Winter Internship Report

On

Frontend Architecture and UI Development for the Chemical PFD Desktop Application

Submitted by

Nitish Karthick V

Under the guidance of

Prof. Prabhu Ramachandran
Department of Aerospace Engineering
IIT Bombay

February 19, 2026

Acknowledgment

I would like to express my sincere gratitude to the FOSSEE team at IIT Bombay for the incredible opportunity to be a part of the Winter Internship program. This experience has been both intellectually rewarding and personally inspiring. Working on the Chemical PFD desktop application allowed me to dive deep into frontend architecture, complex UI design, and professional software development using open-source tools.

During this internship, I had the opportunity to design and implement core features for the Chemical PFD platform including the Infinite Canvas and the PDF reporting module making a tangible contribution to its growing open-source ecosystem. Through this hands-on work, I was able to explore the intricacies of desktop application development in a practical environment, gaining insight into real-world software engineering workflows.

I am grateful to Prof. Prabhu Ramachandran for his vision in leading the FOSSEE initiative, and for making open-source tools accessible to students across the country. His efforts in promoting self-reliance in engineering education have created learning platforms that continue to impact thousands of aspiring engineers like me.

A special note of thanks to my mentor, Mr. Chirag Koyande, for his continuous guidance and support during the internship. Their practical advice, clear expectations, and willingness to help whenever I faced roadblocks played a significant role in shaping my project outcomes.

This internship has been a defining chapter in my academic journey, and I leave it with not only new skills and knowledge but also with clarity and excitement for the path that lies ahead in the software engineering and frontend development domain.

Contents

Chapter 1

Introduction

The FOSSEE (Free/Libre and Open-Source Software for Education) project at IIT Bombay, an initiative of the Ministry of Education, Government of India, is committed to promoting the widespread adoption of open-source software in academic and research institutions. By reducing dependence on expensive proprietary tools, FOSSEE empowers individuals and institutions to explore free/libre alternatives that are equally capable and accessible.

1.1 FOSSEE Winter Internship.

The FOSSEE (Free/Libre and Open Source Software for Education) project, based at IIT Bombay, promotes the use of open-source software in educational institutions. This internship was conducted under the FOSSEE Summer Fellowship, focusing on contributing to the "Chemical PFD" project,a tool designed to reduce reliance on proprietary process simulation software.

1.2 Chemical PFD Desktop Application

The Chemical PFD Desktop Application is a cross-platform tool that enables chemical engineers to design Process Flow Diagrams (PFDs). It allows users to drag and drop standard chemical unit operations, connect them with streams, and define process parameters. The application bridges the gap between simple drawing tools and complex simulation packages..

1.3 PyQt / PySide UI Framework

The frontend is built using PyQt (Python bindings for the Qt application framework). Qt was selected for its:

- **Graphics View Framework:** Essential for handling the complex 2D interactions required by the canvas.
- **Signal-Slot Mechanism:** Facilitating robust event handling and communication between UI components.
- **Native Look and Feel:** Ensuring the application feels responsive and professional on Windows, Linux, and macOS.

1.4 ReportLab PDF Engine

ReportLab is utilized for the "Generate Report" feature. Unlike simple screen captures, ReportLab allows for the programmatic generation of vector-based PDFs. This ensures that the output documents—containing equipment lists, stream tables, and the PFD itself—are high-resolution and suitable for printing and academic submission.

1.5 Key Features of the Desktop Frontend

- Infinite Workspace: A pan-and-zoom capable canvas for large-scale diagrams.
- **Smart Routing:** Automatic orthogonal wire routing for clean connections.
- **Theme Support:** Toggleable Dark and Light modes for user comfort.
- **Export Capabilities:** High-fidelity PDF generation and image export.

Chapter 2

Frontend Architecture and System Design

2.1 Problem Statement

Chemical engineering students and professionals often lack free, accessible tools to create standardized PFDs. Existing open-source tools often suffer from poor user interfaces (UI) or lack specific domain context (e.g., distinguishing between a "stream" and a distinct "line"). The goal was to build a modern, responsive frontend that provides an intuitive user experience without sacrificing technical rigor.

2.2 Approach and Methodology

The development approach followed an iterative Agile methodology:

- **Requirement Analysis:** Studying the existing "Chemical PFD" backend constraints and user needs.
- **Prototype Design:** Creating isolated prototypes for the infinite canvas and connection logic.
- **Integration:** Merging these modules into the monolithic application structure.
- **Refinement:** Polishing visual artifacts (themes, icons) and optimizing rendering performance.

Chapter 3

Feature Implementation and Development

3.1 Infinite Canvas and Zoom System

The workspace of a Process Flow Diagram can quickly become cluttered as the complexity of the simulated chemical plant increases. To accommodate diagrams of any scale without artificially restricting the user, a custom Infinite Canvas was developed. This system replaces a static, fixed-size drawing board with a dynamically expanding graphical environment. It is coupled with a robust mathematical zooming mechanism, allowing engineers to navigate smoothly between high-level plant overviews and highly detailed component views.

3.1.1 Logical vs. Visual Coordinate Separation

To support a theoretically infinite workspace, the system separates "Scene Coordinates" (logical position) from "View Coordinates" (screen pixels). Objects exist at fixed logical coordinates, while the View transforms these into pixels based on the current scroll position and scale factor.

3.1.2 Dynamic Workspace Expansion

The canvas does not have a fixed size. As the user drags a component near the edge of the current viewport, the sceneRect creates a "virtual expansion," seamlessly growing the scrollable area. This prevents the user from confusingly running out of drawing space.

3.1.3 Zoom Scaling Mechanisms

Zooming is implemented via an affine transformation matrix.

- Wheel Event: Rolling the mouse wheel scales the view matrix by a factor of 1.1 or 0.9.
- Anchor Point: The scaling transformation is anchored to the mouse cursor's position, ensuring the view zooms "into" the detail the user is pointing at, rather than the center of the screen.

3.1.4 Output Screenshots

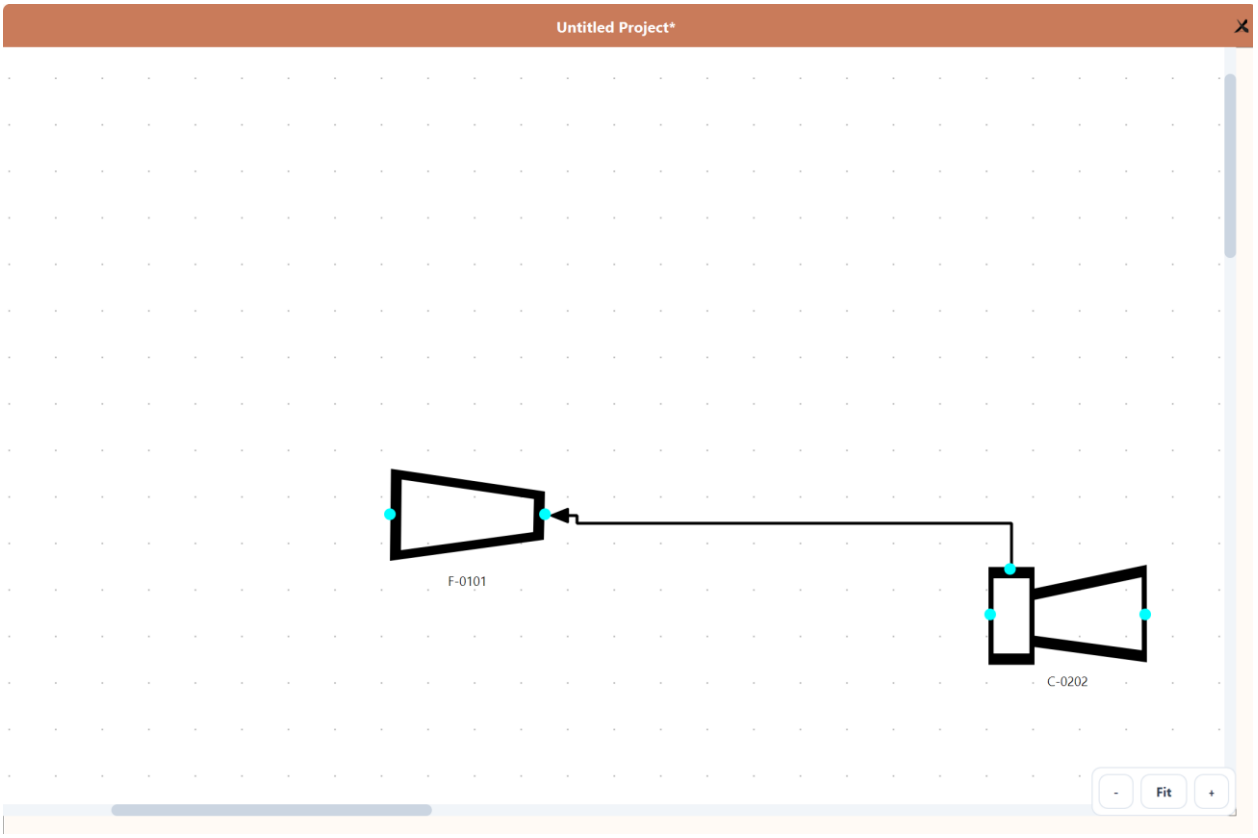


Figure 3.1: Logical and visual coordinate rendering on the Infinite Canvas demonstrating accurate spatial separation.

3.2 Connection Engine

At the heart of any Process Flow Diagram is the intricate network of streams connecting various chemical unit operations. The Smart Connection Engine was engineered to handle this routing logic programmatically. Instead of relying on manual, freehand line drawing, this engine automates the pathfinding process. It ensures that all connections adhere to strict engineering schematic standards while maintaining precise anchor points on every piece of equipment, even as they are moved across the canvas.

3.2.1 JSON-based Component Grip Mapping

Every chemical unit (e.g., Distillation Column) has specific inlet and outlet ports ("grips"). These logical positions are mapped in a grips.json configuration file, allowing the engine to know exactly where a stream should attach to a component, regardless of its size.

3.2.2 Orthogonal (90-degree) Routing Algorithm

To ensure professional-grade diagrams, connections must not be diagonal. The routing algorithm calculates a path using a Manhattan distance heuristic:

1. Identify Start (Source Port) and End (Dest Port) points.
2. Calculate intermediate waypoints to form an "L", "S", or "Z" shape.
3. Adjust segments to align with the grid.

3.2.3 Visual Polish: Arrows and Bridge Curves (Jumps)

- **Directional Arrows:** Added to the end segment of every connection to indicate flow direction.

- **Bridge Curves:** When a vertical line crosses a horizontal line without connecting, a small semi-circle "jump" is rendered at the intersection point. This is standard in electrical and process schematics to distinguish crossing lines from joining lines.

3.2.4 Routing Output Screenshots

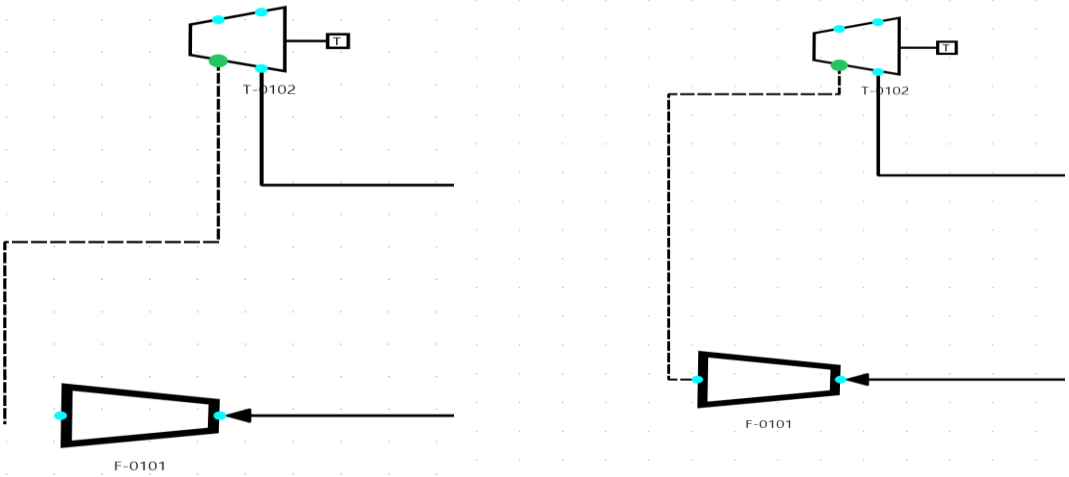


Figure 3.2 & Figure 3.3: Automated Routing Progression. Figure 3.2 (left) illustrates the initiation of a connection stream from the source equipment, while Figure 3.3 (right) demonstrates the engine successfully locking the 90-degree orthogonal path to the destination grip.

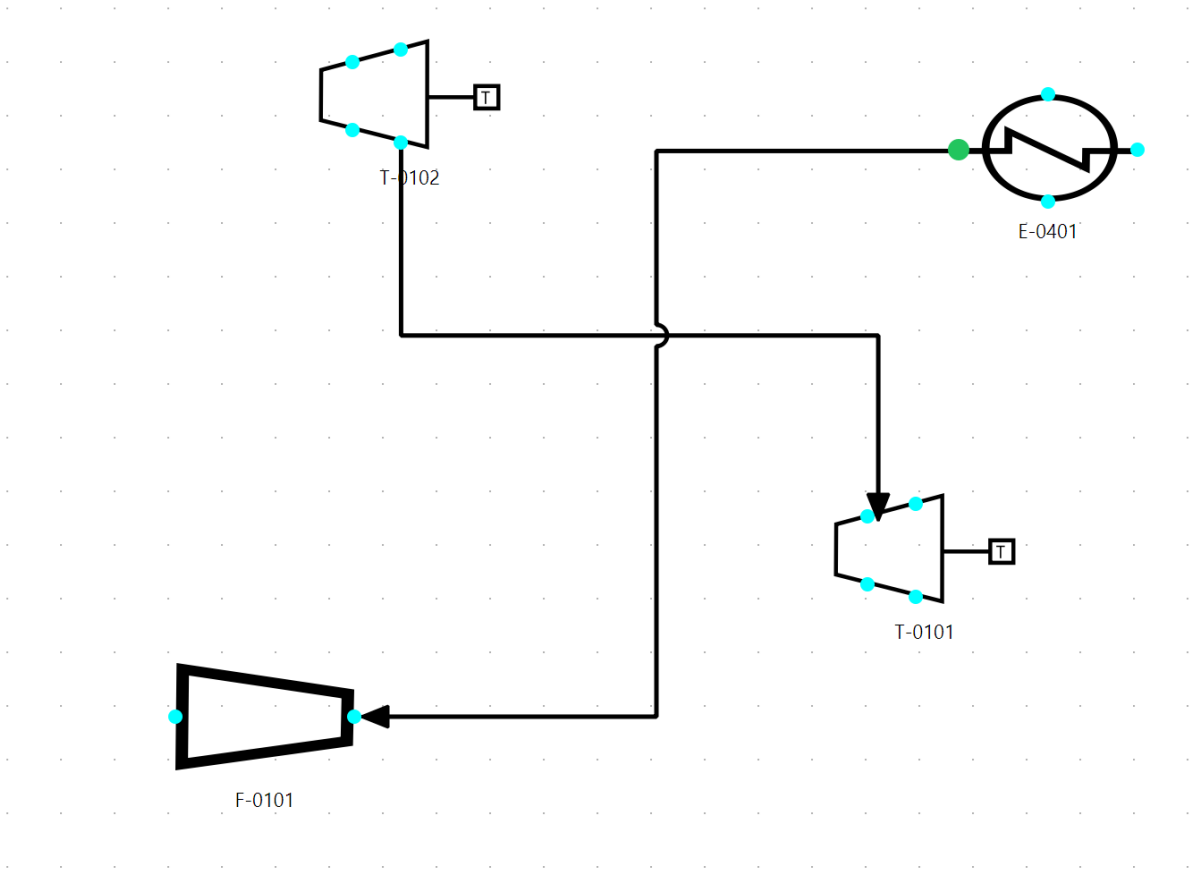


Figure 3.4: Visual rendering of a bridge curve ("jump") to clearly distinguish overlapping, non-connected streams.

3.3 Professional PDF Reporting Engine

Designing the visual layout of a PFD is only the first phase of the workflow; exporting that data for analysis, presentation, and inventory management is equally critical. The Professional PDF Reporting Engine was integrated to completely replace legacy raw-text outputs with formatted, industry-standard documents. This module bridges the gap between the visual graphics on the canvas and the quantitative data required for official engineering documentation.

3.3.1 Data Extraction and LaTeX Tag Cleaning

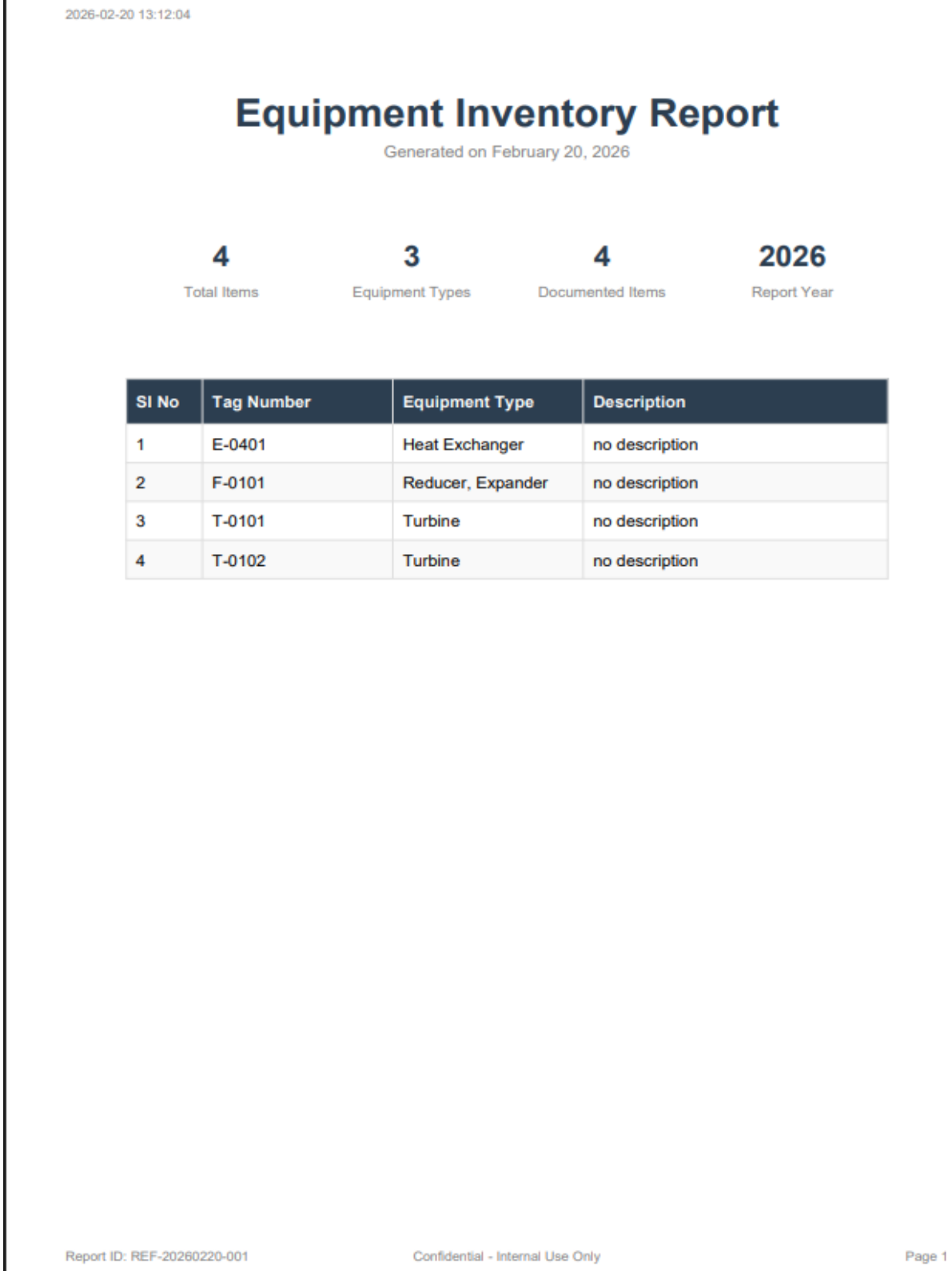
User input often contains LaTeX-style formatting for chemical names (e.g., H_2SO_4). Reviewing input strings, the engine parses these tags using regex and replaces them with ReportLab's superscript/subscript implementations to ensure chemical formulas render correctly in the final PDF.

3.3.2 ReportLab Layout (Summary Cards & Tables)

The PDF report is structured dynamically:

- Header: Project details and timestamp.
- Summary Cards: A grid of 4 key statistics (Total Units, Streams, etc.).
- Data Table: A detailed list of all equipment and their descriptions. The layout uses ReportLab's Platypus flowables engine to automatically handle page breaks and table styling.

3.3.3 Generated PDF Output Screenshots



2026-02-20 13:12:04

Equipment Inventory Report

Generated on February 20, 2026

4	3	4	2026
Total Items	Equipment Types	Documented Items	Report Year

Sl No	Tag Number	Equipment Type	Description
1	E-0401	Heat Exchanger	no description
2	F-0101	Reducer, Expander	no description
3	T-0101	Turbine	no description
4	T-0102	Turbine	no description

Report ID: REF-20260220-001 Confidential - Internal Use Only Page 1

Figure 3.5: Output of the Professional PDF Reporting Engine detailing dynamic summary statistics and the structured data table.

3.4 Adaptive Theme Manager and UI

A modern engineering tool requires a user interface that is not only highly functional but also visually accessible during long, complex design sessions. The Adaptive Theme Manager and UI enhancements were implemented to overhaul the application's overall aesthetics and structural responsiveness. This structural upgrade ensures the interface scales beautifully across different monitors while introducing dedicated viewing modes to reduce user eye strain.

3.4.1 Dark Mode Implementation Strategy

A ThemeManager class was implemented to centralize color definitions. Instead of hardcoding colors, widgets reference variables (e.g., PRIMARY_BG). Switching modes triggers a signal that updates these variables and re-applies the application-wide stylesheet.

3.4.2 Responsive Sidebar (Flow Layout)

The component library sidebar uses a responsive flow layout. As the sidebar width changes, component icons automatically reflow to fill the available space, maintaining an organized grid without horizontal scrollbars.

3.4.3 High Fidelity SVG Component Rendering

All chemical unit icons are rendered from SVG (Scalable Vector Graphics) source files. This ensures that regardless of the theme (Dark/Light) or zoom level, the icons remain crisp and pixel-perfect.

3.4.4 Theme Output Screenshots

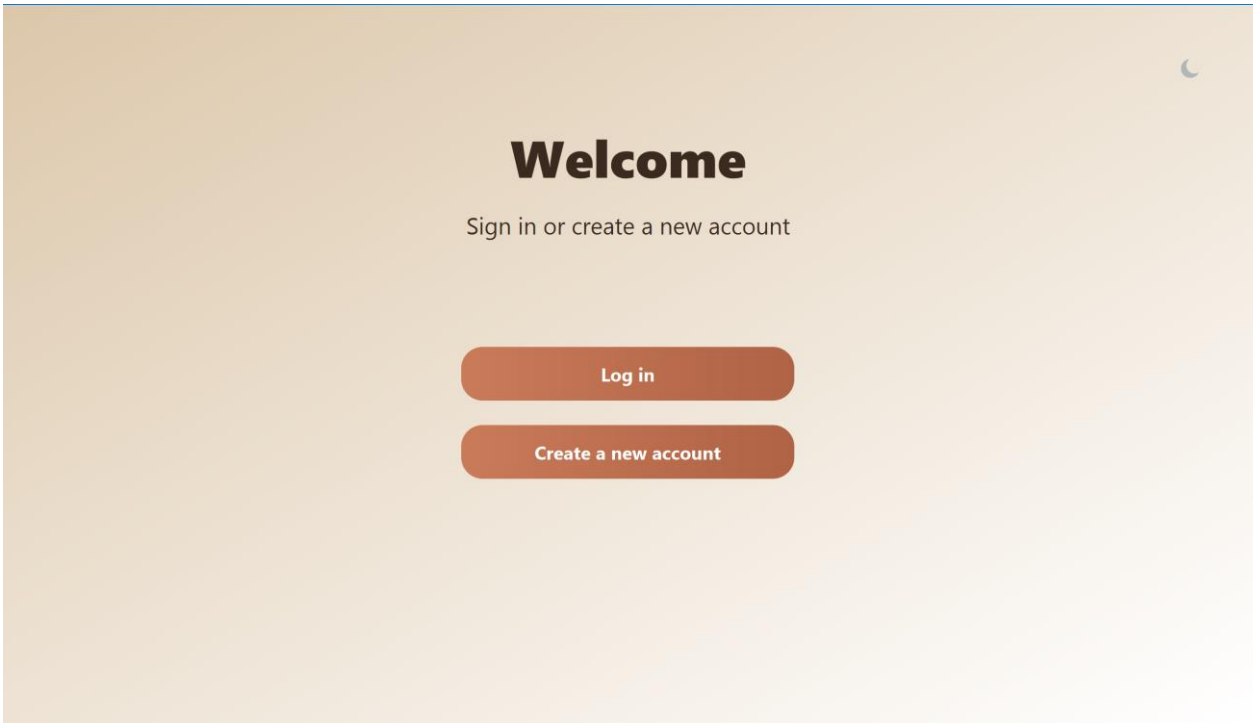


Figure 3.6: The application welcome interface rendered in the standard Light Theme.

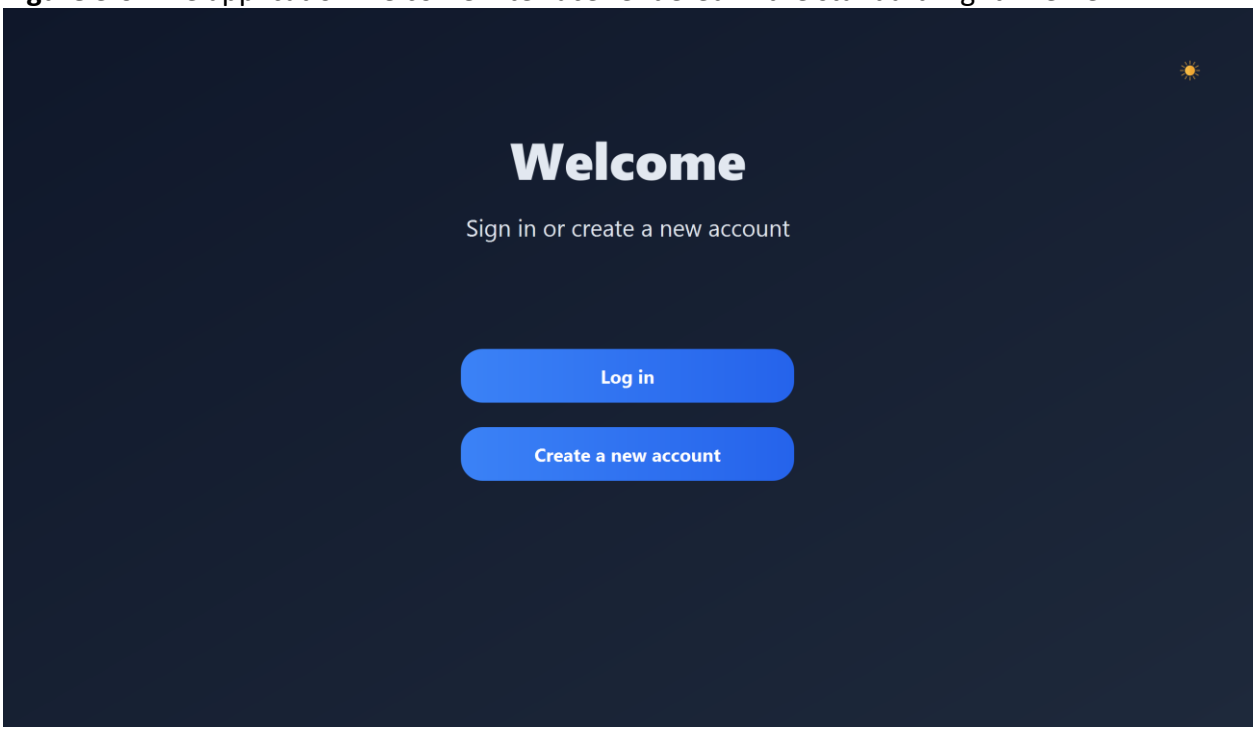


Figure 3.7: The application welcome interface rendered using the Adaptive Dark Theme manager.

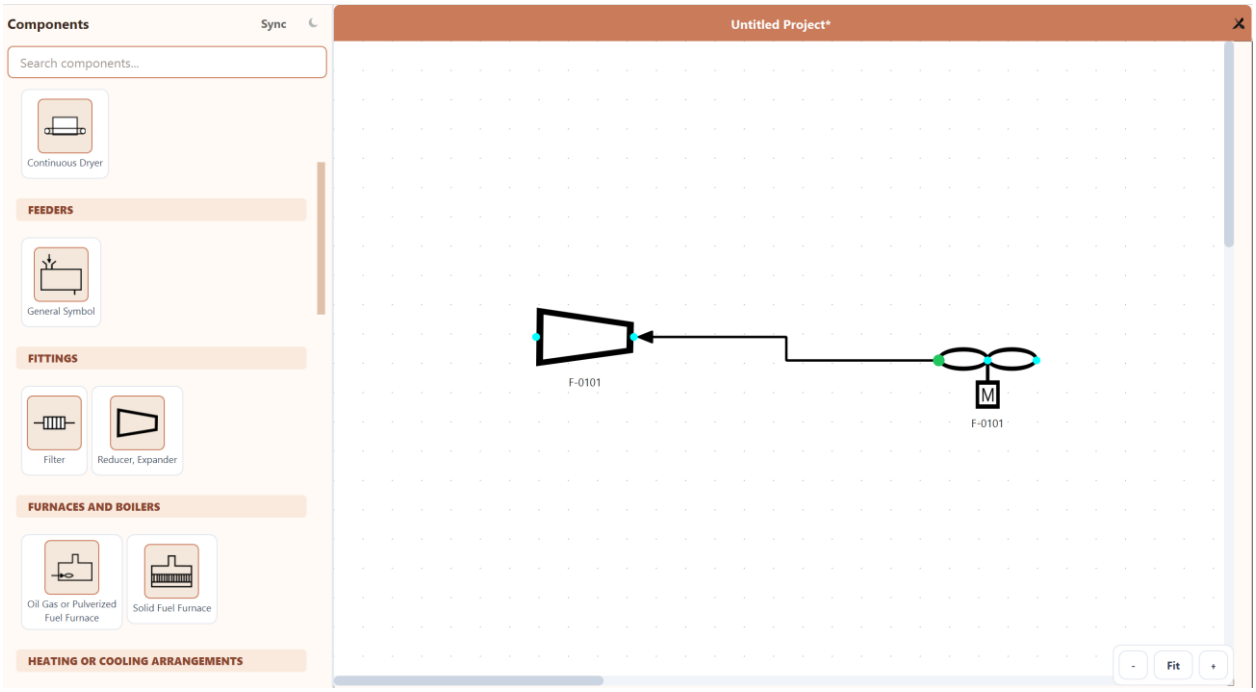


Figure 3.8: Main workspace and responsive component sidebar operating in the standard Light Theme.

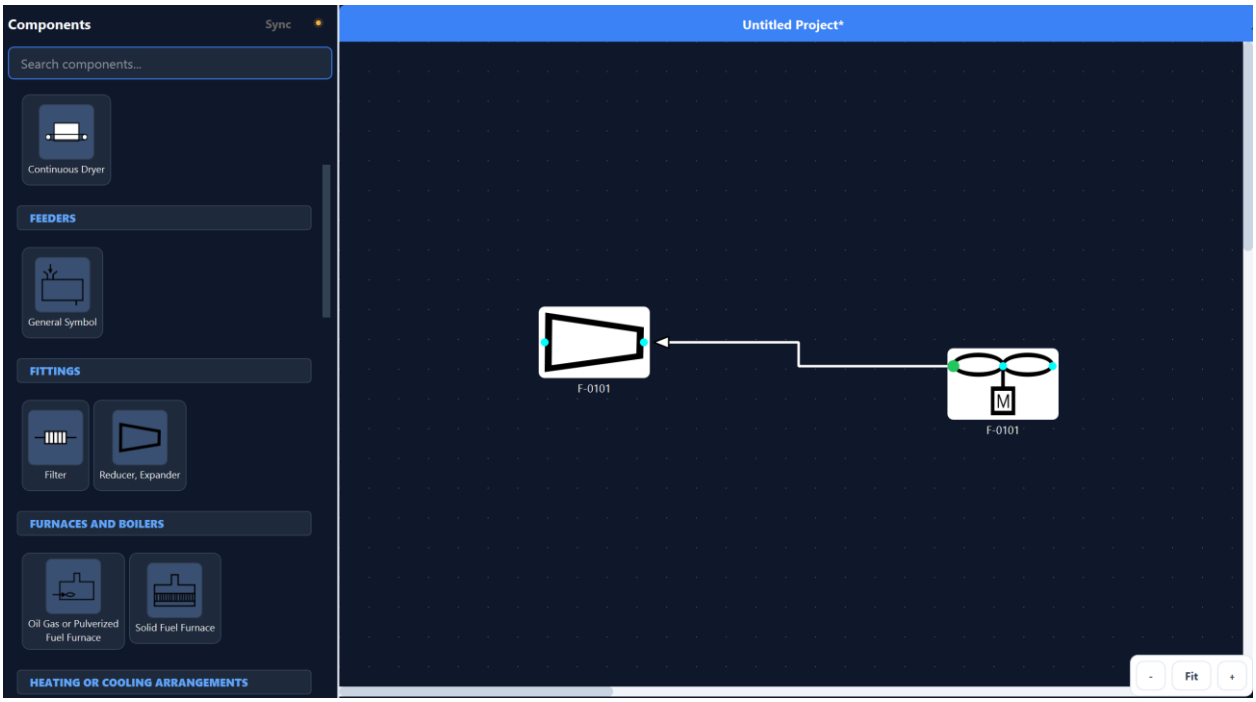


Figure 3.9: Main workspace demonstrating high-contrast Dark Mode SVG rendering, adaptive background plates, and inverted connection lines.

3.5 Bug Fixes and Codebase Security

3.5.1 Resolving Connection Rendering Crashes

Fixed a critical AttributeError where the connection painter attempted to access a destroyed scene object during rapid deletions. Implemented logic to check for object validity before repaint events.

3.5.2 Wiring Logic Restoration

Addressed a regression where connection lines would "snap" to the wrong port after a component was moved. The fix involved updating the internal coordinate mapping of the Connection object upon the ItemPositionHasChanged event of the attached component.

3.5.3 Version Control Audits and Gitignore Setup

Cleaned the repository by removing compiled Python files (__pycache__, .pyc) and temporary editor files. Configured a comprehensive .gitignore to prevent future pollution of the codebase, ensuring a clean history for the FOSSEE maintainers.

Chapter 4

Conclusion and Future Scope

During this internship, I explored and developed the frontend architecture of the Chemical PFD desktop application, closely adhering to modern UI standards. The project involved building and verifying functional blocks such as the Infinite Canvas, a Smart Connection Engine with orthogonal routing, an adaptive Theme Manager, and a PDF Reporting Engine. Each feature was thoroughly tested, with validation carried out through rigorous debugging to ensure absolute reliability and correctness.

These verified software modules are now integrated into the official Chemical PFD codebase, where they form the interactive graphical foundation of the software. They serve as robust, reusable UI components for academic projects, process engineering training, and open-source development. This contribution helps broaden the reach of the application as a practical tool for learning and prototyping in chemical engineering domains.

Moving ahead, there is great potential to extend this work by introducing more complex frontend capabilities such as real-time collaborative editing, canvas animations, machine learning-assisted auto-routing, or CAD format exports. Such enhancements can transform the application into a go-to platform for system-level plant design, encouraging deeper experimentation and innovation in chemical engineering education and research.

Chapter 5

Bibliography

General References

- FOSSEE Initiative, IIT Bombay: https://fossee.in/
- Python 3.x Official Documentation: https://docs.python.org/3/
- Qt for Python (PySide/PyQt) Official Documentation: https://doc.qt.io/qtforpython/
- ReportLab Open Source PDF Library User Guide: https://docs.reportlab.com/
- W3C Scalable Vector Graphics (SVG) Specification: https://www.w3.org/Graphics/SVG/
- JSON (JavaScript Object Notation) Data Interchange Standard: https://www.json.org/

Technical Documentation & Libraries

1. **PyQt6/PySide6 Graphics View Framework:** https://doc.qt.io/qt-6/graphicsview.html
2. **ReportLab PLATYPUS (Page Layout and Typography Using Scripts):** https://docs.reportlab.com/reportlab/userguide/ch5_platypus/
3. **Manhattan Distance & Orthogonal Routing Algorithms:** https://en.wikipedia.org/wiki/Taxicab_geometry