Winter Internship Report

On

Chemical PFD Builder (Desktop)

Submitted by

Parshv Keyur Modi

Under the guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

IIT Bombay

January 31, 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 Builder within the DWSIM ecosystem allowed me to dive deep into the process of desktop GUI development, canvas-based diagram editing, and integration with open-source scientific software tools.

During this internship, I had the opportunity to design and implement significant portions of the desktop frontend for the Chemical PFD Builder, making a tangible contribution to its growing open-source ecosystem. Through this hands-on work, I was able to explore the intricacies of event-driven GUI architectures and REST API integration in a practical environment, gaining insight into how software engineering practices apply to real-world scientific tools.

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 Mr. Chirag Koyande, my mentor during the internship, for his continuous guidance and support. His practical advice, clear explanations, 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 software engineering and open-source development.

Contents

Acknowledgment **1**

1 Introduction **7**
1.1 FOSSEE Winter Internship 7
1.2 DWSIM 7
1.3 Chemical PFD Builder 7
1.4 Desktop Application (PyQt5) 8
1.5 Key Features of the Desktop Application 8

2 Internship Overview **9**

3 Technologies and Tools Used **10**

4 Project Overview **11**
4.1 Project Context 11
4.2 My Role and Scope 11

5 System Architecture (Desktop Application Perspective) **12**
5.1 High-level Layered Architecture 12
5.1.1 UI Layer (Presentation Layer) 12
5.1.2 Application Logic Layer 12
5.1.3 Data and Integration Layer 13
5.2 Event-driven PyQt Architecture 13
5.3 QStackedWidget Navigation Logic 14
5.4 Canvas Subsystem Architecture 14
5.4.1 Event Processing Flow 14
5.4.2 Rendering Pipeline 15
5.4.3 Grip Editing Integration 15
5.4.4 State and Export Integration 15
5.5 Save Workflow and State Management 15
5.5.1 Dirty State Tracking 16
5.5.2 Manual Save Flow 16
5.5.3 Discard Flow 16

5.5.4 Application Close Protection 16
5.6 API Client Integration 16

6 Development Phases **18**
6.1 Phase 1: Authentication System and Base Architecture (Dec 9 – Dec 11) 18
6.1.1 Problem 18
6.1.2 What Was Implemented 18
6.1.3 Why This Mattered 19
6.1.4 Commit Metrics 19
6.2 Phase 2: Canvas Integration and Component System (Dec 13 – Dec 17) . 19
6.2.1 Problem 19
6.2.2 What Was Implemented 19
6.2.3 Why This Mattered 20
6.2.4 Commit Metrics 20
6.3 Phase 3: Symbol Dialog and Component Library Expansion (Dec 23 – Dec 24) 20
6.3.1 Problem 20
6.3.2 What Was Implemented 20
6.3.3 Why This Mattered 20
6.4 Phase 4: Grip Editor and Precision Controls (Late Dec – Jan 3) 20
6.4.1 Problem 21
6.4.2 What Was Implemented 21
6.4.3 Why This Mattered 21
6.5 Phase 5: Theme System, Runtime Fixes, Export, and Workflow (Late Dec – Jan) 21
6.5.1 Problems 21
6.5.2 What Was Implemented 21
6.5.3 Why This Mattered 22

7 Features Implemented **23**
7.1 Authentication UI and Local Database 23
7.2 Canvas Subsystem 23
7.3 Component Library and Component Widget 23
7.4 Grip Editor with Live Preview 23
7.5 Centralized Theme Management 23
7.6 Save Workflow and Smart Discard 24
7.7 High-quality Export Improvements 24

8 Application Screenshots **25**
8.1 Login Screen 25
8.2 Canvas with Components 26
8.3 Grip Editor Dialog 26
8.4 Light and Dark Theme 27

9 Bugs Fixed & Technical Improvements **28**
9.1 Incorrect Grip Positions . . . 28
9.2 Qt Runtime Errors . . . 28
9.3 Backend Save File Mismatch . . . 28
9.4 Cross-platform UI Inconsistencies . . . 29

10 Code Evolution & Commit Analysis **30**
10.1 Development Timeline . . . 31
10.2 Feature Contribution Summary . . . 32
10.3 Bug Fix Summary . . . 32
10.4 Commit Classification . . . 33
10.5 Contribution Summary Table . . . 33
10.6 Personal Code Growth (Insertions / Deletions) . . . 33

11 Challenges Faced **34**

12 Conclusion & Learning Outcomes **35**
12.1 Summary . . . 35
12.2 Technical Learning Outcomes . . . 35
12.3 Recommended Next Steps . . . 35

A Complete Commit References **37**

List of Figures

5.1 Layered Desktop Architecture (UI → Application Logic → Data & Integration). 13
5.2 Canvas Rendering and Interaction Subsystem. 14
5.3 Manual Save and Smart Discard Workflow. 16

8.1 Login Screen — Authentication UI with input validation and toast notifications. 25
8.2 Canvas Screen — Components placed and connected on the interactive PFD canvas. 26
8.3 Grip Editor — default grip positions; live preview shows grips at component edges. 26
8.4 Grip Editor — after dragging grips inward; live preview reflects updated anchor positions immediately. 26
8.5 Light Theme. 27
8.6 Dark Theme. 27

List of Tables

10.1 Development Timeline (Selected Commits) 31
10.2 Feature Contributions by Area 32
10.3 Major Bugs Fixed 32
10.4 Classification of Selected Commits 33
10.5 Overall Contribution Counts 33

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

As part of its mission, FOSSEE conducts the Winter Internship Program, which provides students an opportunity to contribute to active open-source projects. The program aims to develop students' technical skills, encourage collaborative development practices, and inspire long-term involvement in the open-source community. I was selected for this internship through a selection process that involved technical evaluations and task submissions.

1.2 DWSIM

DWSIM is an open-source chemical process simulator developed for modelling, simulating, and optimizing chemical processes. It is used by chemical engineers, researchers, and students as a powerful alternative to proprietary simulators. DWSIM supports a wide range of unit operations, thermodynamic models, and process configurations, making it a comprehensive tool for chemical process engineering.

1.3 Chemical PFD Builder

The Chemical PFD Builder is a hybrid system (Web + Desktop) built within the DWSIM ecosystem, intended for chemical engineers to create, annotate, and export Process Flow

Diagrams (PFDs). It enables engineers to visually design and document chemical processes using an interactive canvas, a symbol/component library, and a structured save and export workflow. The Desktop Application is developed using PyQt5 and interoperates with a Django REST backend for persistence and collaboration.

1.4 Desktop Application (PyQt5)

The Desktop Application is a PyQt5-based client that provides a native desktop interface for the Chemical PFD Builder. It allows users to work with PFDs offline, manage components and symbols, draw and connect process elements with precision grip anchors, and export finished diagrams as high-quality PDFs or images. The desktop client communicates with the shared Django REST backend, ensuring consistency with the web frontend while offering a responsive, platform-native user experience.

1.5 Key Features of the Desktop Application

The Desktop Application is designed to provide a professional, production-grade environment for chemical process diagram authoring. Some of its key features include:

- **Interactive Canvas with Event-driven Rendering**: A PyQt5-based canvas that supports drag-and-drop component placement, mouse-driven interactions, and keyboard shortcuts, with a rendering pipeline that updates in real time.
- **Component and Symbol Library**: A structured library of chemical engineering symbols loaded from JSON assets, allowing users to instantiate and connect process components directly onto the canvas.
- **Grip Editor with Live Preview**: A dedicated editor for adjusting component connection anchor points (grips), with live preview so that positional changes are reflected on the canvas immediately.
- **Centralized Theme Management**: A unified light/dark theme system that applies consistent styling across all screens and dialogs via Qt Style Sheets (QSS).
- **Manual Save Workflow and Smart Discard Logic**: A controlled save system with dirty-state tracking, unsaved-changes protection on close, and explicit save/discard transitions integrated with the backend API.
- **High-quality Export Pipeline**: Export functionality that generates high-resolution image and PDF outputs with correct coordinate fidelity, integrated with both local and backend export endpoints.
- **Cross-platform Compatibility**: Defensive UI sizing, platform-aware styling, and tested compatibility across Windows, Linux, and macOS.

Chapter 2

Internship Overview

Duration:	2 Months (December 2025 – January 31, 2026)
Project:	Chemical-PFD-Web-Desktop
Role:	Frontend Developer (Desktop Version – PyQt5)
Team Size:	3 frontend developers (Desktop), 1 backend developer
Mentor:	Mr. Chirag Koyande
Academic Guide:	Prof. Prabhu Ramachandran

This document is a formal technical report of my personal contributions during a two-month part-time Winter Internship working on the Desktop Application of the Chemical PFD Builder (DWSIM ecosystem). The Desktop Application is a PyQt5-based client that shares persistence and export APIs with the web frontend through a Django REST backend. The objective of my work was to design, implement, and harden frontend features and interactions that enable professional Process Flow Diagram (PFD) creation, editing, and export on desktop platforms.

My responsibilities focused entirely on frontend architecture, UI/UX enhancements, canvas system implementation, component management, precision editing tools, theming, export systems, and workflow management features.

This report concentrates on:

- Features implemented by me (UI and canvas-level)
- Bugs fixed by me and technical improvements
- Architecture decisions and code-level modifications I introduced
- Commit-based code evolution and analysis
- Challenges encountered and learning outcomes

Chapter 3

Technologies and Tools Used

The following technologies and tools were used throughout the project:

- Python
- PyQt5
- Qt Designer (.ui files)
- SQLite (initial authentication phase)
- Pandas (added in commit `4cffcda`)
- OpenPyXL (added in commit `16c8a24`)
- Git and GitHub
- REST API client integration
- Django REST Framework (backend, provided by backend developer)

Chapter 4

Project Overview

4.1 Project Context

The Chemical PFD Builder is a hybrid system (Web + Desktop) intended for chemical engineers to create, annotate, and export process flow diagrams. Desktop functionality includes authoring PFDs offline, interacting with a symbol/component library, precise placement of connection grips, and exporting diagrams as high-quality PDFs/images. The Desktop Application developed using PyQt5 must interoperate with a Django REST backend for persistence. The project goals and team responsibilities were defined in the project description and team document.

4.2 My Role and Scope

I was a member of the Desktop Application Team (3 members). My scope was exclusively frontend (desktop) and included:

- Implementing authentication UI and local auth integration (initially)
- Integrating and developing the interactive canvas subsystem
- Building the component/symbol library and component widget
- Implementing the Grip Editor and live grip preview
- Centralising theme management (light/dark)
- Designing the save workflow (manual save, smart discard)
- Improving export functionality (high-quality export)
- Fixing runtime errors and improving cross-platform compatibility
- Integrating with the backend API client (save/load/export)

Chapter 5

System Architecture (Desktop Application Perspective)

This chapter documents the architecture of the Desktop Application from the viewpoint of the components I designed and modified.

5.1 High-level Layered Architecture

The Desktop Application follows a layered architecture consisting of three principal layers.

5.1.1 UI Layer (Presentation Layer)

This layer consists of all PyQt5-based screens and dialog interfaces including: Login Screen, Signup Screen, Landing Page, Canvas Screen, Symbol Dialog, Grip Editor, and Theme Manager. The UI layer is responsible purely for rendering and user interaction. It delegates business logic to the application layer and does not directly communicate with backend services.

5.1.2 Application Logic Layer

This layer implements the core functional behavior of the system. It includes:

- Navigation Controller – Handles page transitions with slide animations
- App State Manager – Maintains global application state
- Component Library – Manages symbol definitions
- Canvas Rendering Engine – Controls drawing logic
- Save Workflow Manager – Tracks unsaved state
- Export Manager – Handles export generation

This layer ensures that business logic remains decoupled from UI rendering.

5.1.3 Data and Integration Layer

This layer manages communication with external systems:

- API Client – Communicates with backend server
- Backend Server – Handles persistent data storage
- SQLite Database (Initial Phase) – Used for early authentication
- File System – Manages local save and export operations

This separation ensures extensibility and backend interchangeability.

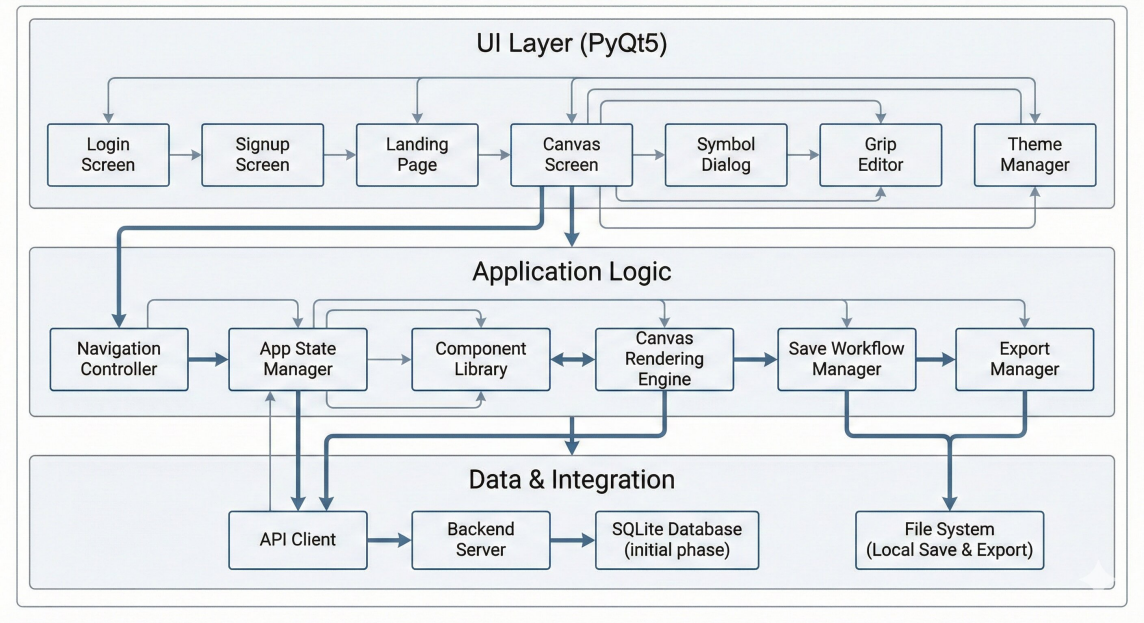

Figure 5.1: Layered Desktop Architecture (UI → Application Logic → Data & Integration).

5.2 Event-driven PyQt Architecture

Key architectural patterns used and implemented:

- **QStackedWidget-based navigation**: The application uses a global QStackedWidget instance (exposed via `app_state`) and a navigation controller that implements slide animations between screens (`src/navigation.py`). This enables clean separation of screens and animated transitions while keeping application state centralized.
- **App State Manager**: A small module `src/app_state.py` holds global variables used across screens (current theme, reference to the stacked widget), enabling theme broadcast and centralized state mutation.
- **Event-driven Canvas**: The canvas is implemented as a PyQt widget with an Event Handler that processes user input (mouse events, keyboard shortcuts). Events are dispatched to the Render Engine which updates component geometry and triggers canvas refreshes.

- **Separation of Rendering and Business Logic**: The Canvas Rendering Engine focuses on geometry and painting; the Component Library and App State Manager handle component definitions and application-level state respectively.

5.3 QStackedWidget Navigation Logic

Navigation is implemented using slide animations with `QPropertyAnimation` on widget geometry. The navigation helper (`slide_to_index(target_index, direction)`):

- Validates current vs target index to avoid redundant animations
- Prepositions the next widget off-screen and animates both current and next positions using InOutCubic easing
- Ensures animations are referenced from the stacked widget to avoid premature garbage collection
- Sets the stacked widget current index upon animation completion

5.4 Canvas Subsystem Architecture

The canvas subsystem represents the core interactive engine of the desktop frontend. It follows an event-driven rendering architecture.

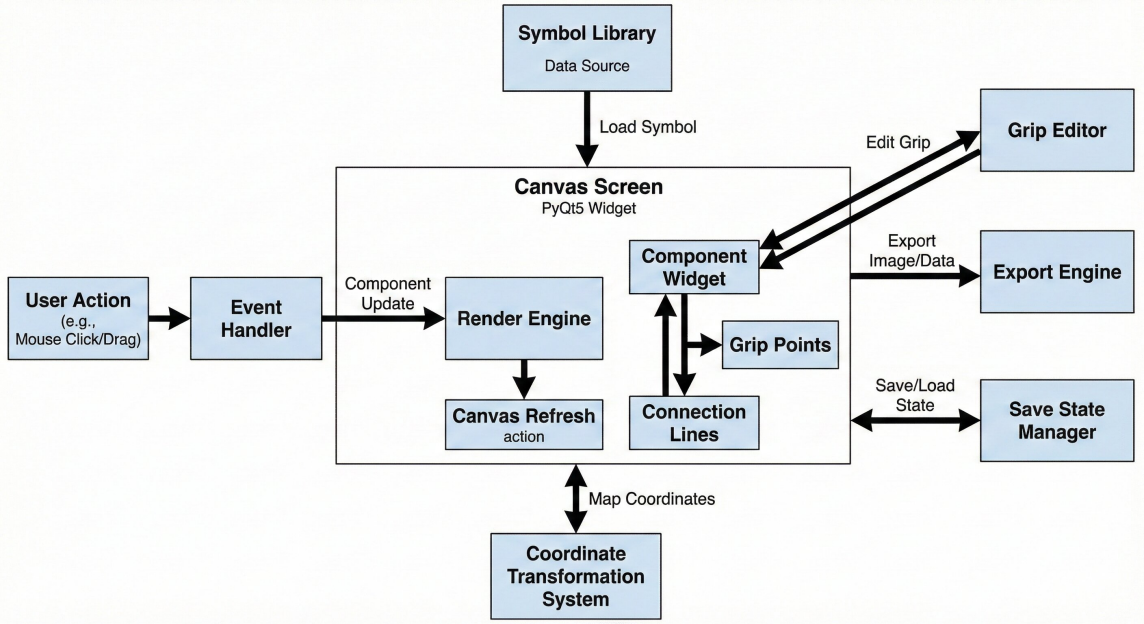

Figure 5.2: Canvas Rendering and Interaction Subsystem.

5.4.1 Event Processing Flow

User interactions such as mouse clicks, drag events, or keyboard shortcuts are captured by the Event Handler. These events trigger state updates within the canvas subsystem.

5.4.2 Rendering Pipeline

The rendering pipeline follows the sequence:

1. User Action
2. Event Handler
3. Component Update
4. Render Engine Invocation
5. Canvas Refresh

The Render Engine recalculates component geometry, grip alignment, and connection paths before refreshing the visual canvas.

5.4.3 Grip Editing Integration

The Grip Editor operates as an external editing module that modifies component anchor points. Upon modification, updated grip coordinates are recalculated, coordinate transformation is applied, and the Component Widget is re-rendered.

5.4.4 State and Export Integration

The Save State Manager tracks all modifications and marks the canvas as "dirty" when changes occur. The Export Engine retrieves canvas data for high-quality output generation.

5.5 Save Workflow and State Management

The save system follows a controlled state-transition workflow to prevent data loss and enforce user confirmation.

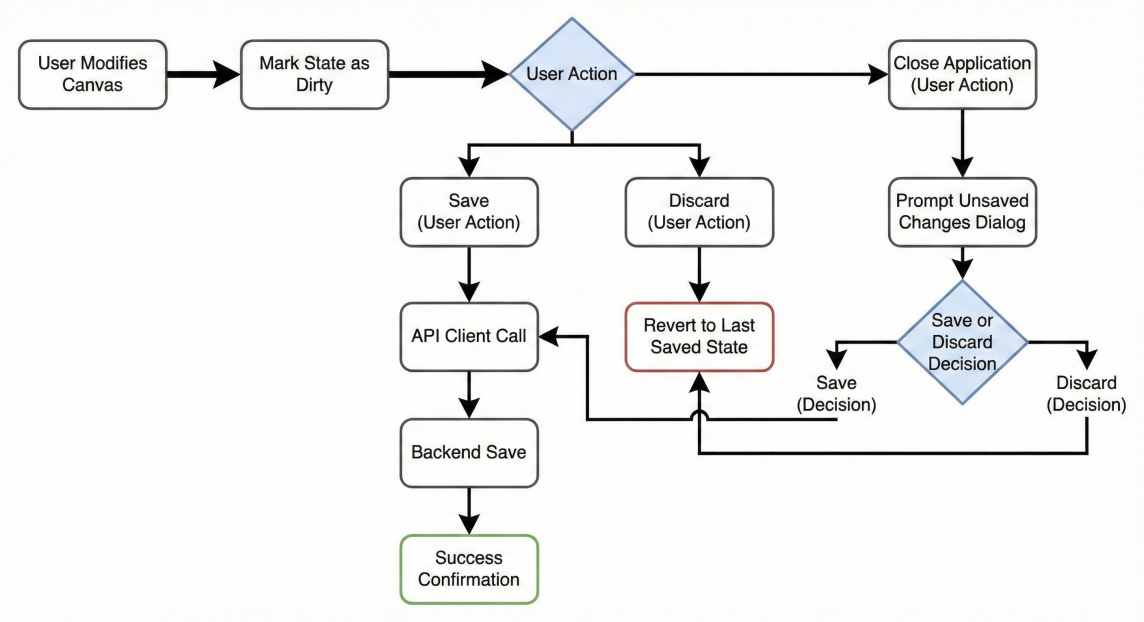

Figure 5.3: Manual Save and Smart Discard Workflow.

5.5.1 Dirty State Tracking

When a user modifies the canvas, the system marks the application state as "dirty". This flag indicates unsaved changes are present.

5.5.2 Manual Save Flow

When the user selects Save: (1) API Client call is triggered, (2) backend persistence occurs, (3) success confirmation is returned, and (4) the dirty state flag is cleared.

5.5.3 Discard Flow

When the user selects Discard, the canvas state reverts to the last saved version and the dirty flag is cleared.

5.5.4 Application Close Protection

When closing the application with unsaved changes, an unsaved changes dialog is shown and the user must explicitly Save or Discard. This structured workflow ensures data integrity and improves user trust.

5.6 API Client Integration

I implemented and extended the desktop API client to:

- Authenticate with backend endpoints

- Save and load project JSON (upload/download diagram data)
- Trigger backend export endpoints (server-side PDF generation) or perform local export if needed

Chapter 6

Development Phases

This chapter groups my commits into logical development phases describing the technical problems, the code-level changes made, and their significance. Grouping is based on the commit history from my Git contributions, analyzed from commit messages, file-level diffs, and merge records.

6.1 Phase 1: Authentication System and Base Architecture (Dec 9 – Dec 11)

Representative commits: `3feb069`, `4367f47`, `a6201bc`

6.1.1 Problem

Initial desktop app lacked user onboarding screens and local storage. Early prototypes were unstable across platforms (Windows/Linux/macOS).

6.1.2 What Was Implemented

- Implemented `main.py` that creates a global `QApplication` and a `QStackedWidget` to host screens, exposed via `src/app_state.py`.
- Implemented local SQLite initialization function `create_db_table` in `src/db.py` to provide an initial authentication DB schema (table `login_info`), including creation of local database (`app_users.db`), unique username constraints, input validation, password confirmation logic, and error handling for duplicate users.
- Added login and signup screens in `src/screens.py` with UI logic, validation, and toast integration via `loadUi` from PyQt5.
- Implemented `src/navigation.py` with slide animation (`slide_to_index`) and animation object lifetime management.
- Implemented toast notifications (`src/toast.py`) for immediate feedback on actions.

- Added fullscreen support and resolved UI inconsistencies across operating systems (`4367f47`, `a6201bc`).

Files created: `main.py`, `db.py`, `screens.py`, `toast.py`, `navigation.py`

6.1.3 Why This Mattered

Providing a robust login/signup flow and consistent navigation was a prerequisite for other features (user profile, recent projects, cloud sync) and stabilized the user experience during later backend integration.

6.1.4 Commit Metrics

The initial authentication commit introduced 12 files changed and approximately 1626 insertions across UI and Python modules.

6.2 Phase 2: Canvas Integration and Component System (Dec 13 – Dec 17)

Representative commits: `2617669`, `8421ae9`, `c230c2e`

6.2.1 Problem

A basic canvas shell existed but lacked a structured component library, rendering engine, and robust component widget. Cross-team features required a desktop-side component library consistent with backend/component catalogs.

6.2.2 What Was Implemented

- Created `src/canvas_screen.py` — a PyQt widget that hosts the canvas, handles mouse interactions, and integrates the component widget.
- Implemented `src/component_library.py` — a management layer for symbol metadata, loading from local assets (JSON) and exposing programmatic access for canvas instantiation. Added asset `ui/assets/grips.json` and styling QSS.
- Implemented `src/component_widget.py` — the per-component visual and interactive element with grip anchors and connection logic.
- Redesigned the landing page with custom QSS styling and improved asset handling (`c230c2e`).
- Resolved multiple merge conflicts while integrating component library changes.

6.2.3 Why This Mattered

A structured component library and component widget are essential to achieve consistent rendering and user interactions. This work enabled drag-and-drop, connection creation, and prepared the canvas for advanced editing features.

6.2.4 Commit Metrics

Canvas integration commit recorded approximately 363 insertions and 23 deletions. Subsequent landing page and styles finalization added approximately 1399 insertions and 53 deletions.

6.3 Phase 3: Symbol Dialog and Component Library Expansion (Dec 23 – Dec 24)

Representative commits: `5462167`, `b93e0d4`, `e411d99`

6.3.1 Problem

Users needed to select symbols from a shared library and the app needed to call backend APIs for symbol metadata. The canvas needed keyboard shortcuts to improve UX.

6.3.2 What Was Implemented

- Implemented the Symbol Dialog (`add_symbol_dialog.py`) with UI and logic to fetch and render symbol metadata, integrating backend API calls and loading animations.
- Updated the API client integration to fetch symbols and support future project save/load operations.
- Implemented keyboard shortcuts and canvas-level handlers for common operations (`e411d99`).
- Enhanced component library extensibility (`b93e0d4`).

6.3.3 Why This Mattered

Symbol Dialog and API Client integration prepared the Desktop app for cloud sync and ensured the symbol selection UI matched backend-provided libraries.

6.4 Phase 4: Grip Editor and Precision Controls (Late Dec – Jan 3)

Representative commits: `7e596b9`, `161f1e4`, `3a9ed58`, `43cf790`, `7987c64`, `571805d`

6.4.1 Problem

Complex components require adjustable grip points (anchors) to fine-tune connection endpoints. Initial implementation produced incorrect coordinates under transforms (translation/scale), producing misaligned connectors.

6.4.2 What Was Implemented

- Implemented a dedicated Grip Editor dialog allowing users to adjust component grip points, modify connection anchors, and preview changes in real time (`7e596b9`).
- Implemented coordinate transformation mapping between widget-local coordinates and canvas world coordinates (Coordinate Transformation System), critical for zoom/pan and export fidelity.
- Added live preview logic in the Grip Editor to show re-rendered component positions immediately as grips change (`3a9ed58`).
- Fixed multiple coordinate calculation bugs (`43cf790`, `7987c64`, `571805d`) by adjusting transformation math and anchor offset handling in `component_widget.py` and `canvas_screen.py`.

6.4.3 Why This Mattered

Grip accuracy is essential for engineering diagrams — connectors must attach precisely to component ports. The coordinate fixes and live preview greatly improved reliability and user confidence.

6.5 Phase 5: Theme System, Runtime Fixes, Export, and Workflow (Late Dec – Jan)

Representative commits: `1f7f74d`, `59c6f9d`, `c1897a2`, `f92d23b`, `b7fbba0`

6.5.1 Problems

- Inconsistent theme application across screens
- Runtime errors on certain UI lifecycle events
- Exports needed higher fidelity (images/PDF) and consistent backend interaction
- Need for safe-save behavior to avoid data loss

6.5.2 What Was Implemented

- Centralized theme logic into `src/theme.py` with utilities `apply_theme_to_screen` and `apply_theme_to_all`, providing a single source of truth for theme property and QSS application across all screens with light/dark mode toggling (`1f7f74d`).

- Resolved Qt runtime errors by refining widget object lifetimes and guarding theme application (`59c6f9d`, `412f9e2`).
- Implemented export improvements to generate high-quality image/PDF outputs with proper coordinate fidelity, integrated with backend export endpoints (`c1897a2`).
- Implemented Recent Projects, Manual Save Workflow, and Smart Discard Logic: dirty-state tracking, unsaved changes dialog on close, and explicit save/discard transitions (`f92d23b`).
- Fixed backend-related file-save bugs that caused missing theme data or corrupt metadata after reload (`b7fbba0`).

6.5.3 Why This Mattered

These improvements made the desktop app production-ready: stable theme, safe save behavior, and reliable export are mandatory features for engineering applications where data integrity is critical.

Chapter 7

Features Implemented

7.1 Authentication UI and Local Database

Implemented both login and create-account screens with input validation and local SQLite table creation (`3feb069`). Added toast notification system for immediate feedback and cross-platform UI tweaks for screen consistency (`4367f47`, `a6201bc`).

7.2 Canvas Subsystem

Implemented `canvas_screen.py` with event processing and a rendering pipeline (Event Handler → Component Update → Render Engine → Canvas Refresh). Implemented keyboard shortcuts and operations to increase editor productivity.

7.3 Component Library and Component Widget

Built `component_library.py` for symbol metadata management and JSON-based local asset loading. Implemented `component_widget.py` with interactive grips, connection anchor points, and low-level painting logic.

7.4 Grip Editor with Live Preview

Implemented Grip Editor UI for editing component grips and anchor points, with live preview so that changes reflect on the canvas immediately (`3a9ed58`).

7.5 Centralized Theme Management

Implemented `theme.py` to apply QSS and theme properties consistently across all screens; toggling theme updates all pages via the global app state (`1f7f74d`).

7.6 Save Workflow and Smart Discard

Implemented manual save, dirty-state tracking, unsaved changes dialog, and discard behavior (`f92d23b`). Integrated the save UI with the API client to synchronize with backend project endpoints.

7.7 High-quality Export Improvements

Added image/PDF export improvements to increase output resolution and ensure coordinate fidelity during export (`c1897a2`).

Chapter 8

Application Screenshots

8.1 Login Screen

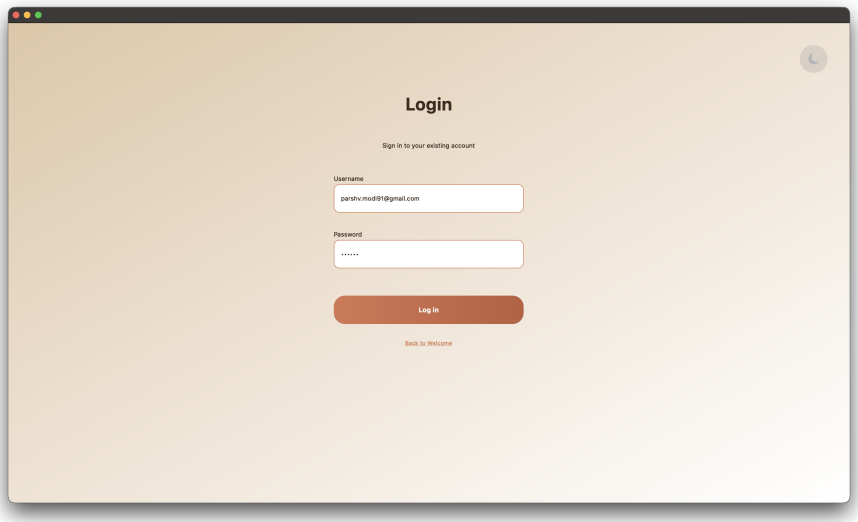

Figure 8.1: Login Screen — Authentication UI with input validation and toast notifications.

8.2 Canvas with Components

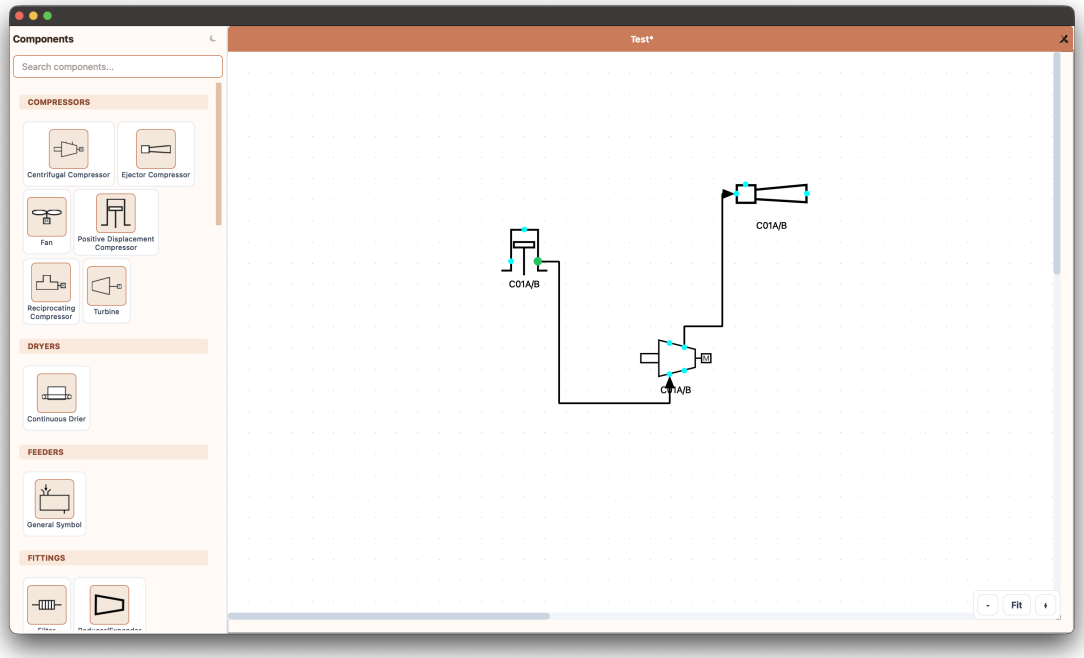

Figure 8.2: Canvas Screen — Components placed and connected on the interactive PFD canvas.

8.3 Grip Editor Dialog

The Grip Editor allows users to interactively reposition connection anchor points (grips) on a component. The Live Preview panel updates in real time as grips are dragged, demonstrating the coordinate transformation system implemented to fix grip misalignment bugs (commits `43cf790`, `7987c64`, `571805d`).

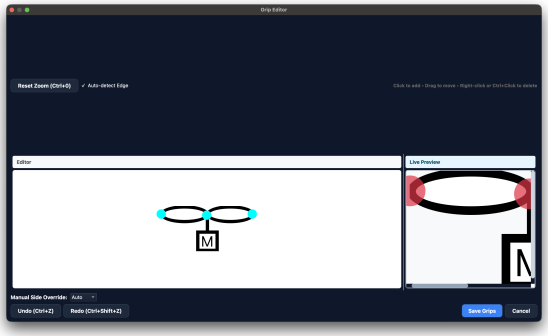

Figure 8.3: Grip Editor — default grip positions; live preview shows grips at component edges.

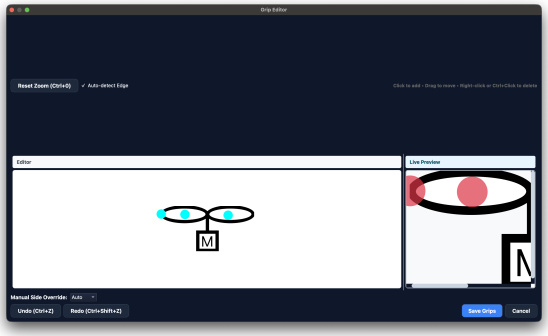

Figure 8.4: Grip Editor — after dragging grips inward; live preview reflects updated anchor positions immediately.

8.4 Light and Dark Theme

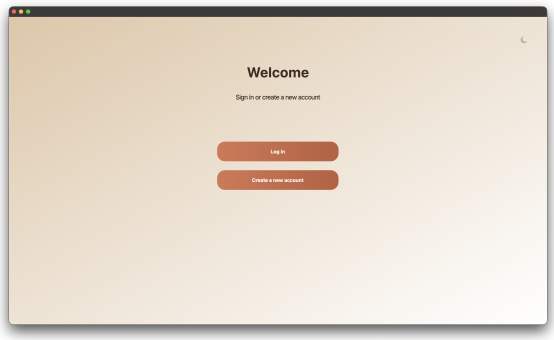

Figure 8.5: Light Theme.

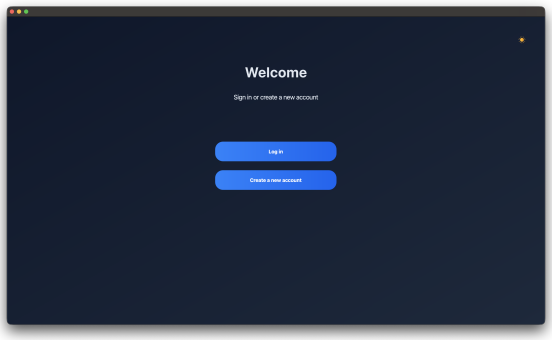

Figure 8.6: Dark Theme.

Chapter 9

Bugs Fixed & Technical Improvements

9.1 Incorrect Grip Positions

Symptoms: Connection lines and anchors appeared offset after certain transformations (zoom, pan, widget resize).

Root cause: Inconsistent coordinate space conversions between component-local coordinates and canvas/world coordinates.

Fix: Revised coordinate transformation functions and anchor offset handling in `component_widget.py` and `canvas_screen.py`, added tests/edge-case handling for zero-size components. (Commits: `43cf790`, `7987c64`, `571805d`).

9.2 Qt Runtime Errors

Symptoms: Occasional crashes relating to widget painting or destroyed objects during screen transitions.

Root cause: QPropertyAnimation objects being garbage-collected, or theme application being invoked before UI widgets were initialized.

Fix: Persist animations on the stacked widget (stored as `widget._anims`), guard theme application when global widget is not initialized, and apply unpolish/polish style property patterns. (Commits: `59c6f9d`, `412f9e2`).

9.3 Backend Save File Mismatch

Symptoms: Files saved to backend were corrupted or had missing theme data after reload.

Root cause: Serialization missing theme and grip metadata, and edge-case handling for export IO.

Fix: Ensured export pipeline includes component grip metadata and theme flags in the saved JSON; fixed API client calls to correctly upload files with metadata. (Commit: `b7fbba0`).

9.4 Cross-platform UI Inconsistencies

Symptoms: Controls and fonts rendered differently across OSes and some interactions were misaligned.

Fix: Adjusted UI templates and style sheets (.ui and QSS), introduced defensive sizing logic in `main.py`. (Commit: `a6201bc`).

Chapter 10

Code Evolution & Commit Analysis

10.1 Development Timeline

Table 10.1: Development Timeline (Selected Commits)

Date / Commit	Type	Notes
Dec 9, 2025 — `3feb069`	Feature	Login/signup screens, SQLite creation, app scaffolding. (∼1626 insertions)
Dec 9, 2025 — `4367f47`	Enhancement	UI updates; fullscreen mode.
Dec 10, 2025 — `a6201bc`	Bugfix	Cross-platform UI compatibility.
Dec 11, 2025 — `847068a`	WIP	API client changes and UI sync preparation.
Dec 13, 2025 — `2617669`	Feature	Canvas integration, new component library. (∼363 insertions)
Dec 17, 2025 — `c230c2e`	Feature	Finalized landing page, grips.json, styles.qss. (∼1399 insertions)
Dec 23, 2025 — `5462167`	Feature	Symbol Dialog and API client updates.
Late Dec — `7e596b9`	Feature	Grip Editor and updated Symbols dialog.
Late Dec — `3a9ed58`	Feature	Live preview in Grip Editor.
Late Dec — `43cf790`, `7987c64`, `571805d`	Bugfix	Multiple coordinate transformation fixes.
Jan — `1f7f74d`	Enhancement	Centralized theme management.
Jan — `c1897a2`	Enhancement	High-quality export improvements.
Jan — `f92d23b`	Feature	Recent Projects, Manual Save Workflow, Smart Discard Logic.
Jan — `b7fbba0`	Bugfix	Theme issues and backend file save bug.

10.2 Feature Contribution Summary

Table 10.2: Feature Contributions by Area

Feature	Contribution Summary
Authentication UI	Login/signup UI, local SQLite helper, validation, toast feedback. (`3feb069`)
Canvas Subsystem	Canvas screen with render pipeline and event handling; component widget creation. (`2617669`)
Component Library	Local asset management and component metadata loader. (`ui/assets/grips.json`)
Grip Editor	Interactive grip editing with live preview and coordinate mapping. (`7e596b9`, `3a9ed58`)
Theme Management	Centralized theme application and toggling across screens. (`1f7f74d`)
Save Workflow	Manual save, dirty-state flag, Recent Projects, smart discard. (`f92d23b`)
Export Pipeline	Improved local export quality and backend export API integration. (`c1897a2`)

10.3 Bug Fix Summary

Table 10.3: Major Bugs Fixed

Symptom	Fix Implemented	Commit(s)
Grip misalignment	Revised coordinate transforms and anchor offset math	`43cf790`, `7987c64`, `571805d`
Qt runtime errors	Persist animations; guard theme application; manage widget lifecycles	`59c6f9d`, `412f9e2`
Backend save mismatch	Added metadata to save payload and corrected API client upload logic	`b7fbba0`
Cross-platform UI issues	Tweaked UI sizes and QSS; added defensive sizing	`a6201bc`

10.4 Commit Classification

Table 10.4: Classification of Selected Commits

Commit	Class	Notes
`3feb069`	Feature	Authentication screens, SQLite DB creation, UI scaffolding.
`2617669`	Feature	Canvas integration and component library.
`c230c2e`	Enhancement	Landing page finalization; styles and assets.
`5462167`	Feature	Symbol dialog and API integration.
`7e596b9`	Feature	Grip Editor.
`43cf790`	Bugfix	Grip coordinate fixes.
`1f7f74d`	Enhancement	Theme centralization.
`f92d23b`	Feature	Recent Projects and save/discard workflow.
`c1897a2`	Enhancement	Export improvements.
`b7fbba0`	Bugfix	Backend save and theme fix.

10.5 Contribution Summary Table

Table 10.5: Overall Contribution Counts

Contribution Type	Approximate Count
Major Feature Implementations	5
Bug Fixes	10+
UI Enhancements	8+
Workflow Improvements	5+
Merge Conflict Resolutions	Multiple

10.6 Personal Code Growth (Insertions / Deletions)

Selected examples from commit diffs:

- `3feb069` — 12 files changed, 1626 insertions (initial UI + auth)
- `4367f47` — 4 files changed, 91 insertions, 320 deletions (UI adjustments)
- `847068a` — ∼315 insertions, 47 deletions (API client / local UI changes)
- `2617669` — 363 insertions, 23 deletions (canvas integration)
- `c230c2e` — 1399 insertions, 53 deletions (landing page, styles, assets)
- `412f9e2` — 92 insertions, 3 deletions (runtime fix)

Chapter 11

Challenges Faced

The major technical and process challenges encountered included:

- **Merge Conflicts**: Integrating large UI and component library changes required careful conflict resolution. Frequent rebasing, diff inspection, and reconciliation of divergent implementations were necessary to preserve functionality.
- **Coordinate Math for Grips and Connectors**: Ensuring precise mapping between component-local coordinates and canvas/world coordinates required careful handling of transforms, especially when zoom/scale was involved. I developed the Coordinate Transformation System and validated changes with live previews.
- **Qt Lifecycle and Garbage Collection Issues**: PyQt's C++/Python object lifetime behavior caused intermittent runtime errors when animations or toasts were created without preserving references. Fixed by persisting animation references and cleaning up only after completion.
- **Managing Complex PyQt Widget Hierarchies**: Maintaining clean parent-child widget relationships while supporting dynamic content updates required careful architectural decisions.
- **Backend Integration Edge-Cases**: Working with asynchronous API expectations and serialisation constraints required iterations to ensure metadata (grip coords, theme) was preserved in saved project payloads.
- **Ensuring Cross-Platform Consistency**: UI sizing, fonts, and scaling behave differently on different OSes; resolving these required testing on multiple platforms and defensive sizing logic.

Chapter 12

Conclusion & Learning Outcomes

12.1 Summary

During the two-month part-time internship, I personally implemented the majority of the desktop frontend user-facing functionality described in this report: authentication UI, canvas subsystem, component library, grip editor, theme management, export pipeline, and save/discard workflow. I resolved multiple critical bugs, improved runtime stability, and integrated the desktop client with backend APIs where required. Under the mentorship of Mr. Chirag Koyande, I improved both technical depth and problem-solving approach. The project strengthened my understanding of frontend architecture, state management, event-driven systems, and software quality practices in a collaborative team environment.

12.2 Technical Learning Outcomes

- Practical experience with event-driven GUIs and PyQt5 rendering
- Designing robust coordinate transformation systems for vector graphics
- Managing state across a multi-screen GUI using QStackedWidget and global app state
- Implementing safe save workflows and export pipelines for engineering diagrams
- Techniques for resolving merge conflicts and maintaining code quality in a collaborative repository

12.3 Recommended Next Steps

- Expand unit and integration tests (pytest-qt): canvas event handling, grip updates, API integration
- Implement persistence of undo/redo stack across saves
- Benchmark export pipeline for very large diagrams and optimize rendering path
- Add CI checks for UI resource validity and linting for Python modules

Bibliography

[1] Project Description and Team Roles (Chemical PFD Builder) — internal project document.

[2] Commit logs, commit diffs and detailed commit history (commit_details.txt, commit_diffs.txt, commits.txt).

[3] Official report format PDF used to structure this document.

Appendix A

Complete Commit References

All commits authored by Parshv Keyur Modi that directly reflect the work documented in this report:

- `3feb069` — Login and Signup Mechanism (multiple UI files + Python modules)
- `4367f47` — UI Updates and Fullscreen Mode
- `a6201bc` — Cross-Platform UI Compatibility Fix
- `847068a` — API client changes and UI sync (intermediate commit)
- `2617669` — Canvas Integration: updated component library, screens, removed db.py
- `c230c2e` — Finalize feature: modified landing page, canvas updates, and styles
- `5462167` — Implement Symbol Dialog and update API client
- `b93e0d4` — Update component library
- `e411d99` — Added shortcuts on canvas screen
- `7e596b9` — Add Grip Editor and updated Symbols dialog
- `161f1e4` — Update Grip Editor and Component Widget
- `3a9ed58` — Added live preview in grip editor
- `43cf790` — Fixed incorrect grip positions
- `7987c64` — Further grip corrections
- `571805d` — Final grip alignment fixes
- `1f7f74d` — Centralised theme and minor fixes
- `59c6f9d` — Solved Qt error
- `412f9e2` — Runtime error fix

- `c1897a2` — High quality export and bug fixes
- `f92d23b` — Implement Recent Projects, Manual Save Workflow, and Smart Discard Logic
- `b7fbba0` — Fixed theme issues and backend file save bug
- `4cffcda` — Added pandas in requirements.txt
- `16c8a24` — Added openpyxl in requirements.txt