FOSSEE Summer Internship Report

On

Development of Chemical PFD Builder

Submitted by

SHUBHAM KAYANDE

3rd Year B.Tech Student, CSE MIT Engineering College Chh. Sambhajinagar

Under the Guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

Indian Institute of Technology Bombay

Mentor:

Chirag Koyande

July 1, 2026

Acknowledgments

I would like to express my sincere gratitude to the **FOSSEE Project, Indian Institute of Technology (IIT) Bombay**, for providing me with the opportunity to be a part of the **Semester long Internship Program**. This internship has been an invaluable learning experience, allowing me to contribute to the development of the **Chemical PFD Builder** while gaining practical exposure to open-source software development.

I am deeply thankful to **Prof. Prabhu Ramachandran** for his vision and leadership in promoting open-source technologies through the FOSSEE initiative.

I would like to express my heartfelt gratitude to my mentor, **Mr. Chirag Koyande**, for his continuous guidance, technical support, and valuable feedback throughout the internship. His mentorship helped me improve my technical skills and approach software development with confidence.

I also thank the entire Chemical PFD Builder team for their collaboration, code reviews, and encouragement. Working with the team enhanced my understanding of professional software development practices, including Git, testing, documentation, and collaborative development.

Finally, I would like to thank my family, teachers, and friends for their constant support and motivation throughout this internship. The knowledge and experience I gained during this journey will remain valuable in my future academic and professional career.

Table of Contents

Chapter 1

Introduction

1.1 FOSSEE Project

The **Free/Libre and Open Source Software for Education (FOSSEE)** project is an initiative of the **Indian Institute of Technology (IIT) Bombay** under the **National Mission on Education through Information and Communication Technology (NMEICT)**, Ministry of Education, Government of India. The primary objective of the project is to promote the use of Free and Open Source Software (FOSS) in academic institutions, research organizations, and industries across India.

FOSSEE encourages students, educators, and researchers to adopt open-source software by developing educational resources, organizing workshops, and creating engineering tools that serve as alternatives to commercial software. Through internships, fellowships, and collaborative projects, students gain practical software development experience while contributing to applications used by the engineering community.

The project has played an important role in building a strong open-source ecosystem by encouraging innovation, collaboration, and knowledge sharing among contributors from different technical backgrounds.

1.1.1 Projects and Activities

The FOSSEE project supports various initiatives aimed at promoting open-source software in education and research. Some of its major activities include:

- **Textbook Companion Project:** Conversion of solved textbook examples into open-source software implementations.
- **Lab Migration Project:** Migration of laboratory experiments from proprietary software to open-source alternatives.
- **Niche Software Development:** Development and maintenance of specialized engineering software for different technical domains.

- **Workshops and Training Programs:** Conducting faculty development programs, workshops, and technical training sessions.
- **Internship and Fellowship Programs:** Providing students with opportunities to contribute to real-world open-source software projects.
- **Community Support:** Maintaining forums, documentation, and collaborative platforms for users and developers.

These initiatives help bridge the gap between theoretical education and practical software development while promoting the adoption of open-source technologies.

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

The primary purpose of the Chemical PFD Builder is to simplify the creation of Process Flow Diagrams through an interactive graphical interface. Instead of manually drawing engineering diagrams using traditional software, users can build process flows by dragging components onto the canvas and connecting them using intelligent routing mechanisms.

The project has been designed with the following objectives:

- Provide a free and open-source platform for creating Process Flow Diagrams.
- Support both desktop and web-based diagram editing.
- Improve accessibility for students, educators, and engineers.
- Maintain compatibility between desktop and web platforms.
- Simplify project management through cloud-based storage and synchronization.
- Encourage collaborative development using open-source technologies.

1.2.2 System Architecture

The Chemical PFD Builder follows a hybrid architecture consisting of a common backend that serves both the desktop and web applications.

The architecture is divided into three major layers:

Backend

The backend is developed using **Django REST Framework** and provides centralized services including:

- User authentication
- Project management
- Component management
- Diagram storage
- RESTful APIs
- Export and report generation

The backend acts as the central communication layer between the database and the client applications.

Web Frontend

The web application is built using **React.js**, **TypeScript** It provides users with a browser-based editor for creating and editing Process Flow Diagrams without requiring software installation.

Major features include:

- Interactive drawing canvas
- Component library
- Cloud synchronization
- Project management
- Export functionality

Desktop Frontend

The desktop application is developed using **Python** and **PyQt5**. It provides a native environment with enhanced editing capabilities and improved performance for large engineering diagrams.

Its major features include:

- Drag-and-drop component placement
- Intelligent connection routing
- Component editing
- Project management
- Diagram validation
- Report generation
- Offline support

Shared Database

Both applications communicate with the same backend services, allowing users to store, retrieve, and synchronize projects seamlessly across platforms.

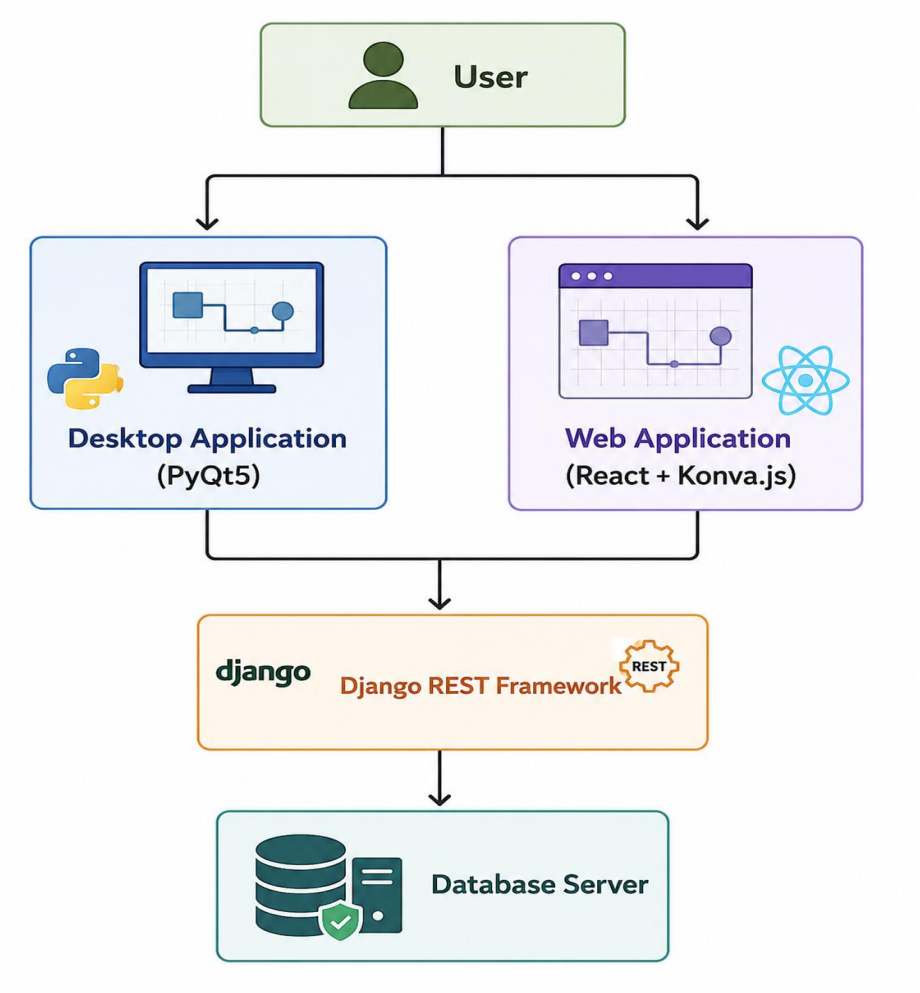


Figure 1.1: Hybrid System Architecture

1.2.3 Key Features

The Chemical PFD Builder offers several features that improve the efficiency of Process Flow Diagram creation.

- Drag-and-drop component placement.
- Large library of standard chemical engineering symbols.
- Intelligent orthogonal connection routing.
- Diagram validation and error detection.
- Project save, load, edit, and management.
- Import and export of engineering diagrams.

- Cross-platform compatibility between desktop and web.
- Support for custom engineering symbols.
- Cloud-based project synchronization.
- Report generation for engineering documentation.

These features provide users with a complete environment for designing and managing Process Flow Diagrams efficiently.

1.2.4 Technologies Used

The project combines multiple modern technologies to build a scalable and maintainable software system.

Layer	Technology
Programming Language	Python, JavaScript, TypeScript
Desktop Application	PyQt5
Web Application	React.js, Konva.js
Backend	Django REST Framework
Database	SQLite / PostgreSQL
Version Control	Git, GitHub
API Communication	REST API
Graphics	SVG
Testing	PyTest
Documentation	Markdown

1.2.5 Benefits of Chemical PFD Builder

The Chemical PFD Builder offers several advantages compared to conventional diagram creation tools.

- Free and open-source software.
- User-friendly graphical interface.
- Platform-independent project access.
- Faster diagram creation through intelligent tools.
- Improved collaboration using centralized project storage.
- Easy maintenance and extensibility through modular architecture.
- Suitable for educational, research, and industrial applications.

1.2.6 Conclusion

The Chemical PFD Builder is a comprehensive open-source application that simplifies the creation and management of Process Flow Diagrams while promoting the use of open-source software in engineering education. By combining modern web technologies, desktop application development, and a centralized backend, the project provides a flexible and efficient platform for students, educators, and professionals.

This internship provided an opportunity to contribute to the development of the desktop application by implementing new features, improving existing functionalities, enhancing software quality through testing, and resolving critical issues. The experience offered valuable exposure to collaborative software development, version control, software testing, and real-world engineering application development.
For more information,visit the GitHub repository: Link

Chapter 2

Screening Task

2.1 Problem Statement

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

2.1.1 Project Overview

The screening project involved developing a Chemical Equipment Visualizer capable of reading equipment information from CSV files and presenting the processed data through both web and desktop interfaces.

The uploaded CSV contained equipment information such as:

- Equipment Name
- Equipment Type
- Flow Rate
- Pressure
- Temperature

The backend processed the uploaded data, generated statistical summaries, and exposed REST APIs that were consumed by both frontend applications.

The system was designed so that users could upload datasets once and visualize the same information from either the desktop application or the web application.

2.1.2 Functional Requirements

The application was required to provide the following functionalities:

- Upload CSV datasets.
- Parse and validate uploaded files.
- Store processed data in the database.
- Generate statistical summaries.
- Display equipment information in tabular format.
- Visualize data using graphs and charts.
- Maintain upload history.
- Generate downloadable PDF reports.
- Support user authentication.

- Work consistently on both Desktop and Web platforms.

2.2 System Design

The application followed a three-layer architecture consisting of:

Backend Layer

The backend was developed using **Django REST Framework**.

Its responsibilities included:

- CSV parsing
- Data validation
- Database operations
- Authentication
- API development
- Report generation

Web Application

The web frontend was built using **React.js**.

Major features included:

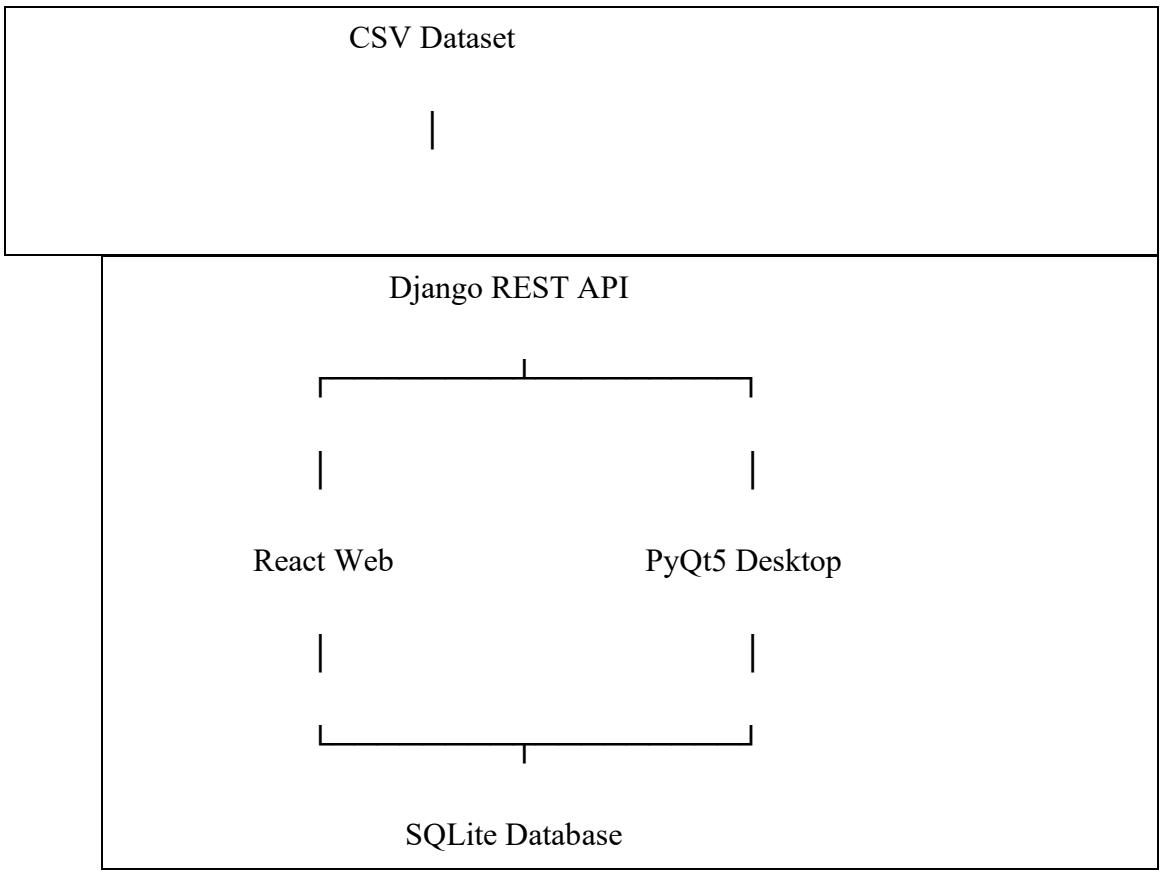
- CSV upload interface
- Interactive dashboard
- Charts using Chart.js
- Equipment table
- Summary statistics

Desktop Application

The desktop application was developed using **PyQt5**.

It provided:

- Native desktop interface
- CSV upload
- Data visualization
- Charts using Matplotlib
- Equipment analysis
- Report generation



2.3 Implementation

The implementation was divided into three independent modules.

2.3.1 Backend Development

The backend handled all server-side operations.

Major functionalities included:

- CSV parsing using Pandas.
- Equipment data validation.
- Database storage.
- Summary generation.
- REST API development.
- Automatic history management.
- PDF report generation.

The backend served as the common interface for both desktop and web applications.

2.3.2 Web Frontend

The web interface provided a modern dashboard allowing users to interact with uploaded datasets.

Implemented features included:

- Drag-and-drop CSV upload.
- Equipment data table.
- Charts and graphs.
- Dashboard statistics.
- Responsive layout.

2.3.3 Desktop Frontend

The desktop application focused on providing a native user experience.

Implemented features included:

- File upload dialog.
- Data table.
- Equipment visualization.
- Charts using Matplotlib.
- PDF report generation.
- Dataset history.

2.4 Technologies Used

Component	Technology
Backend	Django 4.2 + Django REST Framework
Database	SQLite
Data Processing	Pandas
PDF Generation	ReportLab
Web Frontend	React 18 + Vite
Web Charts	Chart.js
Desktop Frontend	PyQt5

Desktop Charts	Matplotlib

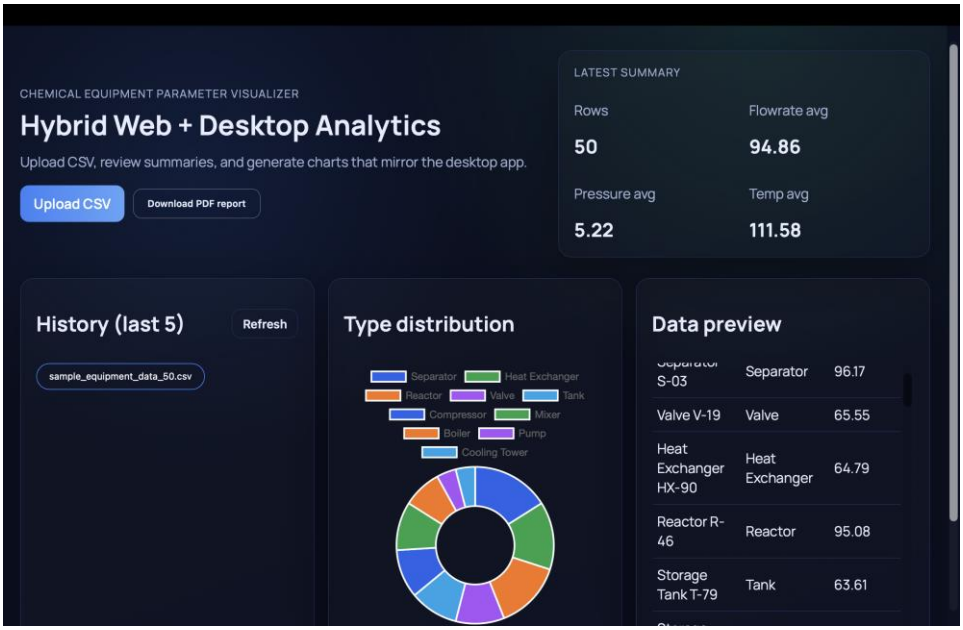
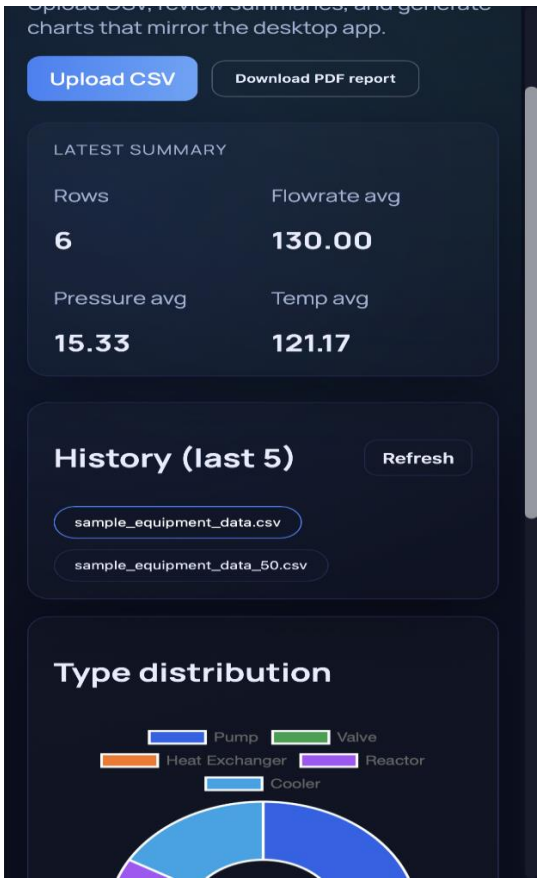


Figure 2.1: Selection Task desktop Frontend Dashboard



2.6 Outcome

The screening task was successfully completed by developing a fully functional hybrid application that satisfied all the required objectives.

The project demonstrated the integration of backend services with both web and desktop applications while ensuring consistent functionality across platforms. It also provided practical exposure to REST API development, desktop application programming, frontend development, and data visualization.

Successfully completing this task led to my selection for the **FOSSEE Chemical PFD Builder Internship**, where I contributed to the development of the desktop application by implementing new features, improving routing mechanisms, enhancing user experience, strengthening the export system, writing unit tests, and fixing software issues.

Chapter 3

Smart Auto-Routing using Breadth-First Search (BFS)

3.1 Problem Statement

In the initial version of the Chemical PFD Builder, creating connections between components required significant manual effort. Users had to carefully draw connection lines while ensuring that they reached the correct destination without affecting the readability of the Process Flow Diagram (PFD). As the number of components increased, manually creating clean and organized connections became difficult and time-consuming.

Furthermore, manually drawn connections often resulted in inconsistent layouts, unnecessary bends, and poor visualization, making large process diagrams difficult to understand.

To improve the user experience, an intelligent routing mechanism was required that could automatically determine an appropriate path between two selected components while maintaining a clean and structured layout.

The objective of this task was to design and implement a **Smart Auto-Routing System** capable of automatically generating valid paths between component connection points.

3.2 Objective

The main objectives of the Smart Auto-Routing feature were:

- Automatically generate connection paths between two components.
- Minimize manual routing effort.
- Improve diagram readability.
- Produce consistent routing behavior.
- Prepare the foundation for advanced routing improvements such as obstacle avoidance and strict orthogonal routing.

3.3 Challenges

Developing the routing algorithm involved several technical challenges:

- Determining a valid path between source and destination grips.
- Preventing unnecessary routing complexity.
- Maintaining good routing performance for large diagrams.

- Handling dynamic movement of components.
- Providing smooth integration with the existing canvas system.

These challenges required selecting a pathfinding algorithm that was efficient, reliable, and easy to integrate with the desktop application.

3.4 Selection of Breadth-First Search (BFS)

To solve the routing problem, **Breadth-First Search (BFS)** was selected as the core pathfinding algorithm.

BFS is a graph traversal algorithm that explores all neighboring nodes before moving to the next level. Since each movement on the routing grid has equal cost, BFS guarantees the shortest available path between the source and destination.

The routing canvas was represented as a logical grid where each grid cell acts as a graph node. The algorithm begins at the source connection point and explores neighboring cells until the destination is reached.

This approach provided:

- Shortest valid path
- Predictable routing
- Simple implementation
- Reliable performance

3.5 Algorithm Workflow

The Smart Auto-Routing process follows the steps below:

1. User selects the source component.
2. User selects the destination component.
3. Connection grips are identified.
4. A logical routing grid is generated.
5. BFS starts from the source grip.
6. Neighboring cells are explored level by level.
7. The destination grip is reached.
8. The shortest path is reconstructed.
9. The generated path is rendered on the canvas.

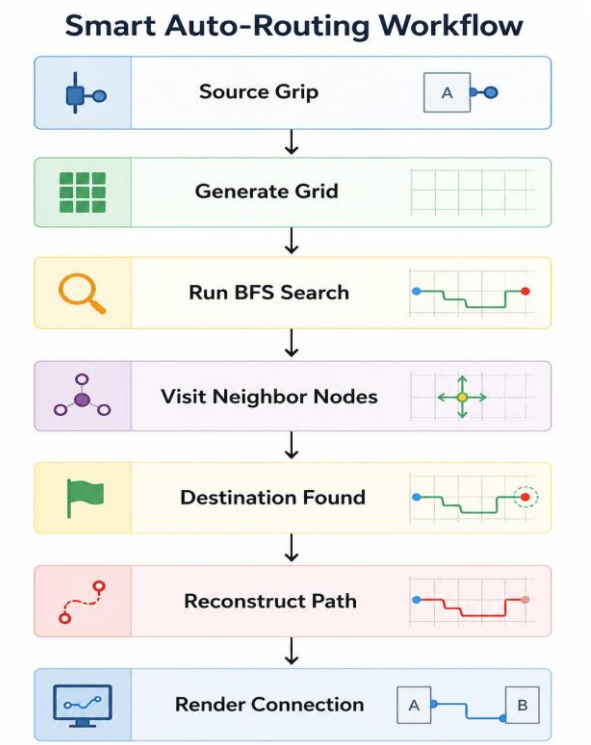


Figure 3.1 Smart Auto-Routing Workflow

3.6 Implementation

The routing system was integrated into the desktop application's connection management module.

When a new connection is created, the routing engine automatically:

- Detects the source grip.
- Detects the destination grip.
- Generates the routing grid.
- Executes the BFS algorithm.
- Stores the shortest path.
- Draws the connection on the canvas.

The implementation was designed to work seamlessly with the existing component architecture without requiring additional user interaction.

3.7 Advantages of BFS-Based Routing

The implemented solution provides several advantages:

- Automatically creates connection paths.

- Produces the shortest available route.
- Reduces manual editing.
- Improves readability of Process Flow Diagrams.
- Provides deterministic routing behavior.
- Forms the foundation for future routing enhancements.

3.8 Code Snippet

```
from auto_router import AutoRouter

from PyQt5.QtCore import QPointF

router = AutoRouter(grid_resolution=10)

router.add_component_obstacle(component.logical_rect)

for c in existing_connections:

    router.add_connection_obstacle(c.path)

grid_path = router.find_path(QPointF(x1, y1), QPointF(x2, y2))

if grid_path:

    painter_path = router.build_painter_path(grid_path)

    connection.setPainterPath(painter_path)
```

Explanation

The BFS algorithm begins from the source node and explores neighboring grid cells level by level. Once the destination is found, the stored parent information is used to reconstruct the shortest path, which is then rendered as the connection line.

3.9 Results

The Smart Auto-Routing feature significantly improved the usability of the desktop application.

The generated routes were:

- Automatic
- Consistent

- Shortest
- Easy to understand
- Suitable for engineering diagrams

The implementation also served as the base for later improvements such as obstacle avoidance, orthogonal routing, and connection optimization.

3.10 Outcome

The Smart Auto-Routing system successfully automated the process of generating connection paths between components in the Chemical PFD Builder. By using the Breadth-First Search algorithm, the application was able to compute the shortest valid path while reducing manual effort and improving the overall quality of Process Flow Diagrams.

This feature marked an important enhancement to the desktop application and laid the foundation for subsequent routing improvements implemented during the internship.

Chapter 4

Orthogonal Routing and Connection Optimization

4.1 Problem Statement

Although the Smart Auto-Routing system was capable of generating a valid path between two components, the generated connections still had several practical limitations. In many situations, connection lines passed too close to components, intersected equipment, or produced visually inconsistent layouts. These issues reduced the readability of Process Flow Diagrams, especially in projects containing a large number of interconnected components.

For engineering diagrams, connection lines should follow a structured layout consisting of horizontal and vertical segments while avoiding unnecessary bends and overlaps. Therefore, additional improvements were required to transform the routing system into a cleaner and more professional solution.

The objective of this task was to improve the routing mechanism by implementing strict orthogonal routing, obstacle avoidance, accurate arrow alignment, and overall connection optimization.

4.2 Objectives

The primary objectives of this task were:

- Maintain strictly orthogonal (90°) connection paths.
- Prevent connection lines from passing through components.
- Align arrows accurately with component grip points.
- Reduce unnecessary bends in routing paths.
- Improve the visual appearance and readability of Process Flow Diagrams.
- Ensure routing updates correctly when components are moved.

4.3 Challenges

Several technical challenges were encountered while improving the routing system:

- Preventing connections from overlapping component boundaries.
- Preserving orthogonality while still finding a valid path.
- Correctly aligning arrowheads with source and destination grips.
- Updating existing routes dynamically after component movement.
- Maintaining routing performance for larger diagrams.

Addressing these challenges required modifications to both the routing algorithm and the rendering logic of the desktop application.

4.4 code Snippet

connection.py

```
def calculate_path(self, components=None, other_connections=None, use_auto_router=None): # Clear old geometry before recomputing self.path = [] self.painter_path = QPainterPath()

should_use_auto_router = (
    use_auto_router if use_auto_router is not None else self.use_auto_router
)

if should_use_auto_router and components:
    self._calculate_path_with_auto_router(components, other_connections)
else:
    self._calculate_path_rule_based()

# Final safety pass: reroute around component bodies
if components and self.path:
    self.path = self._reroute_around_components(self.path, components)


def _reroute_around_components(self, points: list, components: list) -> list: blocked_components = [ comp.logical_rect for comp in components if comp not in (self.start_component, self.end_component) and hasattr(comp, "logical_rect") ]

if not blocked_components:
    return points
safe_points = [QPointF(points[0])]
for index in range(1, len(points)):
    segment_points = self._build_safe_segment(
        safe_points[-1],
        QPointF(points[index]),
        blocked_components,
    )
    safe_points.extend(segment_points)
```

```
return self._ensure_orthogonal_path(safe_points)
```

main.py

```
def _reexec_with_pythonw_on_macos(): if sys.platform != "darwin": return if
os.environ.get("PFD_PYTHONW_REEXEC") == "1": return if
os.path.basename(sys.executable).lower() == "pythonw": return

pythonw_path = shutil.which("pythonw")
if not pythonw_path:
    return


env = os.environ.copy()
env["PFD_PYTHONW_REEXEC"] = "1"
os.execvpe(pythonw_path, [pythonw_path, *sys.argv], env)



qt_plugin_base, qt_platform_dir = _resolve_qt_paths() if qt_plugin_base:
os.environ["QT_PLUGIN_PATH"] = qt_plugin_base if qt_platform_dir:
os.environ["QT_QPA_PLATFORM_PLUGIN_PATH"] = qt_platform_dir
```

4.5 Maintaining Strict Orthogonal Routing

One of the major improvements introduced during the internship was enforcing **strict orthogonal routing**.

Instead of allowing arbitrary line directions, every connection was constrained to horizontal and vertical segments only. This approach follows the conventions used in professional Process Flow Diagrams, making diagrams cleaner and easier to understand.

The routing logic was modified to ensure that every generated connection maintained right-angle turns while preserving the shortest feasible route.

Major improvements included:

- Horizontal and vertical routing only.

- Consistent right-angle turns.
- Reduction of unnecessary direction changes.
- Improved visual consistency.

4.6 Preventing Connections from Passing Through Components

A significant issue observed during development was that connection lines occasionally passed directly through component bodies. This reduced diagram clarity and could create ambiguity regarding equipment connections.

To resolve this issue, the routing engine was enhanced to consider component boundaries while generating paths.

During route generation:

- Components were treated as restricted regions.
- The routing algorithm searched for alternative paths around obstacles.
- Existing component boundaries were respected before drawing connections.

As a result, generated routes automatically navigated around components instead of intersecting them.

4.7 Connection Arrow Alignment

Another improvement involved correcting the alignment of connection arrows.

Initially, arrowheads were not perfectly aligned with the component grip points, causing slight visual offsets. Although functionally correct, these inconsistencies affected the professional appearance of engineering diagrams.

The alignment logic was refined so that:

- Arrowheads terminate exactly at grip locations.
- Connection endpoints remain synchronized after component movement.
- Visual consistency is maintained across different component sizes.

This enhancement significantly improved the overall appearance of the generated diagrams.

4.8 Dynamic Route Updates

The routing system was further enhanced to support automatic route updates whenever components were repositioned.

Instead of requiring users to delete and recreate existing connections, the application recalculated routing paths whenever connected components changed position.

This functionality ensured that:

- Connections remained valid after movement.
- Manual adjustments were minimized.
- Diagram consistency was preserved throughout editing.

4.9 Routing Optimization

Several optimizations were introduced to improve both routing quality and performance.

The optimization process focused on:

- Reducing unnecessary bends.
- Eliminating redundant path segments.
- Improving route smoothness.
- Preserving orthogonality.
- Maintaining fast routing performance for complex diagrams.

These refinements resulted in cleaner Process Flow Diagrams while keeping the routing algorithm responsive.

4.10 Related Development Work

The routing improvements were implemented across multiple development iterations during the internship.

Major milestones included:

Development Activity	Purpose
Smart Auto Routing	Automatic path generation
Strict Orthogonal Routing	Maintain 90° connections
Obstacle Avoidance	Prevent routing through components
Arrow Alignment	Improve connection accuracy
Routing Cleanup	Remove unnecessary path segments
Dynamic Route Updates	Update routes after component movement

This iterative approach allowed continuous improvement of the routing engine while maintaining application stability.

4.11 Results

After implementing these improvements, the desktop application generated significantly cleaner and more reliable Process Flow Diagrams.
The routing system now provides:

- Automatic path generation.
- Strict orthogonal connections.
- Obstacle avoidance.
- Accurate arrow alignment.
- Dynamic route updates.
- Improved diagram readability.

These enhancements greatly reduced manual editing effort and improved the overall user experience.

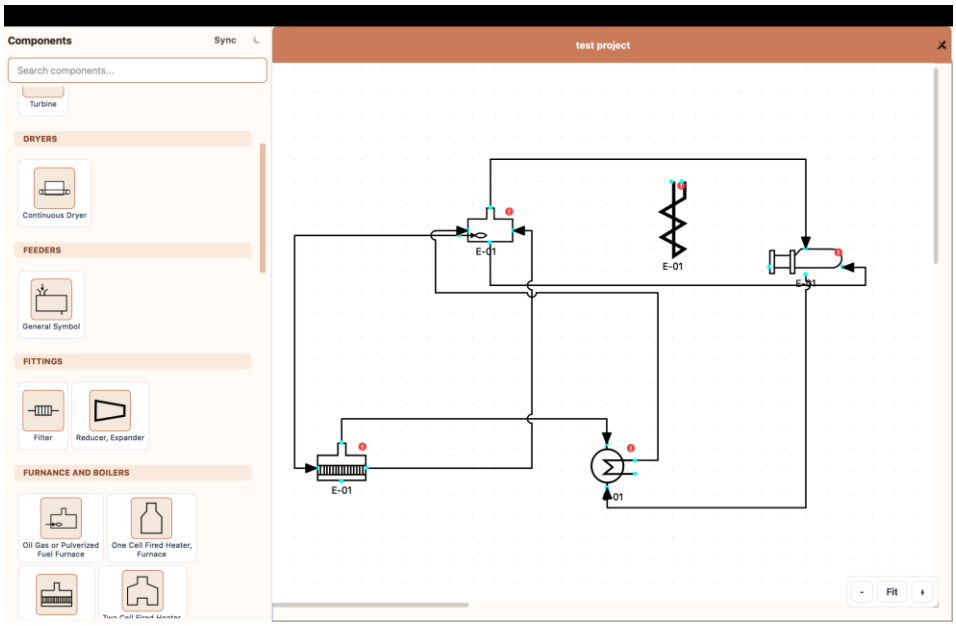


Fig 4.1 Orthogonal routing Image

4.12 Outcome

The routing optimization work transformed the desktop application's connection system into a more robust and professional solution suitable for engineering diagrams. By combining intelligent path generation with strict orthogonal routing and obstacle avoidance, the application now produces cleaner, more readable, and visually consistent Process Flow Diagrams.

These improvements formed one of the major technical contributions of my internship and significantly enhanced the usability of the Chemical PFD Builder.

Chapter 5

Desktop User Interface and Component Management

5.1 Problem Statement

As the Chemical PFD Builder evolved, several usability issues were identified in the desktop application. Users experienced difficulties while managing projects, creating custom components, navigating between screens, and interacting with the canvas. Some operations required unnecessary user effort, while others lacked confirmation or editing capabilities.

To improve the overall user experience, several enhancements were implemented in the desktop application. These improvements focused on making the application more intuitive, user-friendly, and consistent with modern desktop software.

5.2 Objectives

The main objectives of this task were:

- Improve the overall user experience of the desktop application.
- Simplify project management operations.
- Enhance the Add New Symbol interface.
- Improve Grip Editor usability.
- Enable editing and deletion of custom components.
- Improve navigation and confirmation dialogs.
- Make the landing page more informative and interactive.

5.3 Back to Home Confirmation

Initially, clicking the **Back to Home** option could unintentionally discard the user's current work if changes had not been saved. This behavior increased the risk of accidental data loss.

To address this issue, a confirmation dialog was implemented before returning to the home screen. The dialog prompts the user to confirm the action, ensuring that navigation only occurs after explicit confirmation.

Benefits

- Prevents accidental navigation.
- Reduces the possibility of losing unsaved work.

- Improves application reliability.

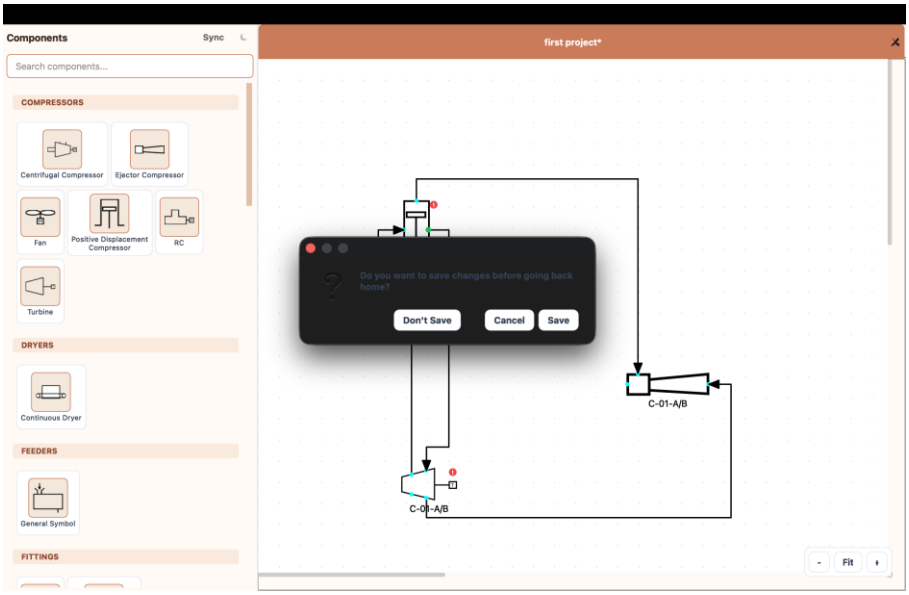


Figure 5.1 Back to home popup

5.4 Project Save Workflow Improvements

The original save mechanism did not clearly distinguish between **Save** and **Save As**, which occasionally resulted in confusion while updating existing projects or creating new ones.

The save workflow was redesigned to provide separate operations for:

- Save
- Save As

The recent project list was also updated immediately after saving, ensuring that newly saved projects appeared without requiring the application to restart.

Improvements

- Clear separation of Save and Save As.
- Automatic update of recent projects.
- Better project management experience.

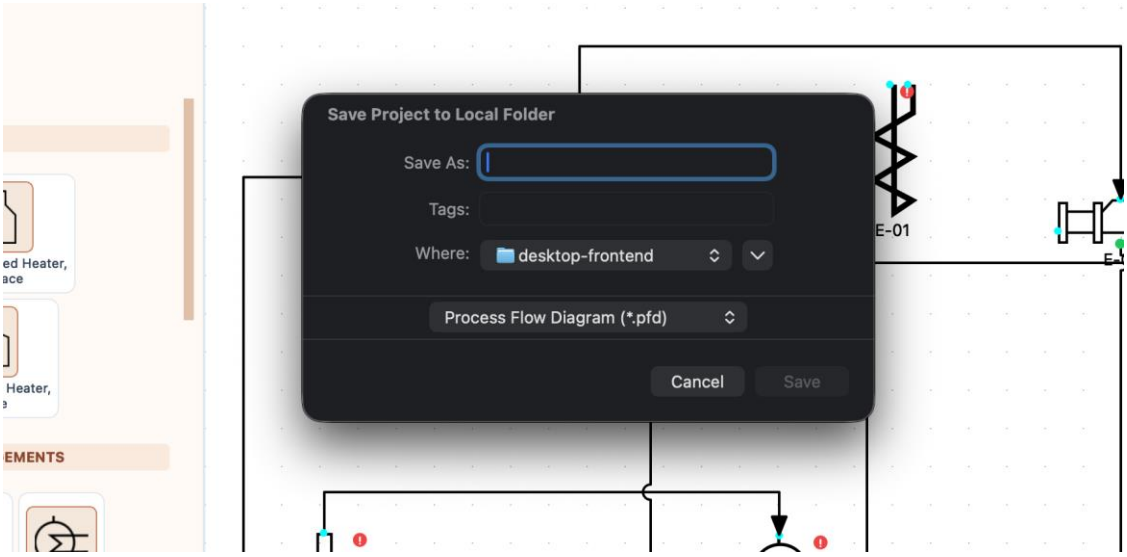


Figure 5.2 project save workflow

5.5 Landing Page Enhancements

The landing page was redesigned to improve accessibility and project management.

New improvements included:

- Displaying all recently opened projects.
- Quick access to existing projects.
- Edit option directly from the landing page.
- Delete option for project management.
- Cleaner layout and improved navigation.

These enhancements reduced the number of steps required to access frequently used projects.

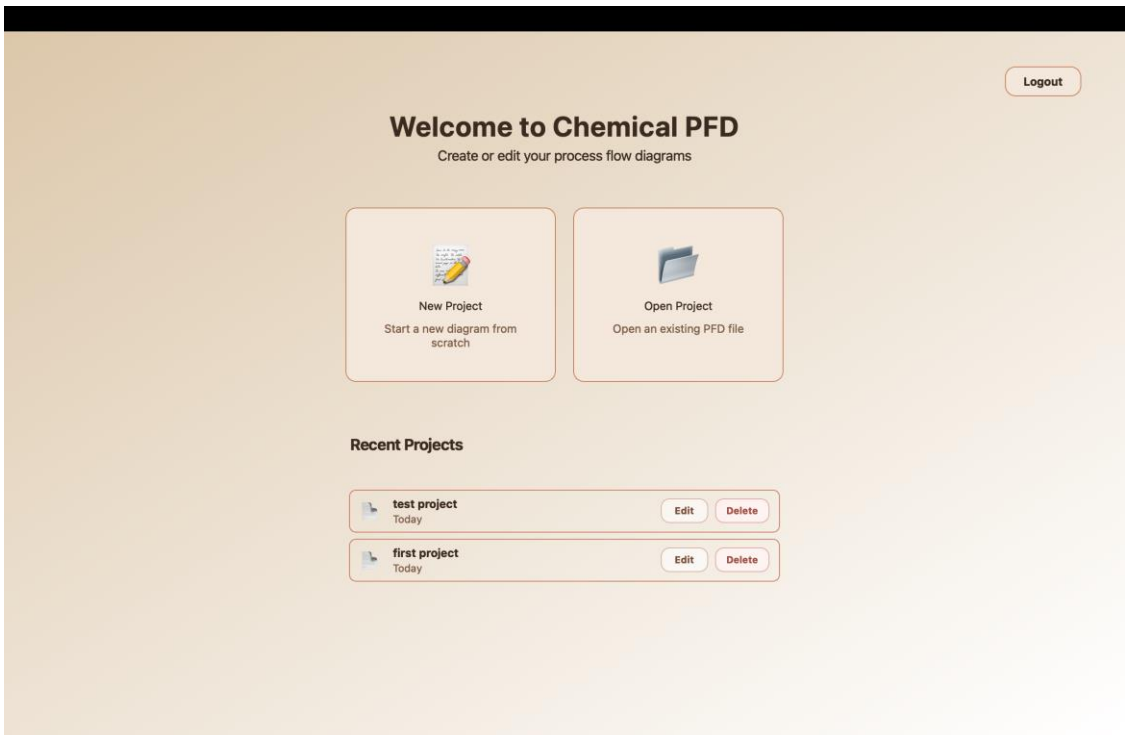


Figure 5.3 Recent Projects

5.6 Add New Symbol Interface Improvements

The **Add New Symbol** dialog is responsible for allowing users to create custom engineering components.

The previous interface required multiple manual operations and was not fully aligned with the design of the web application.

The interface was redesigned to provide:

- Improved form layout.
- Better field organization.
- Simplified component creation process.
- Better validation of user input.
- Consistent user experience.

These changes reduced user effort and made the feature easier to understand.

5.7 Grip Editor Improvements

The Grip Editor is responsible for defining connection points on custom components.

Several improvements were introduced to simplify grip management:

- Automatic grip handling.
- Improved editing workflow.
- Better visual feedback.
- Reduced manual configuration.

These enhancements made custom component creation faster and less error-prone.

5.8 Component Management

Initially, users could create custom components but had limited options for managing them afterwards.

To improve flexibility, component management was extended by introducing:

- Edit existing components.
- Delete components.
- Improved component update workflow.
- Better component organization.

This functionality made the component library easier to maintain and significantly improved usability.

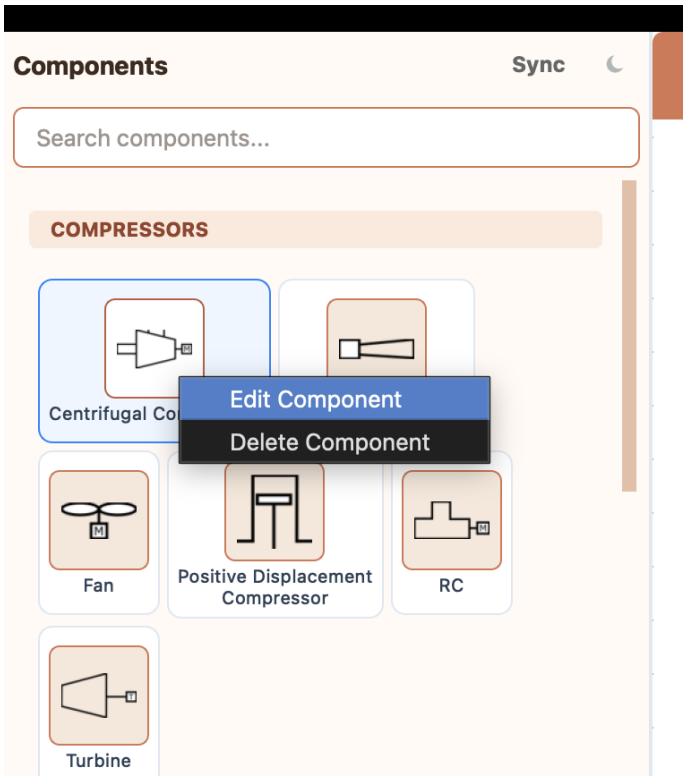


Figure 5.4 Edits Components

5.9 User Experience Improvements

Apart from the major features, several smaller usability improvements were implemented throughout the desktop application.

These included:

- Cleaner dialogs.
- Improved button placement.
- Better navigation flow.
- More responsive interactions.
- Reduced unnecessary user actions.
- Improved visual consistency.

Although individually small, these improvements collectively enhanced the overall experience of using the application.

5.10 Implementation Summary

The user interface improvements were implemented incrementally during the internship.

The major completed features are summarized below.

Feature	Purpose
Back to Home Confirmation	Prevent accidental data loss
Save / Save As Workflow	Better project management
Landing Page Improvements	Easier access to projects
Recent Projects Update	Immediate project visibility
Add New Symbol Redesign	Simplified custom component creation
Grip Editor Improvements	Faster grip configuration
Component Edit/Delete	Better component management

5.11 Results

The implemented improvements significantly enhanced the usability of the desktop application.

The application became:

- Easier to navigate.
- More intuitive for new users.

- More efficient for experienced users.
- Safer for project management.
- Better aligned with modern desktop application design principles.

These changes reduced user effort while improving the overall workflow of creating and managing Process Flow Diagrams.

5.12 Outcome

The desktop user interface enhancements improved both functionality and usability of the Chemical PFD Builder. Features such as improved project management, enhanced component editing, simplified symbol creation, and better navigation contributed to a smoother and more productive user experience.

These improvements complemented the routing enhancements developed during the internship and helped create a more polished and user-friendly desktop application.

Chapter 6

Export and Report Generation System

6.1 Problem Statement

The Chemical PFD Builder allows users to export Process Flow Diagrams for documentation, reporting, and sharing. However, the existing export functionality lacked structured project information and produced reports with limited readability. Important project details such as title, author information, and additional metadata were either missing or not displayed in a professional format.

To improve the quality of exported reports, enhancements were required to provide better document formatting, structured information, and improved readability while maintaining consistency across exported files.

6.2 Objectives

The primary objectives of this task were:

- Improve the appearance of exported reports.
- Introduce a structured footer and title block.
- Display important project information in exported documents.
- Improve readability of exported content.
- Ensure consistency across different export formats.
- Verify functionality through automated unit testing.

6.3 Footer and Title Block Implementation

One of the major improvements made during the internship was the introduction of a **Footer and Title Block** for exported reports.

The footer provides structured information that helps identify the exported Process Flow Diagram and improves the professional appearance of the document.

The title block includes important project-related details, making exported reports more informative and suitable for documentation purposes.

The implementation was integrated into the desktop application's export module without affecting the existing export workflow.

Benefits

- Professional report appearance.
- Better document organization.
- Easy identification of exported projects.
- Improved documentation quality.

6.4 Export Layout Improvements

After implementing the footer, additional improvements were made to optimize the layout of exported reports.

The spacing, positioning, and overall arrangement of export elements were refined to ensure a clean and balanced document layout.

Special attention was given to maintaining proper alignment between the Process Flow Diagram and the footer section, resulting in a more visually appealing export.

The optimized layout improved both presentation quality and usability.

6.5 Readability Enhancements

During testing, it was observed that the footer section contained limited space, making some project details difficult to read.

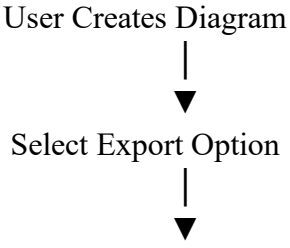
To address this issue, the footer area was redesigned with increased dimensions and improved spacing between individual information fields.

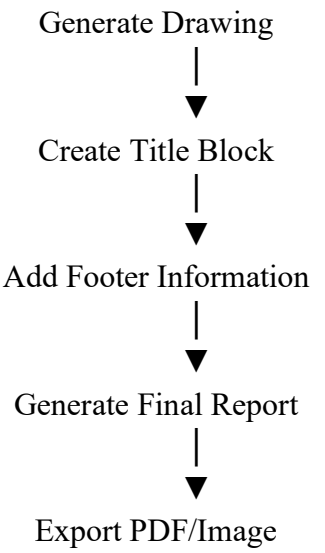
These modifications enhanced readability without affecting the overall structure of the exported document.

The improvements ensured that project information remained clear and easily understandable even in complex engineering diagrams.

6.6 Export Workflow

The export process follows the workflow shown below.





6.7 Testing the Export Module

To ensure the reliability of the new export functionality, comprehensive unit tests were developed.

The testing focused on validating:

- Footer generation.
- Title block creation.
- Export layout.
- Report formatting.
- Data consistency.
- Overall export functionality.

Automated testing helped verify that the exported reports remained correct after future code modifications.

6.8 Documentation

After completing the implementation, the project documentation was updated to describe the newly introduced export functionality.

The desktop README was revised to include:

- Export feature overview.
- Usage instructions.
- Configuration details.
- Testing information.

Maintaining proper documentation helps future contributors understand the implementation and simplifies project maintenance.

6.9 Implementation Summary

The following improvements were completed as part of the export system enhancement.

Feature	Description
Footer Section	Added structured footer to exported reports
Title Block	Displayed project information in exports
Layout Optimization	Improved positioning and spacing
Readability Enhancement	Increased footer size and clarity
Unit Testing	Verified export functionality
Documentation	Updated README with export details

6.10 Results

The implemented improvements significantly enhanced the quality of exported Process Flow Diagrams.

The export system now provides:

- Professional-looking reports.
- Better organization of project information.
- Improved readability.
- Consistent formatting.
- Reliable functionality through automated testing.

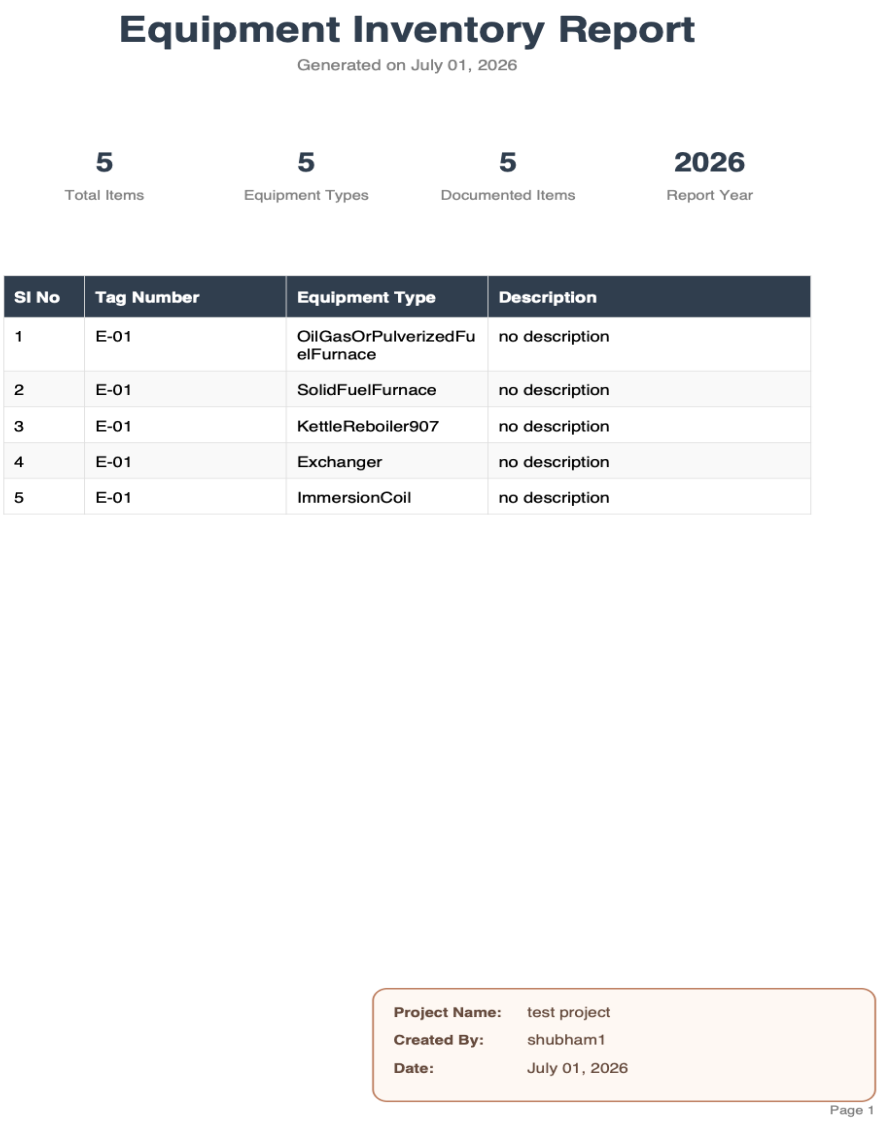
These enhancements increased the usefulness of exported documents for academic, research, and engineering purposes.

6.11 Code Ssnippet

```
def _display_name(value, default):
    if value is None:
        return default
    if isinstance(value, dict):
        for key in ("username", "name", "full_name", "email"):
```

```
            nested_value = value.get(key)
            if nested_value:
                return str(nested_value)
        return default
    text = str(value).strip()
    return text or default
def get_export_metadata(canvas):
    """Build display metadata for export and report title blocks."""
    project_name = _display_name(
        getattr(canvas, "project_name", None) or app_state.current_project_name,
        "Untitled Project",
    )
    created_by = _display_name(app_state.current_user, "Unknown User")
    export_date = datetime.datetime.now().strftime("%B %d, %Y")
    return {
        "project_name": project_name,
        "created_by": created_by,
        "date": export_date,
    }


# --- Footer Title Block ---
    footer_height = 0.95 * inch
    block_width = 3.25 * inch
    block_height = 0.72 * inch
    block_right = A4[0] - 0.5 * inch
    block_left = block_right - block_width
    block_bottom = 0.4 * inch
```

```
    block_top = block_bottom + block_height

    canvas.setStrokeColor(colors.HexColor('#C97B5A'))
    canvas.setFillColor(colors.HexColor('#FFF8F2'))
    canvas.roundRect(block_left, block_bottom, block_width, block_height, 8,
stroke=1, fill=1)

    label_x = block_left + 0.12 * inch
    value_x = block_left + 1.02 * inch
    row_y = block_top - 0.18 * inch
    row_gap = 0.18 * inch

    metadata = self.metadata or {}
    rows = [
        ("Project Name", metadata.get("project_name", "Untitled Project")),
        ("Created By", metadata.get("created_by", "Unknown User")),
        ("Date", metadata.get("date", datetime.datetime.now().strftime('%B %d, %Y'))),
    ]

    canvas.setFont('Helvetica-Bold', 7.5)
    canvas.setFillColor(colors.HexColor('#6B4A3B'))
    for label, value in rows:
        canvas.drawString(label_x, row_y, f"{label}:")
        canvas.setFont('Helvetica', 8)
        canvas.drawString(value_x, row_y, str(value))
        canvas.setFont('Helvetica-Bold', 7.5)
        row_y -= row_gap

    canvas.setFont('Helvetica', 8)
    canvas.setFillColor(colors.gray)
    canvas.drawRightString(A4[0] - 0.5 * inch, 0.28 * inch, f"Page
{canvas.getPageNumber()}")
```



Equipment Inventory Report

Generated on July 01, 2026

5	5	5	2026
Total Items	Equipment Types	Documented Items	Report Year

Sl No	Tag Number	Equipment Type	Description
1	E-01	OilGasOrPulverizedFuelFurnace	no description
2	E-01	SolidFuelFurnace	no description
3	E-01	KettleReboiler907	no description
4	E-01	Exchanger	no description
5	E-01	ImmersionCoil	no description

Project Name: test project
Created By: shubham1
Date: July 01, 2026

Page 1

Figure 6.1 Footer Section

6.12 Outcome

The Export and Report Generation System was successfully enhanced by introducing structured footer and title block support, improving document readability, refining the export layout, and strengthening software reliability through automated testing.

These improvements resulted in higher-quality exported reports while improving the overall user experience of the Chemical PFD Builder.

Chapter 7

Software Testing and Quality Assurance

7.1 Problem Statement

As new features were continuously added to the Chemical PFD Builder, it became essential to ensure that existing functionality continued to work correctly. Features such as component management, export generation, routing, and user interactions required thorough verification to prevent regressions and

maintain application stability.

Manual testing alone was not sufficient because repetitive testing after every modification was time-consuming and prone to human error. Therefore, a structured testing approach was adopted to improve software quality and verify that new changes did not introduce unexpected issues.

7.2 Objectives

The primary objectives of testing were:

- Verify the correctness of newly implemented features.
- Detect bugs before integrating changes into the main codebase.
- Ensure application stability after modifications.
- Reduce the possibility of regression errors.
- Improve confidence during future development.
- Maintain code quality throughout the project.

7.3 Unit Testing

To improve software reliability, unit tests were developed for different modules of the desktop application.

The testing process focused on validating individual components independently without affecting other parts of the system.

The implemented tests verified:

- Component Library functionality.
- Export module.
- Widget behavior.
- Register fallback mechanism.

- User interface interactions.

Writing unit tests helped identify defects during development and ensured that implemented features behaved as expected.

7.4 Component Library Testing

The Component Library is one of the core modules of the Chemical PFD Builder, responsible for loading and managing engineering symbols.

Unit tests were written to verify:

- Component loading.
- Component availability.
- Metadata validation.
- Component creation.
- Component retrieval.
- Error handling.

These tests ensured that components were correctly initialized and available for use within the application.

7.5 Widget Testing

The desktop application consists of multiple interactive widgets responsible for user interaction and project management.

Testing focused on validating:

- Widget initialization.
- User interaction.
- Event handling.
- Window behavior.
- Data updates.
- UI consistency.

The tests confirmed that the application's graphical interface behaved correctly under different user operations.

7.6 Export Module Testing

Following the implementation of the enhanced Export and Report Generation System, additional unit tests were developed to validate the export functionality.

The testing process verified:

- Footer generation.
- Title block creation.
- Export formatting.
- Report generation.
- Layout consistency.
- Output validation.

These tests ensured that exported reports remained accurate and visually consistent after future code changes.

7.7 Bug Identification and Resolution

While writing unit tests and performing regular validation, several issues were identified during development.

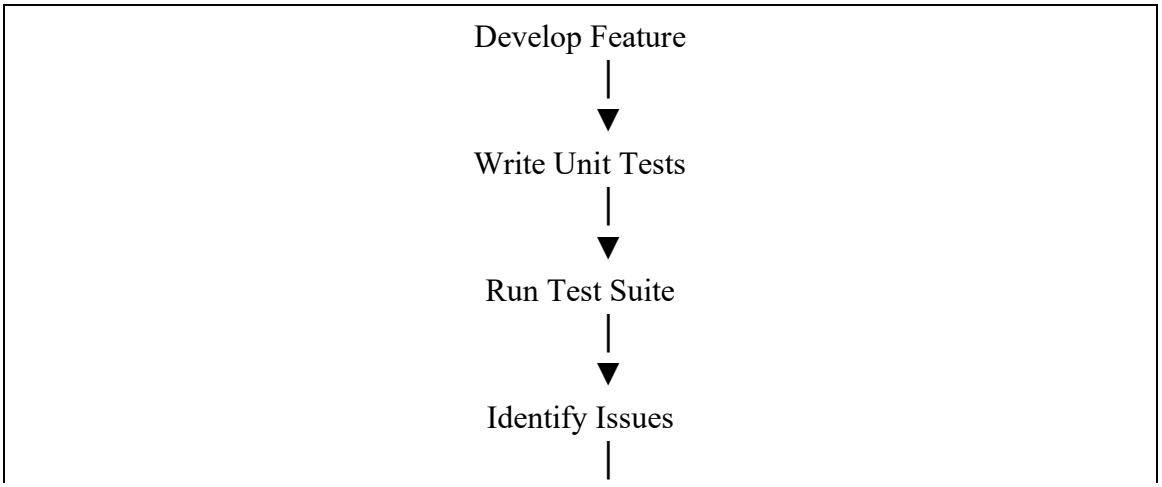
Some of the important improvements included:

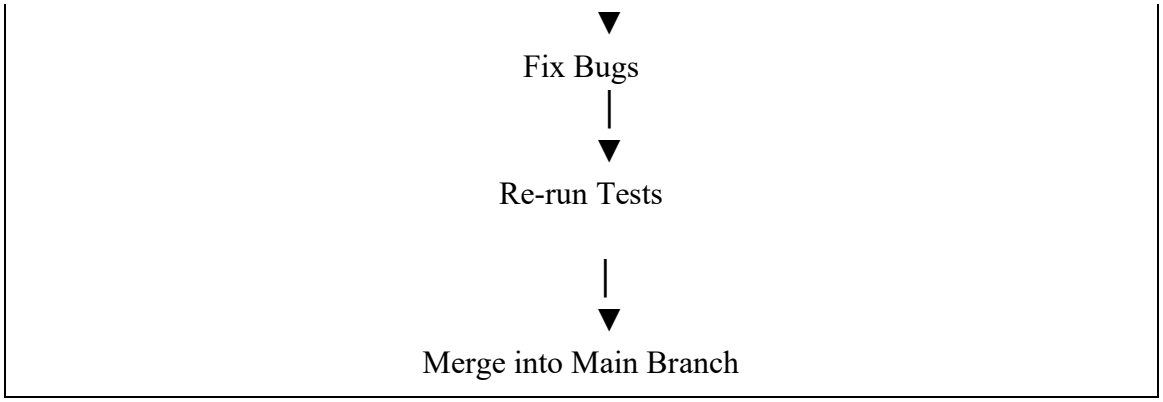
- Register fallback correction.
- Component behavior validation.
- User interface consistency improvements.
- Stability improvements.
- Export reliability enhancements.

Testing played an important role in identifying these issues before they affected end users.

7.8 Testing Workflow

The overall testing process followed the workflow shown below.





7.9 Benefits of Testing

The testing process provided several advantages:

- Improved software reliability.
- Early bug detection.
- Reduced regression issues.
- Better code maintainability.
- Increased confidence during development.
- Improved software quality.

7.10 Implementation Summary

The testing activities completed during the internship are summarized below.

Testing Activity	Purpose
Component Library Tests	Validate component management
Widget Tests	Verify desktop interface behavior
Export Module Tests	Ensure correct report generation
Register Fallback Validation	Verify fallback functionality
Bug Verification	Confirm successful issue resolution

7.11 Results

The testing process significantly improved the quality of the Chemical PFD Builder.

By introducing automated tests and systematic validation, newly developed features could be verified more efficiently, reducing development time and improving application stability.

Testing also made future modifications safer by providing confidence that previously implemented functionality continued to operate correctly.

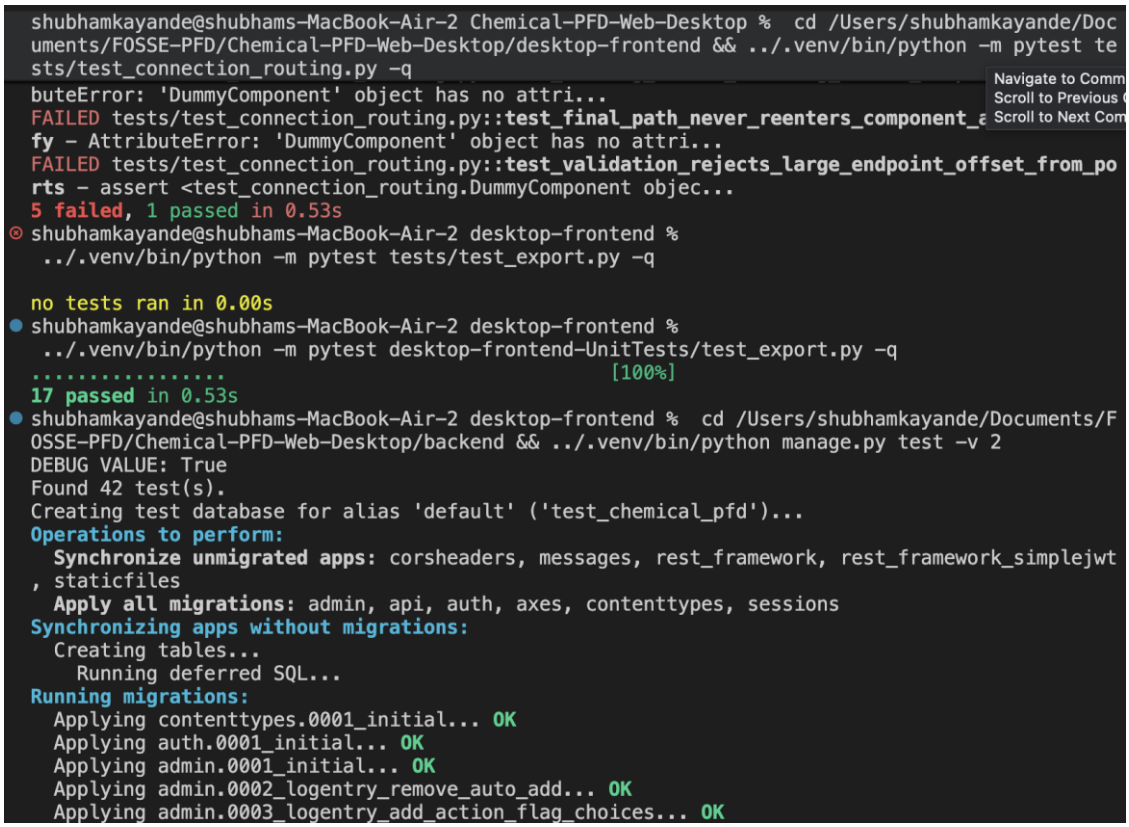


Figure 7.1 Test Report

7.12 Outcome

Software testing played an important role throughout the internship by ensuring the correctness, reliability, and stability of the desktop application. The introduction of unit tests, continuous validation, and systematic bug fixing contributed to a more maintainable and dependable software system.

These testing activities complemented the feature development work carried out during the internship and strengthened the overall quality of the Chemical PFD Builder.

Chapter 8

Development Practices and Open Source Collaboration

8.1 Introduction

One of the most valuable aspects of the FOSSEE internship was the opportunity to contribute to a real-world open-source software project. Unlike academic projects, the Chemical PFD Builder is actively developed by multiple contributors, requiring a structured development process to maintain code quality and project stability.

Throughout the internship, I followed professional software development practices, including version control using Git, collaborative development through GitHub, pull request-based contributions, technical documentation, and continuous communication with my mentor. These practices not only improved the quality of my contributions but also provided valuable exposure to industry-standard software development workflows.

8.2 Version Control Using Git

Git was used throughout the internship to manage source code efficiently and track changes made during development. Each feature or bug fix was implemented in a separate branch before being integrated into the main project.

Using Git made it possible to:

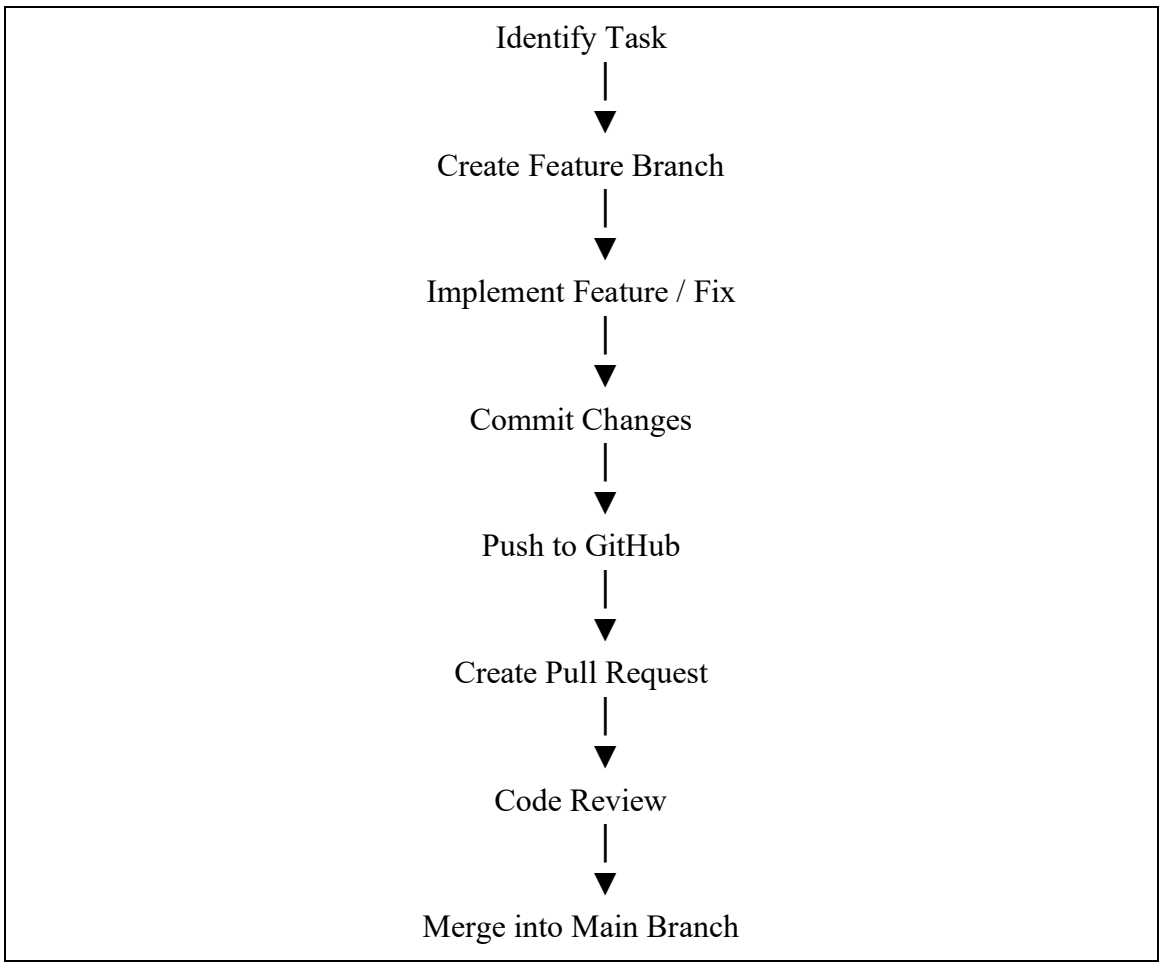
- Track code changes effectively.
- Maintain a complete development history.
- Restore previous versions when required.
- Work on independent features without affecting the main project.
- Collaborate efficiently with the development team.

Learning Git in a real project environment significantly improved my understanding of version control and collaborative software development.

8.3 Pull Request Workflow

All code contributions were submitted through GitHub Pull Requests. Instead of directly modifying the main branch, every completed feature or improvement was pushed to a separate branch and submitted for review.

The development workflow followed during the internship is shown below.



This workflow ensured that every contribution was properly reviewed before becoming part of the project.

8.4 Documentation

Maintaining clear documentation is an essential part of open-source software development. During the internship, I contributed to improving the desktop application's documentation by updating the project README.

The documentation included:

- Feature descriptions.
- Usage instructions.
- Project information.
- Development notes.
- Guidance for future contributors.

Well-maintained documentation improves project maintainability and helps new contributors understand the codebase more easily.

8.5 Collaborative Development

The internship provided valuable experience working within a collaborative software development environment.

Regular communication with my mentor helped me understand project requirements, resolve technical challenges, and improve the quality of my implementations. Feedback received during development enabled me to refine my solutions and follow better coding practices.

Working on an active open-source project also improved my understanding of how multiple developers contribute independently while maintaining a consistent and stable codebase.

8.6 Challenges Faced

Working on an existing software project presented several practical challenges that required continuous learning and problem-solving.

Some of the major challenges included:

- Understanding a large and unfamiliar codebase.
- Debugging complex routing and canvas-related issues.
- Maintaining compatibility while modifying existing functionality.
- Ensuring new features did not affect existing behavior.
- Writing reliable unit tests for newly implemented features.
- Following project coding standards and development practices.

Each challenge provided an opportunity to improve my technical knowledge and software engineering skills.

8.7 Lessons Learned

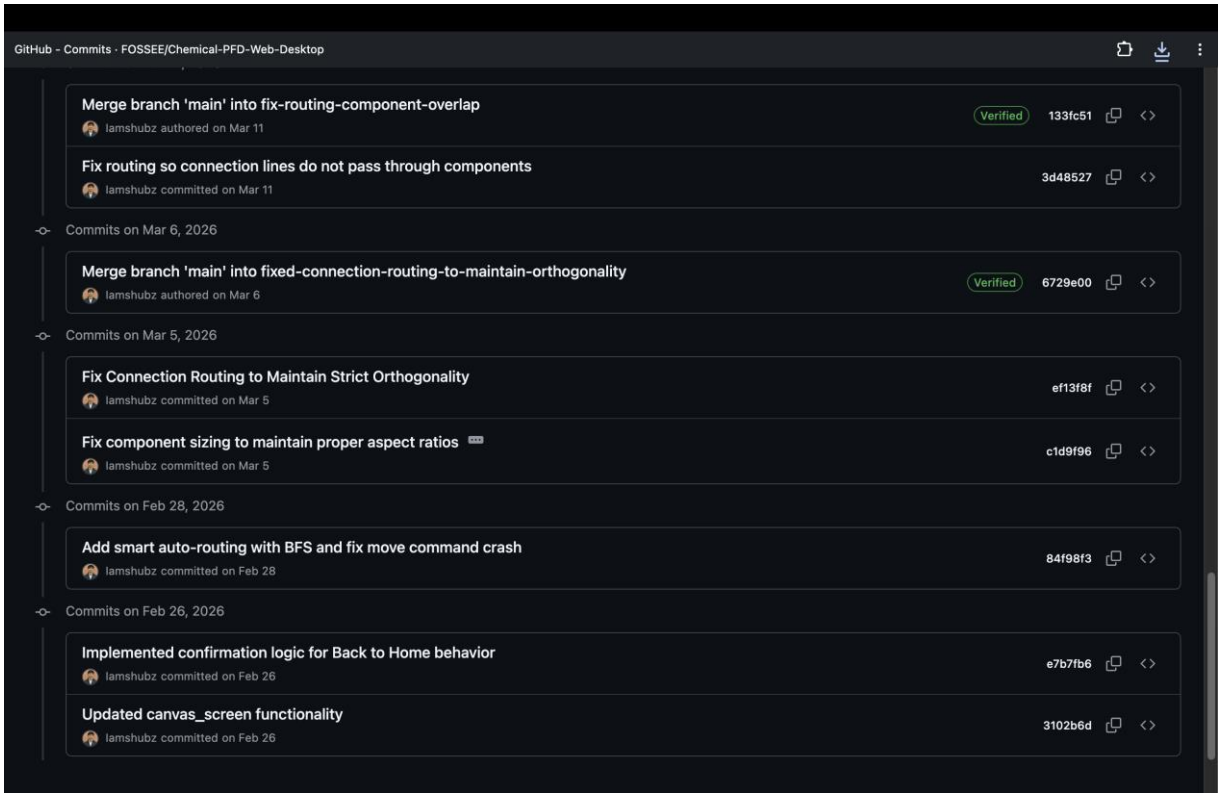
The internship offered several valuable learning experiences beyond programming.

Some of the key lessons learned include:

- Writing clean and maintainable code.
- Understanding large-scale software architecture.
- Following structured Git and GitHub workflows.
- Importance of testing before merging changes.
- Value of proper documentation.
- Importance of collaborative development and constructive feedback.

- Continuous learning while working on real-world software projects.

These experiences have strengthened both my technical foundation and my confidence in contributing to professional software development projects.



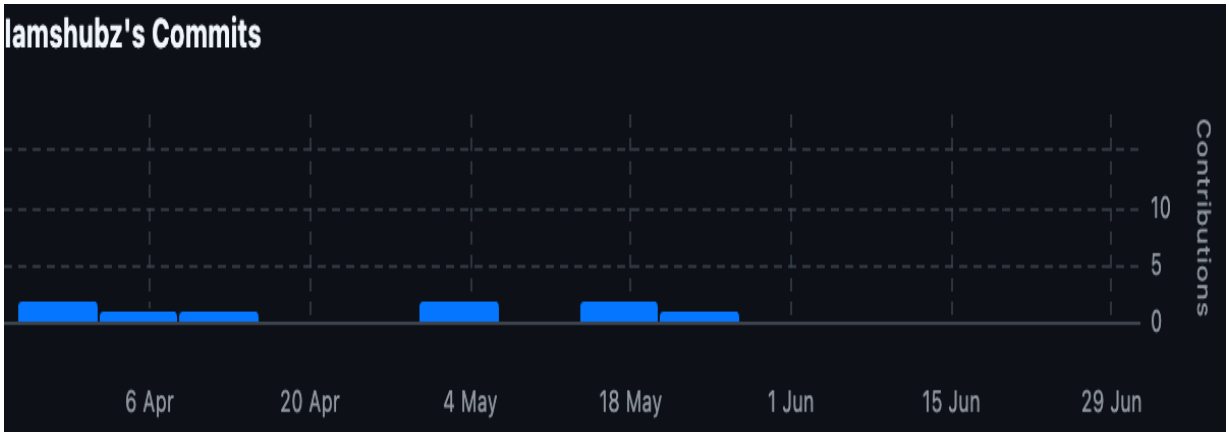


Figure 8.1 Commits Status

8.8 Outcome

The FOSSEE internship provided practical exposure to modern software engineering practices used in professional development environments. Working on an open-source project allowed me to gain experience in version control, collaborative development, documentation, testing, and feature implementation while contributing meaningful improvements to the Chemical PFD Builder.

Beyond technical knowledge, the internship improved my problem-solving ability, teamwork, communication skills, and understanding of software development workflows. These experiences have prepared me to contribute effectively to future software engineering projects in both academic and professional settings.

Chapter 9

Conclusion

9.1 Tasks Accomplished

The FOSSEE Summer Internship at **Indian Institute of Technology (IIT) Bombay** provided me with an excellent opportunity to contribute to the development of the **Chemical PFD Builder**, an open-source desktop application for designing Process Flow Diagrams. Throughout the internship, I worked on multiple modules that enhanced the functionality, usability, and reliability of the application.

One of my major contributions was improving the **connection routing system**, where I implemented Smart Auto-Routing using Breadth-First Search (BFS), enhanced strict orthogonal routing, prevented connection paths from passing through components, improved arrow alignment, and optimized the overall routing behavior.

I also contributed to the **desktop user interface** by improving component management, redesigning the Add New Symbol interface, enhancing the Grip Editor, implementing Edit/Delete functionality, improving project management workflows, and refining navigation features.

Additionally, I enhanced the **Export and Report Generation System** by introducing structured footer and title block support, improving report readability, optimizing export layouts, and validating these features through comprehensive unit testing. I also contributed to project documentation by updating the desktop README and actively followed Git-based collaborative development practices.

Overall, my internship involved feature development, debugging, testing, documentation, and continuous improvement of an open-source engineering application.

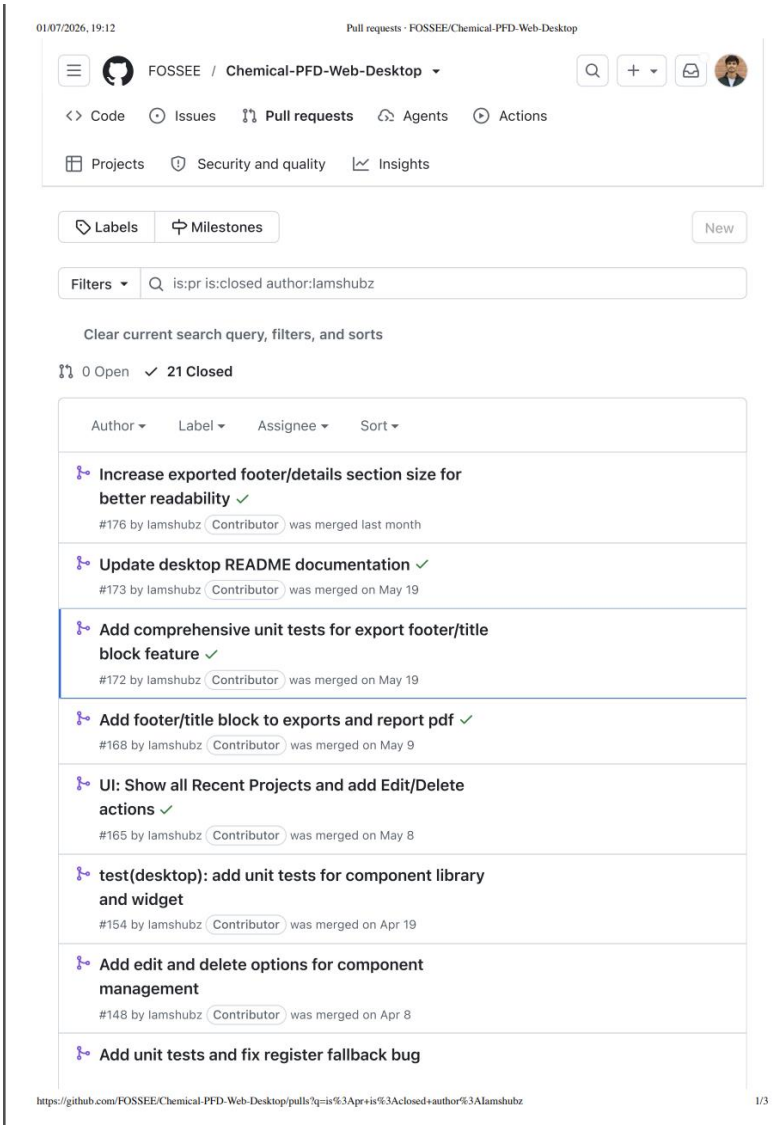


Figure 9.1 Task Accomplished

9.2 Technical Skills Developed

The internship significantly strengthened my technical knowledge by providing hands-on experience with real-world software development.

The key technical skills developed include:

- Python programming
- PyQt5 Desktop Application Development
- Graph Algorithms (Breadth-First Search)
- Orthogonal Connection Routing
- User Interface and User Experience Design
- Software Testing and Unit Testing

- Debugging and Code Refactoring
- Git and GitHub Version Control
- Open Source Software Development
- Technical Documentation

Working on an existing large-scale project also improved my ability to understand and modify complex codebases while maintaining software quality.

9.3 Professional Skills Developed

In addition to technical knowledge, the internship helped me develop several professional skills that are essential in software engineering.

These include:

- Problem-solving and analytical thinking.
- Working collaboratively in a development team.
- Understanding and following coding standards.
- Managing development tasks effectively.
- Communicating technical ideas clearly.
- Writing maintainable and well-documented code.
- Learning from code reviews and mentor feedback.
- Adapting to new technologies and project requirements.

These experiences improved my confidence in working on professional software projects and prepared me for future opportunities in the software industry.

9.4 Future Scope

The Chemical PFD Builder has significant potential for further enhancement. Some possible future improvements include:

- AI-assisted intelligent routing and diagram optimization.
- Real-time collaborative editing for multiple users.
- Advanced diagram validation and error detection.
- Integration with chemical process simulation software.
- Expanded component libraries with additional industrial symbols.
- Cloud synchronization with version history.
- Support for additional export formats and customizable report templates.

Implementing these features would further improve the application's usability and make it more suitable for educational, research, and industrial applications.

9.5 Final Remarks

The FOSSEE Summer Internship was an enriching experience that provided practical exposure to real-world software engineering and open-source development. Working on the Chemical PFD Builder allowed me to apply my academic knowledge to solve practical engineering problems while learning industry-standard development practices.

This internship not only enhanced my technical expertise but also strengthened my problem-solving, collaboration, and software development skills. The guidance provided by my mentor and the opportunity to contribute to an active open-source project have been invaluable to my professional growth.

I am grateful to the FOSSEE team, IIT Bombay, for providing this opportunity, and I believe the knowledge and experience gained during this internship will serve as a strong foundation for my future academic and professional endeavors.

Bibliography

1. FOSSEE Project Documentation.
2. Chemical PFD Builder Project Repository.
3. Python Documentation – https://docs.python.org/
4. PyQt5 Documentation – https://doc.qt.io/
5. Git Documentation – https://git-scm.com/doc
6. GitHub Documentation – https://docs.github.com/
7. Django Documentation – https://docs.djangoproject.com/