => i.id === id);
      if (index !== -1) nextItems[index] = { ...nextItems[index],
          ...patch };
    });

    return {
      editors: {
        ...state.editors,
        [editorId]: {
          ...ed,
```

```
          items: nextItems,
          past: [...ed.past, snapshot],
          future: [],
        },
      },
    };
  });
```

6.2.4 Technical Challenges

Challenge 1: Circular References

Problem: Canvas items contained references to parent editor objects, causing serialization cycles.

Solution: Strictly typed snapshots with shallow copies.

```
// BEFORE - BROKEN
const snapshot = { ...editor }; // Contains circular refs

// AFTER - FIXED
const snapshot = {
  items: editor.items.map((item) => ({ ...item })),
  connections: [...editor.connections],
  counts: { ...editor.counts },
  sequenceCounter: editor.sequenceCounter,
};
```

Challenge 2: Memory Leaks

Problem: Store retained editor state after tab closure.

Solution: `removeEditor` action on unmount.

```
useEffect(() => {
  return () => {
    if (projectId) {
      useEditorStore.getState().removeEditor(projectId);
```

```
    }
  };
}, [projectId]);
```

Challenge 3: Performance

Problem: Deep cloning caused degradation with 100+ items.

Solution: Shallow copies and debounced auto-save.

Challenge 4: Type Safety

Problem: Accessing non-existent editor IDs caused runtime errors.

Solution: Guard clauses throughout.

```
getItemsInOrder: (editorId) => {
  const ed = get().editors[editorId];
  if (!ed) return [];
  return [...ed.items].sort((a, b) => a.sequence - b.sequence);
};
```

6.2.5 Benefits

Benefit	Before	After
Multi-editor support	Impossible	Each tab independent
Undo across tabs	Lost on switch	Preserved per tab
Batch operation undo	10 steps	1 step
State synchronization	Prop drilling	Direct store access
Tag number consistency	Race conditions	Atomic counters
Memory management	No cleanup	`removeEditor`

Chapter 7

Task 5: Export/Import System

7.1 Problem Statement

Enable saving and loading of complete diagram state, and export diagrams to various formats (PNG, JPG, PDF, SVG) with customizable options.

7.2 Tasks Done

This task was completed across pull requests (#63, #67, #88), implementing a comprehensive export/import system.

7.2.1 Versioned Diagram File Format (PR #63)

Listing 7.1: File Format Schema

```
export interface DiagramFileFormat {
  version: string;
  metadata: { name; description; createdAt; updatedAt };
  canvas: { width; height; backgroundColor; grid; viewport };
  items: CanvasItem[];
  connections: Connection[];
}
```

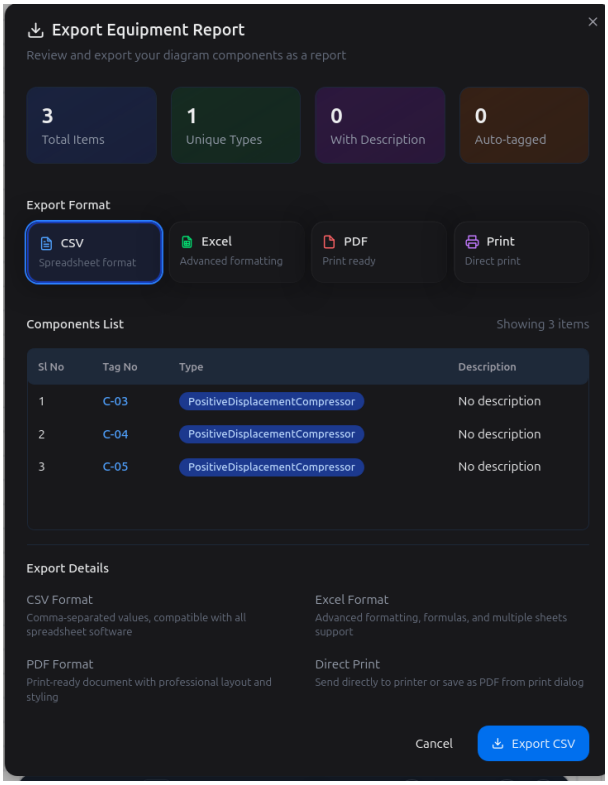

Figure 7.1: Enter Caption

7.2.2 Export Presets (PR #67)

Four presets available:

- **Presentation**: PNG, 2x scale
- **Print**: PDF, 3x scale
- **Web**: JPG, 1x scale
- **Dark**: PNG, dark background

7.2.3 Filename Customization

Listing 7.2: Filename Sanitisation

```
const handleNameChange = (e) => {
  let newName = e.target.value;
  const formatExt = currentFormat?.extension;
  if (formatExt && newName.endsWith(formatExt)) {
    newName = newName.slice(0, -formatExt.length);
```

```
  }
  const sanitized = newName.replace(/[<>:"/\\|?*]/g, "");
  setFilename(sanitized);
};
```

7.2.4 Data-Driven Export (PR #88)

Critical Bug

Cloning the live stage for exports caused missing connections and UI artifacts.

Root Cause

Live stage contained selection rings, grips, and rasterised connection lines.

Solution

Build fresh stage from source data, ignoring UI state.

Listing 7.3: Clean Stage Construction

```
export async function exportToImage(stage, options, items,
   connections) {
  const bounds = calculateContentBounds(items, options.padding);
  const container = document.createElement("div");

  const exportStage = new Konva.Stage({
    container,
    width: bounds.width,
    height: bounds.height,
    listening: false,
  });

  // Background layer
  const bgLayer = new Konva.Layer();
  bgLayer.add(new Konva.Rect({
    x: 0, y: 0,
```

```
    width: bounds.width,
    height: bounds.height,
    fill: options.backgroundColor,
  }));
  exportStage.add(bgLayer);

  // Items layer     render from SVG source
  items.forEach((item) => {
    // Convert SVG string to Konva.Image
  });

  // Connections layer     recalculate paths
  const paths = calculateManualPathsWithBridges(connections,
     items);
}
```

7.2.5 Outcome

The export/import system provides:

- Reliable saving/loading with versioning
- Multiple export formats with presets
- Artifact-free exports without UI elements
- User-friendly filename handling

Chapter 8

Task 6: Backend Integration & API Client

8.1 Problem Statement

Connect the frontend to the Django REST backend for project persistence, authentication, and cloud sync.

8.2 Tasks Done

This task was completed in Pull Requests #76 and #80, establishing a robust API client and auto-save mechanism.

8.2.1 API Client Implementation

Configured Axios instance with authentication token interceptor.

Listing 8.1: API Client Setup

```
const client = axios.create({ baseURL: "http://localhost:8000/api
    " });


client.interceptors.request.use((config) => {
  const token = localStorage.getItem("access_token");
  if (token) config.headers.Authorization = `Bearer ${token}`;
```

```
  return config;
});

export const fetchProjects = () =>
  client.get("/project/").then(res => res.data.projects);

export const saveProjectCanvas = (id, payload) =>
  client.put(`/project/${id}/`, payload);
```

8.2.2 State Conversion Utility

Converts frontend `CanvasItem[]` and `Connection[]` to backend-friendly format, mapping local item IDs to `component_id`.

8.2.3 Auto-Save Mechanism

Debounced effect (2 seconds) on store changes saves only when project ID exists and changes are detected.

Listing 8.2: Auto-Save Implementation

```
useEffect(() => {
  if (!projectId || !currentState) return;

  const timeoutId = setTimeout(async () => {
    const backendState = convertToBackendFormat(
      projectId,
      currentState.items,
      currentState.connections,
      currentState.sequenceCounter
    );
    await saveProjectCanvas(projectId, { canvas_state:
       backendState });
  }, 2000);

  return () => clearTimeout(timeoutId);
```

```
}, [currentState, projectId]);
```

8.2.4 Known Issue

`component_id` currently sends the local item UUID instead of the original library ID – flagged for backend schema adjustment.

Chapter 9

Task 7: Testing Infrastructure

9.1 Problem Statement

Move from zero test coverage to a comprehensive testing framework covering critical pages and components.

9.2 Tasks Done

This task was completed in Pull Request #84, establishing a robust testing infrastructure.

9.2.1 Infrastructure Setup

- Vitest + jsdom environment
- React Testing Library for component tests
- Global setup with `@testing-library/jest-dom`
- Extensive mocking of API, router, and UI components

9.2.2 Component Test Example – Login

Listing 9.1: Login Page Test

```
describe("Login Page", () => {
  it("calls loginUser with correct credentials", async () => {
```

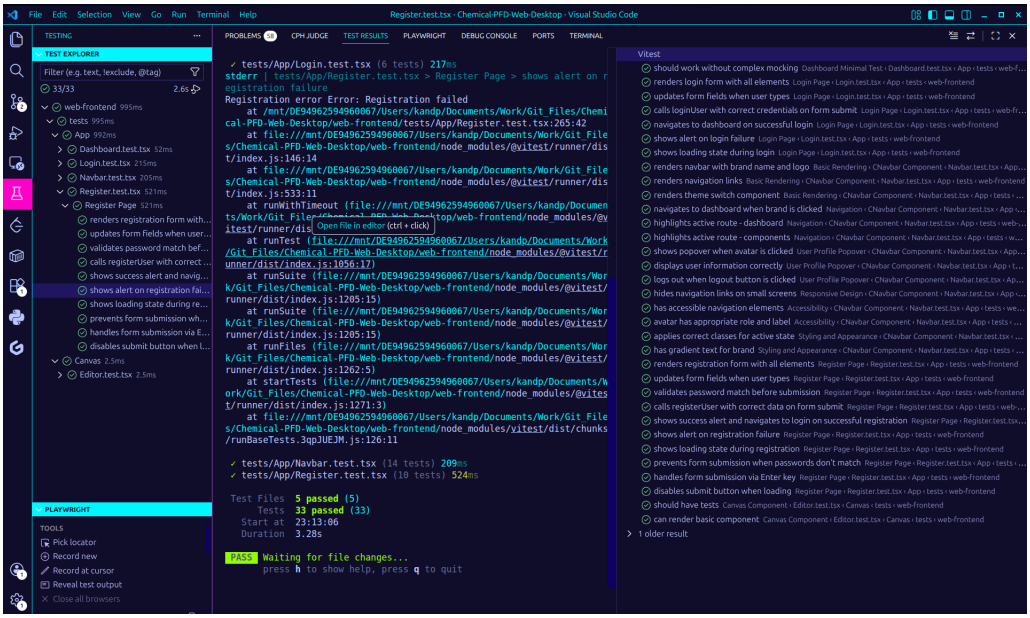

Figure 9.1: VS Code showing test files and passing test suites for Login, Register, and Navbar components

```
    mockedLoginUser.mockResolvedValue({ token: "test-token" });

    render(
      <MemoryRouter>
        <Login />
      </MemoryRouter>
    );

    fireEvent.change(screen.getByPlaceholderText(/username/i), {
      target: { value: "testuser" },
    });
    fireEvent.change(screen.getByPlaceholderText(/password/i), {
      target: { value: "testpass" },
    });
    fireEvent.click(screen.getByText(/Sign In/i));

    await waitFor(() => {
      expect(mockedLoginUser).toHaveBeenCalledWith("testuser", "
         testpass");
      expect(mockNavigate).toHaveBeenCalledWith("/dashboard");
```

```
    });
  });
});
```

9.2.3 Coverage Achieved

Component	Coverage
Login	Full test suite (7 tests)
Register	Full test suite (9 tests)
Navbar	Full test suite (10 tests)
Dashboard	Minimal smoke test

9.2.4 Login.test.tsx – Test Breakdown

Test	What It Tests
renders login form with all elements	UI presence
updates form fields when user types	Input binding
calls loginUser with correct credentials	API call
navigates to dashboard on success	Route change
shows alert on login failure	Error feedback
shows loading state during login	Async UI feedback

9.2.5 Register.test.tsx – Test Breakdown

Test	What It Tests
renders registration form	UI presence
updates form fields	Input binding
validates password match	Client validation
calls registerUser with correct data	API call
shows success alert	Success flow
shows alert on failure	Error handling
shows loading state	Async UI feedback
prevents submission on mismatch	Validation guard
handles Enter key submission	Keyboard events

9.2.6 Navbar.test.tsx – Test Breakdown

Test	What It Tests
renders brand name and logo	Brand identity
renders navigation links	Core navigation
renders theme switch	Theme control
navigates when brand clicked	Home navigation
highlights active route	Active state styling
shows popover on avatar click	Profile menu
displays user information	User data binding
logs out when clicked	Auth flow
hides links on small screens	Responsive design
has accessible elements	Accessibility
applies gradient text	Visual styling

9.2.7 Dashboard.test.tsx – Test Breakdown

Test	What It Tests
should work without complex mocking	Test infrastructure validation

9.2.8 Mocking Strategy Summary

Mock Target	Purpose	Implementation
@/api/auth	Prevent real API calls	`vi.fn()`
react-router-dom	Control navigation	`mockNavigate`, `mockLocation`
global.alert	Capture notifications	`vi.fn()` spy
@heroui/react	Simplify components	Basic div/input elements
localStorage	Mock session data	`vi.spyOn`
ThemeSwitch	Isolate testing	Simple div placeholder

Chapter 10

Task 8: Export Reports & Library Search

10.1 Problem Statement

Provide users with equipment reporting capabilities and efficient component library search.

10.2 Tasks Done

This task was completed across Pull Requests #10, #34, and #67.

10.2.1 Equipment Reports (PR #34, #67)

Generate CSV, Excel, or Print/PDF reports of all components on canvas. Each row includes Tag No, Type, and user-editable Description.

Listing 10.1: Excel Export Implementation

```
const exportToExcel = (items) => {
  const wb = XLSX.utils.book_new();
  const ws = XLSX.utils.aoa_to_sheet([
    ["Sl No", "Tag No", "Type", "Description"],
    ...items.map((i) => [i.slNo, i.tagNo, i.type, i.description])
      ,
  ]);
```

```
  ws["!cols"] = [{ wch: 8 }, { wch: 15 }, { wch: 25 }, { wch: 40
     }];
  XLSX.utils.book_append_sheet(wb, ws, "Equipment Report");
  XLSX.writeFile(wb, `equipment-report-${date}.xlsx`);
};
```

10.2.2 Component Library Search (PR #10, #34)

Real-time filtering with category preservation, visual count badges, and drag-and-drop from cards to canvas.

Listing 10.2: Category-Preserving Filter

```
const filteredComponents = Object.keys(components).reduce(
  (result, category) => {
    const items = components[category];
    const matched = Object.keys(items).filter((key) =>
      items[key].name.toLowerCase().includes(searchQuery.
         toLowerCase())
    );
    if (matched.length > 0) {
      result[category] = matched.map((key) => items[key]);
    }
    return result;
  },
  {}
);
```

Chapter 11

Conclusions

11.1 Tasks Accomplished

Throughout the FOSSEE fellowship, I contributed to the Chemical PFD Editor, a hybrid application for chemical engineering diagramming. Key accomplishments include:

- **Project Foundation:** Established React+TypeScript architecture with Vite, TailwindCSS, and routing.
- **UI/UX:** Implemented dark mode and dashboard search/filter functionality.
- **Canvas Engine:** Built diagram canvas with draggable components, transformer resizing, grip system, grid rendering, and snap-to-grid.
- **Keyboard Shortcuts:** Created centralized shortcut registry with help popover.
- **State Management:** Migrated to Zustand store with per-editor history, batch operations, and memory management.
- **Export/Import:** Developed versioned file format, export presets, and data-driven image rendering.
- **Backend Integration:** Built API client with auth interceptors and auto-save mechanism.
- **Testing:** Established test suite with full coverage for Login, Register, and Navbar components.

- **Reporting & Search:** Added equipment report generation and category-preserving library search.
- **Bug Fixes:** Resolved issues across canvas readability, exports, history sync, and memory leaks.

11.2 Skills Developed

11.2.1 Technical Skills

During this fellowship, I deepened my proficiency in React and TypeScript for building type-safe applications. I gained expertise in Konva.js for complex canvas manipulations and mastered Zustand for scalable state management with undo/redo capabilities. I learned to design robust API clients with Axios, implement authentication flows, and create debounced auto-save mechanisms. Through comprehensive testing with Vitest and React Testing Library, I developed strong skills in writing unit tests and mocking complex dependencies. Additionally, I improved my debugging abilities, identifying and fixing performance bottlenecks and memory leaks in production-like environments.

11.2.2 Professional Skills

Working within a distributed team enhanced my collaboration skills using Git and GitHub for pull requests, code reviews, and issue tracking. I participated in regular meetings, presented technical solutions, and documented features for both developers and end-users. I learned to break down complex features into manageable tasks, estimate effort accurately, and deliver on schedule. My problem-solving approach matured through systematic debugging—from issue reproduction to implementation and verification. I also gained exposure to agile methodologies, including iterative development and continuous integration practices.

11.3 Future Scope

The Chemical PFD Editor has strong potential for future enhancements:

- **Simulation Integration:** Connect with chemical process simulators (e.g., DWSIM, Aspen) for real-time data validation and equipment sizing.
- **Collaboration Features:** Implement real-time multi-user editing with WebSocket support and comment/annotation capabilities.
- **Mobile Support:** Develop responsive mobile viewer and companion app for on-site diagram access.
- **Advanced Export:** Add DXF/DWG export for CAD software integration and enhanced PDF reporting with custom templates.
- **Component Library:** Expand built-in symbol library and allow user-uploaded custom components.
- **Cloud Deployment:** Migrate to scalable cloud infrastructure with improved performance monitoring.
- **Machine Learning:** Explore auto-diagram completion and intelligent component placement suggestions.

This fellowship provided invaluable hands-on experience in full-stack development while fostering professional growth within the open-source community.

Bibliography

[1] FOSSEE Project, *FOSSEE Website*, `https://fossee.in/`, Accessed: 2025-12-05.