FOSSEE Semester Long Internship Report

On

Member Property Refactoring and Design, Details Output State Management for OsdagBridge

Submitted by

Sreejesh S

3rd Year BE Student, Department of CSE With CyberSecurity

Sri Eshwar College of Engineering

Coimbatore

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Nidhi Khare

Aditya Donde

June 11, 2026

Acknowledgments

I sincerely thank everyone who supported and guided me throughout my FOSSEE Winter Internship, which made this project a valuable and enriching learning experience. I am deeply grateful to my mentors, Nidhi Khare and Aditya Donde, for their continuous guidance, technical insights, and constant encouragement during the OsdagBridge refactoring and CAD visualization work. I also thank the Osdag project team, especially Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia, for fostering a supportive and collaborative work environment.

I respectfully acknowledge Prof. Siddhartha Ghosh, Principal Investigator of Osdag, Department of Civil Engineering, IIT Bombay, for providing me with the opportunity to contribute to open-source engineering development. I also express my sincere gratitude to Prof. Kannan M. Moudgalya, Principal Investigator of the FOSSEE project, Department of Chemical Engineering, IIT Bombay, for his leadership and vision in facilitating this internship program. I extend my appreciation to the FOSSEE Managers, Usha Viswanathan and Vineeta Parmar, and their team for their efficient coordination and consistent support.

I gratefully acknowledge the support of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education, Government of India. I also thank my fellow interns and colleagues for their cooperation and constructive discussions throughout the internship. Finally, I express my sincere gratitude to the authorities of Sri Eshwar College of Engineering, including my Department Head and Principal, for their encouragement and academic support.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

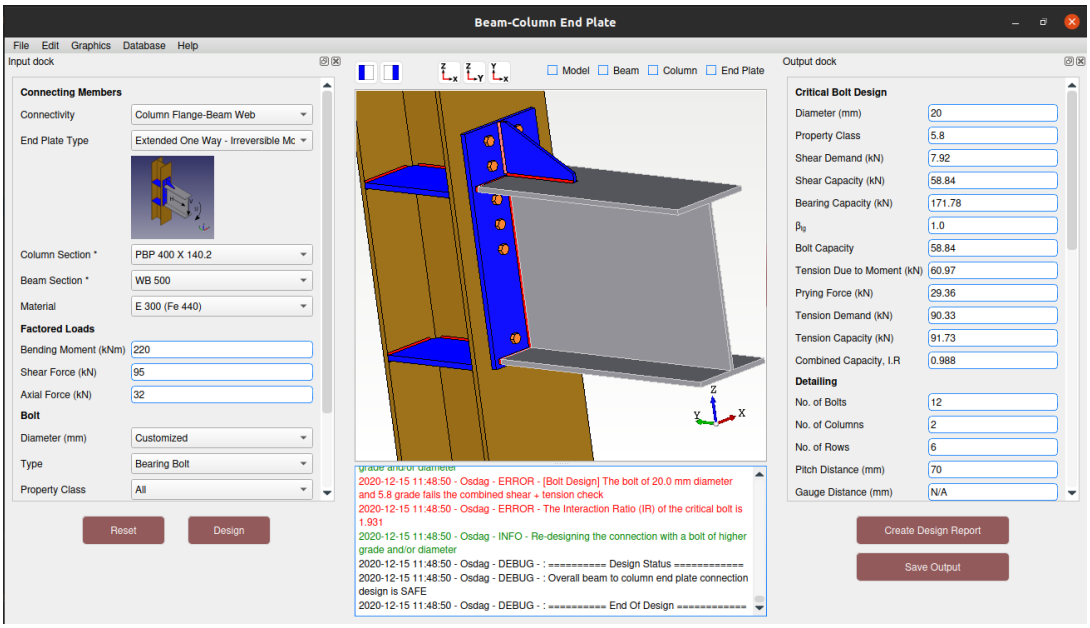


Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The screening task was to implement the Group Design module as a standalone web application with a React frontend and a Django REST backend. The scope covered a full Basic Inputs workflow, a spreadsheet-style popup for custom loading parameters, accurate validation rules for geometry, materials, and loads, and an API layer that supports location-based lookup and server-side validation. The UI had to follow the Osdag-web color theme, use reusable form components, and keep a static bridge cross-section visualization on the right panel.

2.2 Tasks Done

I implemented the end-to-end screening task based on the provided requirements, focusing on both the frontend behavior and backend endpoints. The work included:

- Built the two-panel layout with left-side tabs (Basic Inputs, Additional Inputs placeholder) and a non-interactive right-side bridge cross-section image.
- Implemented the Type of Structure dropdown with the "Other structures not included" message and hard-disable of all remaining inputs.
- Created two mutually exclusive Project Location modes: (A) Location Name mode with State and District dropdowns plus auto-filled wind, seismic, and temperature

values; (B) Custom Loading mode with a spreadsheet-style popup for wind, seismic, and temperature inputs.

- Ensured all auto-filled values are displayed in green and that mode toggles correctly disable the other mode.
- Implemented validation rules for span, carriageway width, footpath, and skew angle warning as specified by IRC 24 (2010).
- Added material selection inputs for girder steel, cross bracing steel, and deck concrete.
- Implemented the Modify Additional Geometry popup with synchronized fields (girder spacing, number of girders, and deck overhang) and correct auto-adjustment logic based on overall width.
- Added backend API endpoints for location lookup, custom loading, materials, and geometry validation with normalized responses.

2.3 Implementation Highlights

2.3.1 Frontend

The React UI uses reusable components (Dropdown, Input, FormSection, PopupModal) to keep behavior consistent across tabs. I implemented state-driven UI for the two location modes, summary fields, and the geometry popup. Validation errors are displayed inline, and warnings (such as skew angle beyond $\pm 15^{\circ}$) are shown without blocking input.

2.3.2 Backend

The Django REST backend exposes endpoints for location lookups, custom loading inputs, materials list, and geometry validation. The geometry validation endpoint returns both error states and auto-adjusted values for the popup so the UI remains consistent. A CSV/JSON data source is supported for location-based inputs, with a fallback to hardcoded values when the dataset is unavailable.

2.4 Validation and Rules Implemented

- Span outside 20–45 m returns "Outside the software range."
- Carriageway width must be ≥ 4.25 m and < 24 m.
- Skew angle outside $\pm 15^\circ$ shows a warning requiring detailed analysis per IRC 24 (2010).
- Overall width = carriageway width +5 m.
- Overhang and spacing are each $<$ overall width.
- (Overall Width − Overhang)/Spacing = No. of Girders with auto-adjustment on any field change.

2.5 Outcome

The screening task resulted in a fully working standalone module with a clean, Osdag-themed UI, complete validations, and a functional API layer. The spreadsheet popup and auto-adjustment logic were fully integrated, and the Basic Inputs flow supports both location-based and custom loading data with consistent feedback to the user.

2.6 My Contributions (Repository Work Summary)

In addition to implementing the functional requirements, I made incremental improvements through multiple commits focused on UI consistency, validations, state management, and backend support.

2.6.1 Osdag-Screening-Task (main branch)

The complete source code and commit history for this task can be accessed at: github. Based on the commit history, my contributions in this repository include:

- Implemented core bridge visualization components (2D cross-section view) and integrated the view into the application layout.

- Added bridge view modes along with validation indicators/highlights and improved schematic dimensions for better clarity.
- Refactored UI layout and form controls for improved responsiveness and consistent branding.
- Integrated enhanced dropdown behavior (including disabling search where needed) and improved accessibility/styling.
- Implemented geometry validation and adjustment utilities and enhanced the geometry popup with equation display.
- Added backend APIs to support location lookup and improved reference data handling.
- Added report-generation capability using jsPDF and improved error handling around the bridge view.
- Updated database usage by migrating from SQLite to PostgreSQL and updating backend requirements (including psycopg).
- Improved summary card styling and dropdown menu behavior for better overall UX.

Chapter 3

Refactoring and Enhancement of Member Properties

3.1 Task 1: Problem Statement

The existing Member Properties system in OsdagBridge faced several limitations that hindered the structural design workflow:

- **Inconsistent Design-Type Behaviour**: The Member Properties tab lacked awareness of the active design type (Optimized vs. Custom), meaning the same UI was presented regardless of context, exposing irrelevant controls and confusing the user.
- **Irrelevant Fields in Sub-Tabs**: The Cross Bracing and End Diaphragm tabs surfaced fields that were not applicable to their respective member types, cluttering the interface and increasing the risk of erroneous input.
- **Welded-Type Inconsistency**: The welded configuration available in End Diaphragm did not mirror the options in Girder Properties, causing mismatches between Optimized and Custom modes.
- **Fragmented UI Terminology and Layout**: Labels such as "Customized" were used inconsistently across tabs, and the overall section-properties layout varied between the Girder, Stiffener, Cross-Bracing, and End Diaphragm dialogs, reducing learnability and professional finish.

- **Legacy Structural Coupling**: The `GirderDetailsTab` was tightly coupled to a now-redundant `schema_builder.py` module, and Girder and Stiffener components contained duplicated logic that resisted reuse and future maintenance.

3.2 Task 1: Tasks Done

To address these issues, a comprehensive refactoring of the Member Properties system was carried out across the UI layout, dialog interaction, and underlying component architecture:

- **Full Layout Refactor of Member Properties Tab**: The Member Properties tab was redesigned from the ground up, reorganising controls for improved clarity, visual hierarchy, and alignment with the rest of the OsdagBridge interface.
- **Design-Type–Based Behaviour**: Implemented conditional logic so that the Member Properties tab responds dynamically to the selected design type. Fields, controls, and validation rules adapt automatically when the user switches between Optimized and Custom modes, eliminating irrelevant options from the active view.
- **Removal of Irrelevant Fields**: Audited and pruned fields from the Cross Bracing and End Diaphragm tabs that did not apply to those member types, reducing cognitive load and the potential for invalid input.
- **Welded-Type Consistency**: Aligned the welded-section options in the End Diaphragm tab with those in Girder Properties, ensuring that both Optimized and Custom modes present a consistent set of welded-type choices across the project.
- **Section-Properties UI Unification (PR #48 – Merged Apr 18)**: Unified and polished the section-properties interface across all four tabs—Girder, Stiffener, Cross-Bracing, and End Diaphragm—delivering a coherent visual language throughout the dialog system.
- **Improved Dialog Interaction Flow**: Refined the interaction model for section-property dialogs, including updated preview and placeholder viewer wiring and tighter integration with the surrounding tab structure.

- **Generalisation of Girder and Stiffener Components**: Extracted shared logic from the Girder and Stiffener implementations into reusable base components, eliminating duplication and simplifying future extension to additional member types.
- **Removal of `schema_builder.py`**: Identified and deleted the unused `schema_builder.py` module following the `GirderDetailsTab` refactor, reducing dead code and narrowing the maintenance surface.
- **Terminology Standardisation**: Replaced all instances of "Customized" with "Custom" across every affected component, aligning the UI language with the established OsdagBridge vocabulary.
- **Default Member Length Update**: Updated the default member length from `30.0 m` to `25.0 m` to better reflect common bridge span configurations.
- **Enhanced Thickness Value Retrieval**: Extended the thickness lookup logic to include `SAIL_APPROVED_THICKNESS_VALUES` as a fallback source, improving robustness for non-standard section inputs.
- **Segment Action Widget Improvements**: Redesigned the add/remove segment action widget with icon integration, introduced an icon-caching mechanism to optimise repeated rendering, and adjusted field-width settings for improved UI consistency.
- **Rename for Clarity**: Renamed the internal helper `_default_dimension_bounds` to `_default_dimension_bounds_for_field` to more precisely communicate its scope and prevent future misuse.

Scale of Change (PR #48): `+4,096` additions and `-1,224` deletions across **12 files**, representing a substantial structural overhaul of the Member Properties subsystem.

[t]0.48

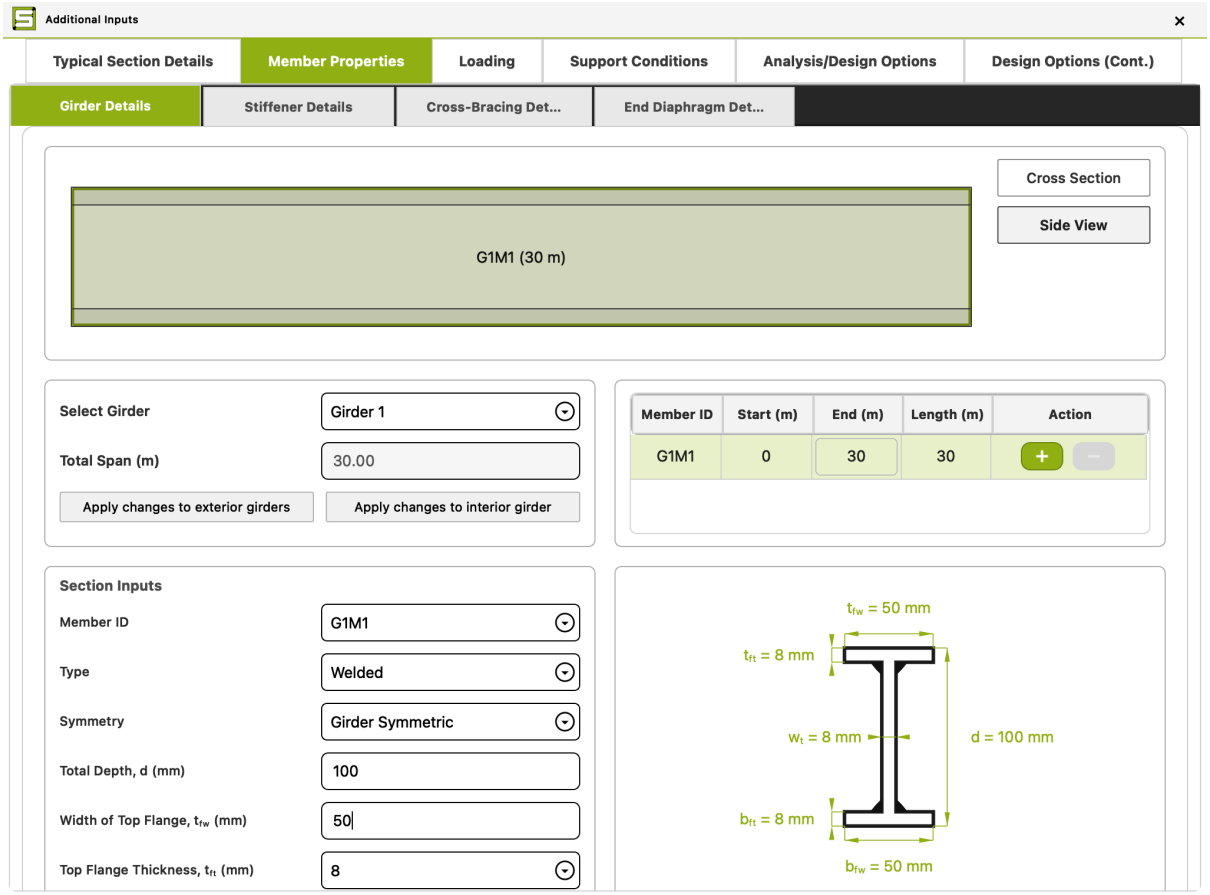


Figure 3.1: Girder tab — Custom mode

[t]0.48

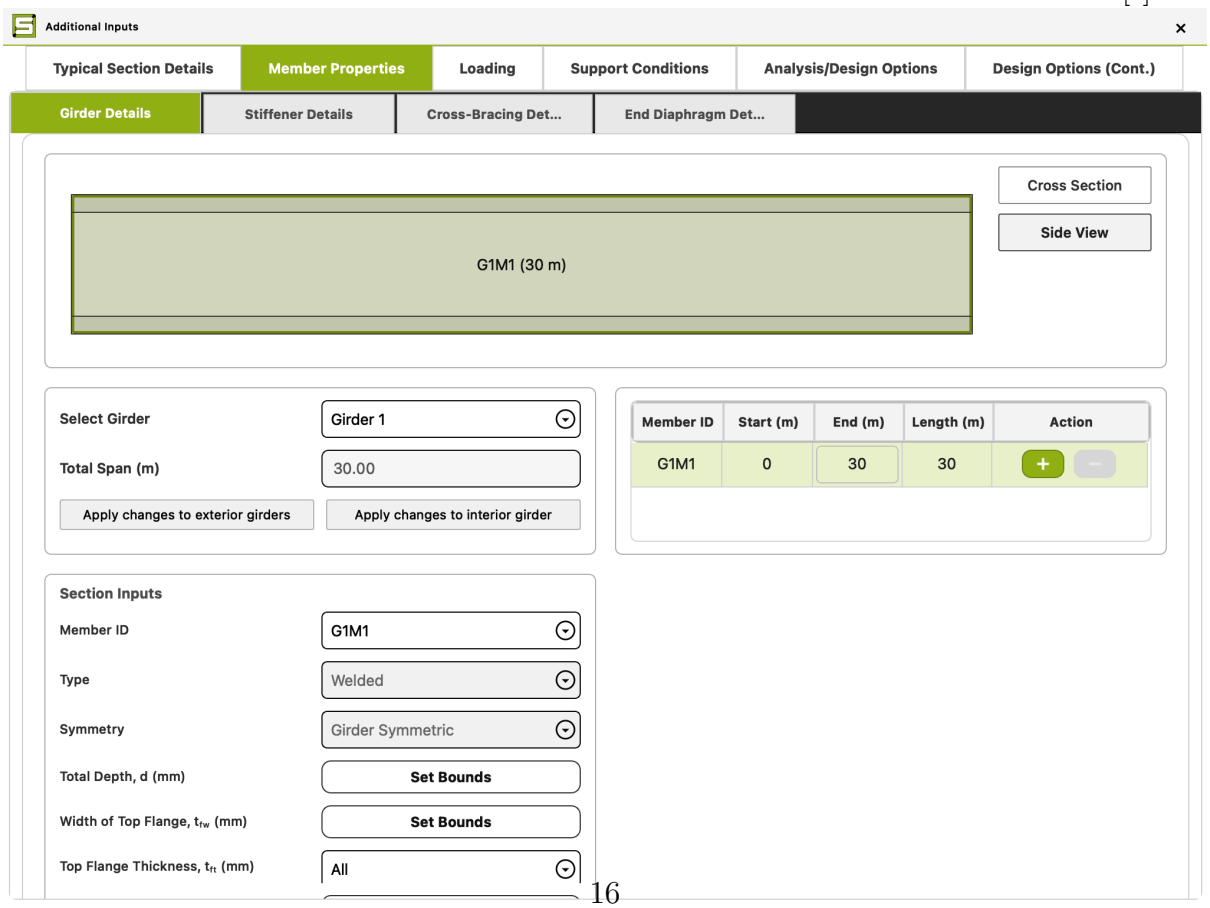


Figure 3.2: Girder tab — Optimized mode

[t]0.48

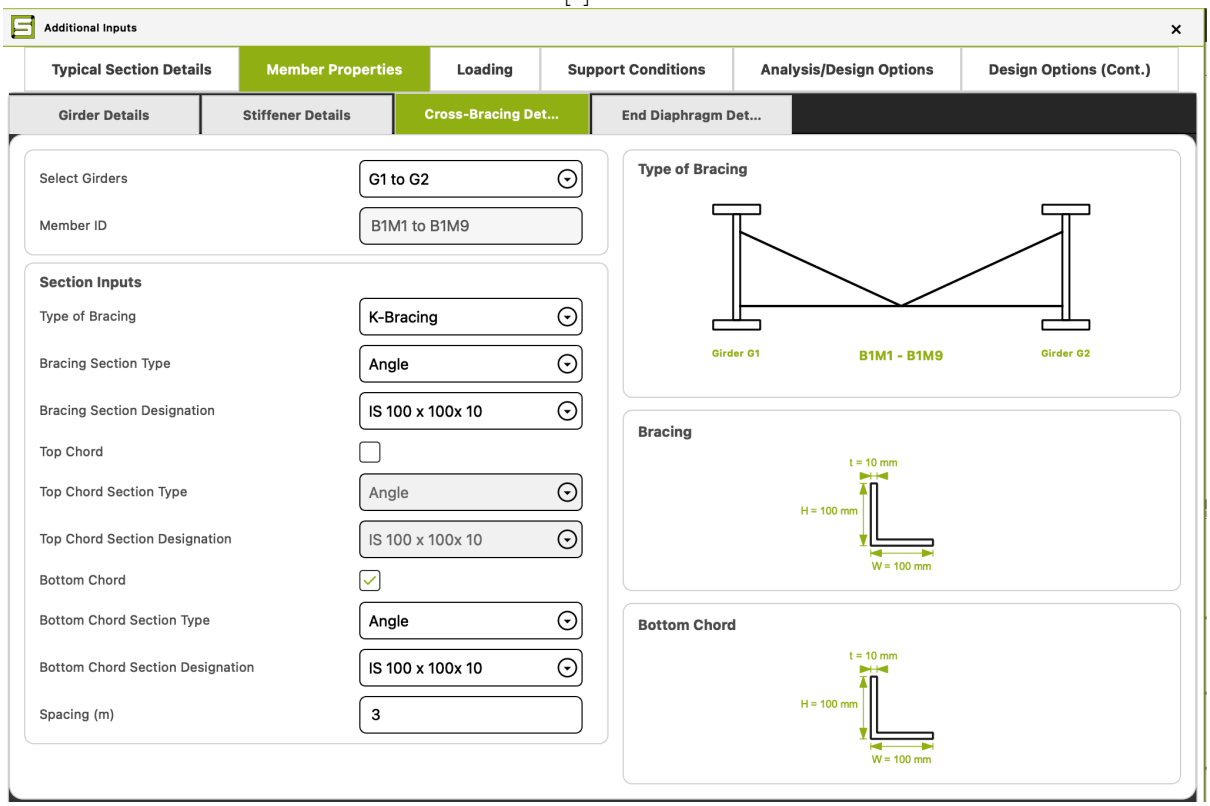


Figure 3.4: Cross Bracing tab — Custom mode

[t]0.48

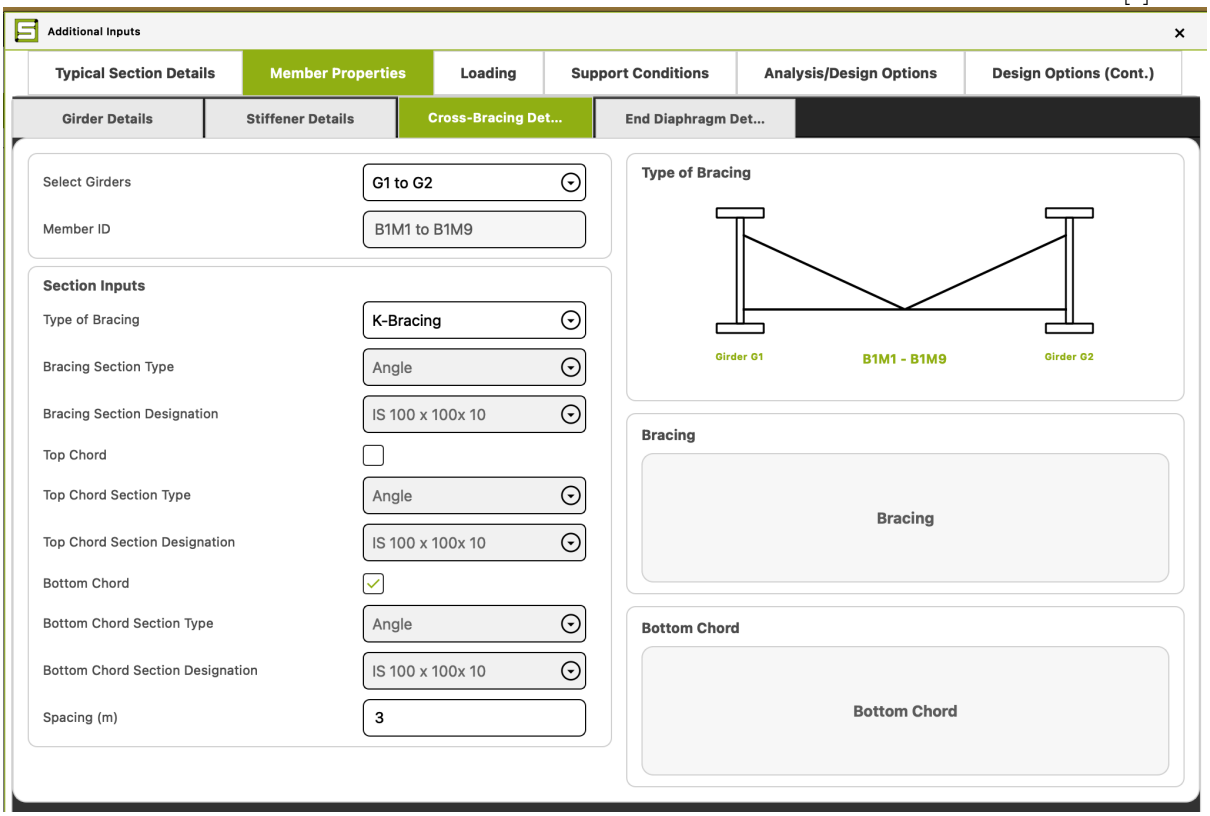


Figure 3.5: Cross Bracing tab — Optimized mode

Figure 3.6: Pruned field set in the Cross Bracing tab. Irrelevant fields present in the original tab have been removed; the remaining controls are further filtered by design type so that only context-appropriate inputs are visible in each mode.

3.3 Task 1: Python Code & Logic

This section details the key implementation patterns introduced during the Member Properties refactor.

3.3.1 Description of the Script

Two core engineering functionalities underpin the refactored system:

- **Design-Type–Aware Field Visibility**: A gating mechanism inspects the current design-type selection and shows or hides UI controls accordingly, keeping the interface minimal and context-relevant at all times.
- **Thickness Fallback Chain**: An enhanced retrieval function queries the primary section catalogue first and falls back to `SAIL_APPROVED_THICKNESS_VALUES` when the primary source does not cover the requested dimension, ensuring valid defaults for all input combinations.

3.3.2 Python Code: Design-Type Gating & Thickness Retrieval

Listing 3.1: Design-type gating and thickness fallback logic in OsdagBridge

```
# --- Design-Type-Based Field Visibility ---
def _apply_design_type_visibility(self, design_type: str) -> None:
    """Show or hide controls based on the active design type."""
    is_custom = (design_type == "Custom")

    # Controls relevant only to Custom mode
    for widget in self._custom_only_widgets:
        widget.setVisible(is_custom)

    # Controls relevant only to Optimized mode
    for widget in self._optimized_only_widgets:
        widget.setVisible(not is_custom)

    # Refresh layout to prevent stale geometry artefacts
    self.layout().activate()

```

```
17
18  # --- Enhanced Thickness Value Retrieval ---
19  def _get_thickness_values(
20      self,
21      field_key: str,
22      section_type: str
23  ) -> list[float]:
24      """Return approved thickness values with SAIL fallback."""
25      primary = self._section_catalogue.get(field_key, section_type)
26      if primary:
27          return primary
28
29      # Fallback: SAIL approved plate thickness series
30      return SAIL_APPROVED_THICKNESS_VALUES.get(field_key, [])
31
32
33  # --- Terminology Standardisation Helper ---
34  def _normalise_mode_label(self, raw_label: str) -> str:
35      """Replace legacy 'Customized' with canonical 'Custom'."""
36      return raw_label.replace("Customized", "Custom")
```

3.3.3 Explanation of the Code

- **Lines 2–13**: `_apply_design_type_visibility` iterates over two pre-registered widget lists—one for Custom-only controls, one for Optimized-only—and toggles their visibility in a single pass. Calling `layout().activate()` prevents ghost geometry from persisting after widgets are hidden.

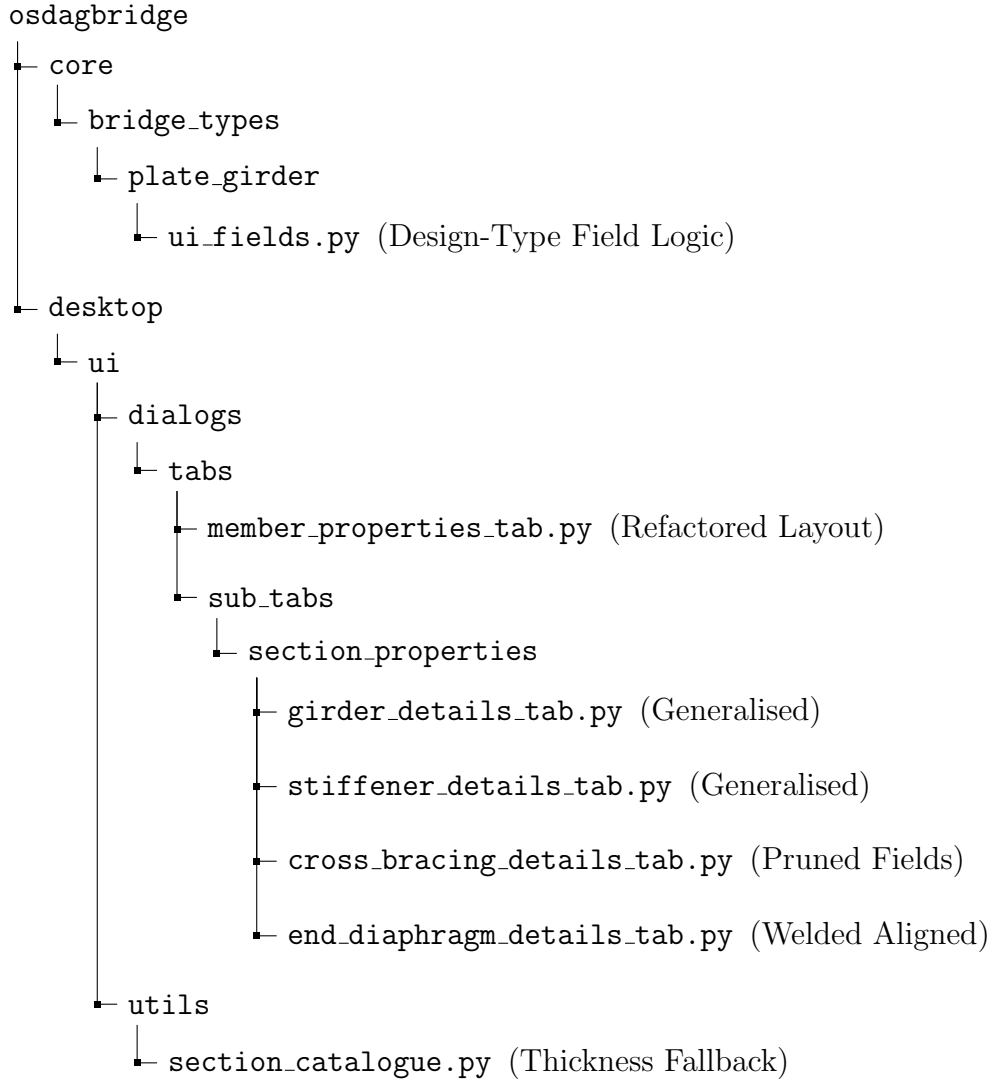
- **Lines 16–27**: `_get_thickness_values` implements a two-stage lookup. The primary catalogue is queried first; only on a cache miss does the function fall back to the `SAIL_APPROVED_THICKNESS_VALUES` dictionary. This keeps the preferred data source authoritative while guaranteeing a valid return value for edge-case inputs.

- **Lines 30–33**: `_normalise_mode_label` provides a single point of control for the "Customized" → "Custom" terminology migration, ensuring that any label arriving from legacy code or serialised state is corrected before it is displayed.

3.4 Task 1: Documentation

The following directory structure illustrates the modular layout of the refactored Member Properties subsystem, highlighting the separation between core logic, UI tabs, and sub-tab components.

3.4.1 Directory Structure



Implementation Note: The design-type gating mechanism is registered at tab-initialisation time, so no additional event wiring is required when new controls are added to either the Custom or Optimized widget lists. The SAIL thickness fallback is confined to `section_catalogue.py`, keeping thickness policy logic out of individual tab files and making future catalogue updates a single-file change.

Chapter 4

Design Tab, Details Tab, and Output State Management

4.1 Task 2: Problem Statement

The initial architecture of OsdagBridge's design result pipeline suffered from several structural and usability limitations:

- **No Unified Output State**: After a design run, result data was scattered across the CAD state, grillage datasets, and ad-hoc local variables. There was no single, structured container that downstream components—the Details tab, output dock, and report generator—could reliably query.
- **Details Tab Coupled to CAD State**: The Details tab loaded its display values directly from the CAD state object. This created a fragile dependency; any change to the CAD rendering pipeline risked silently corrupting the data shown in the Details view.
- **Absent Backend Design Queries**: The backend exposed no standardised interface for retrieving the post-design steel state or utilization ratios (DCR checks), forcing the UI layer to reach into internal solver structures directly.
- **Deflection Plot Errors**: The deflection curve was rendered in the structurally incorrect direction (positive upward), conflicting with the sign convention used in the 3D grillage view and standard structural engineering practice.

- **Dialog Premature Access & ID Mismatches**: Dialogs could be opened before a valid design existed, leading to runtime errors. A mismatch between the design-check tab identifier and the active member ID caused checks to be applied to the wrong member silently.
- **Inconsistent CAD Layout**: The Details tab CAD window and section properties panel were not positioned consistently with the rest of the OsdagBridge interface (e.g., the Girder UI), reducing the visual coherence of the application.
- **No Report Generation Infrastructure**: There was no base framework for producing engineering reports from design results, leaving report generation entirely unimplemented.

4.2 Task 2: Tasks Done

To resolve these issues, a full implementation of the output state pipeline, Details tab fixes, UI layout alignment, and report generation base code was carried out across PRs #69, #74, and #80:

- **Backend Design Query Interface (PR #74)**: Added two new methods to the backend—`get_steel_design_state()` and `get_output_state()`—providing a stable, version-controlled API through which any UI component can retrieve post-design structural data without accessing internal solver objects directly.
- **Steel Design State Retrieval in Template Page**: Implemented steel design state retrieval and synchronisation logic in the template page, ensuring that the current design configuration is always reflected accurately in the UI after each solver run.
- **Placeholder DCR Checks**: Added placeholder Demand-Capacity Ratio (DCR) checks across all code-check categories (flexure, shear, lateral torsional buckling, fatigue, and interaction), establishing the full check skeleton for future solver integration.
- **Output Dictionary Construction (PR #80)**: Built the `output_dict` immediately after each design run. The dictionary consolidates all result data into a single

structured container comprising:

  - All user input parameters (`input_dict` fields)
  - Steel design state: geometry, materials, stiffener details, and section properties from `get_steel_design_state()`
  - Analysis and design utilization outputs: flexure, shear, LTB, fatigue, and interaction check ratios from `get_output_state()`
  - Output dock UI state: active combobox selections, checkbox states, and radio button choices
  - Deflection values extracted from the grillage results `xarray` Dataset

- **Details Tab Migration to `output_dict`**: Decoupled the Details tab from the CAD state by rewriting all data-loading calls to source values from `output_dict`. This makes the Details tab robust to CAD rendering changes and ensures it always reflects the latest solved design.

- **Design Check Tab ID Mismatch Fix**: Identified and resolved a bug where the design-check tab used a stale member identifier that did not match the `active_member_id`, causing checks to be silently displayed for the wrong member.

- **Deflection Plot Direction Fix**: Corrected the deflection curve rendering by multiplying all deflection values by -1 before plotting, bringing the direction into alignment with the sign convention used in the 3D grillage view (negative downward).

- **Deflection Plot Colour**: Applied the project-specified deep purple (`#6A1B9A`) as the deflection plot line colour to distinguish it clearly from other result curves.

- **Project Location Bug Fix**: Fixed a bug where the project location field was incorrectly reset or overwritten when the user switched design types, preserving the user's prior entry across mode changes.

- **Output State Loading in Dialogs & `active_member_id` Tracking**: Refactored the dialog initialisation sequence to load the correct output state on open and to bind all dialogs to the current `active_member_id`, eliminating stale data displays.

- **Dialog Guard Conditions**: Added pre-access guards to all result-dependent dialogs so that they refuse to open (with an informative message) when no valid design result exists, preventing runtime errors from uninitialised `output_dict` access.
- **CAD Layout Alignment (PR #69)**: Repositioned the CAD window to the right side of the Details tab (above the Section Properties panel), mirroring the established Girder UI layout for visual consistency. Aligned CAD previews with their descriptive captions and added shared stiffener table constants used across multiple tabs.
- **Report Generation Base Code**: Implemented the foundational infrastructure for engineering report generation, establishing the document structure, section scaffolding, and `output_dict` data-binding layer that future report content will build upon.

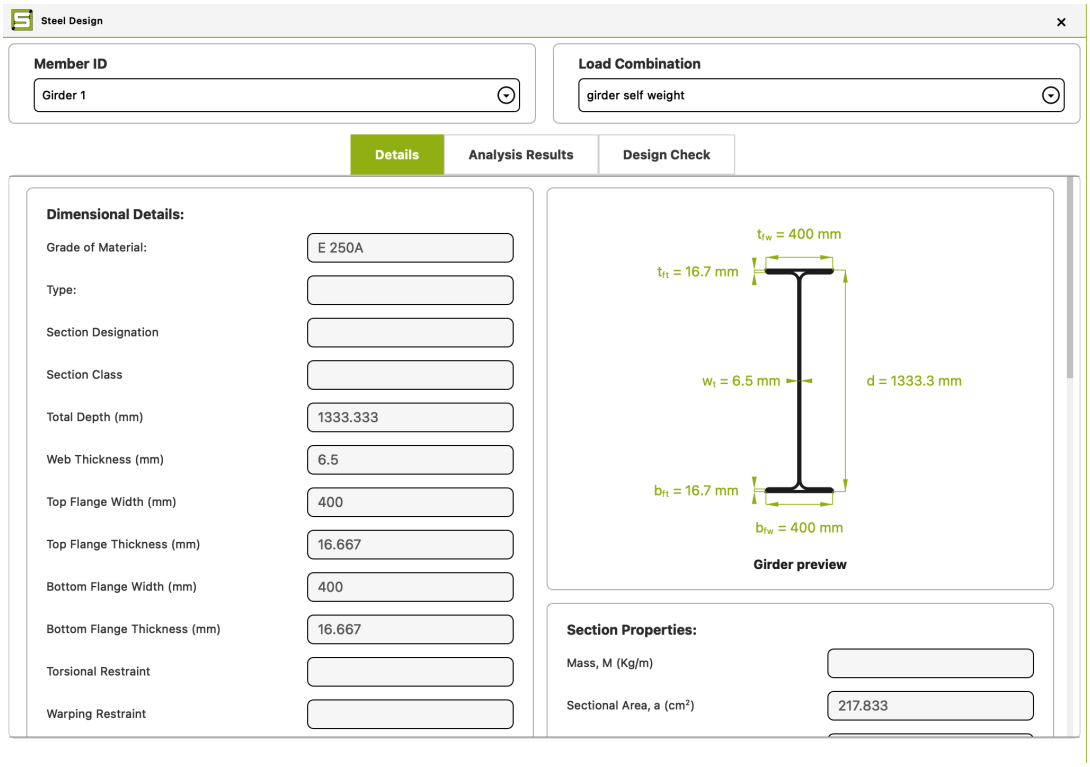


Figure 4.1: Details tab after completing all Task 2 changes: CAD window positioned on the right above Section Properties, all field values sourced from `output_dict`, deflection plot rendered in deep purple (`#6A1B9A`) with the corrected downward sign convention, and dialog guard conditions active.

4.3 Task 2: Python Code & Logic

This section details the implementation of the output dictionary construction and the backend design query interface.

4.3.1 Description of the Script

The implementation centres on two architectural patterns:

- **Post-Design State Aggregation**: A single function collects data from multiple backend sources immediately after the solver completes, assembling a flat, serialisable `output_dict`.
- **Deflection Sign Correction**: A one-pass transformation applied to the grillage `xarray` Dataset before plotting, enforcing the correct structural sign convention throughout the visualisation layer.

4.3.2 Python Code: Output Dictionary & Deflection Fix

Listing 4.1: Output dictionary construction and deflection correction in OsdagBridge

```
# --- Backend Design Query Interface (backend.py) ---
def get_steel_design_state(self) -> dict:
    """Return a structured snapshot of the current steel design state."""
    return {
        "geometry":    self._plate_girder.geometry_dict(),
        "materials":   self._plate_girder.material_dict(),
        "stiffeners":  self._plate_girder.stiffener_dict(),
        "section":     self._plate_girder.section_properties_dict(),
    }

def get_output_state(self) -> dict:
    """Return utilization ratios for all code-check categories."""
    return {
        "util.flexure":     self._checks.flexure_dcr,       # DCR >= 1.0 => fail
        "util.shear":       self._checks.shear_dcr,
```

```
        "util.ltb":          self._checks.ltb_dcr,
        "util.fatigue":      self._checks.fatigue_dcr,
        "util.interaction": self._checks.interaction_dcr,
    }


# --- Output Dictionary Construction (template_page.py) ---
def _build_output_dict(self) -> dict:
    """Aggregate all post-design data into a single output container."""
    steel_state  = self.backend.get_steel_design_state()
    output_state = self.backend.get_output_state()

    # Deflection values from grillage xarray Dataset
    deflection_raw = self._grillage_results["deflection"].values
    deflection      = deflection_raw * -1.0  # Correct: positive = downward

    return {
        # User inputs
        **self._input_dict,
        # Steel design state
        **steel_state,
        # Analysis utilization outputs
        **output_state,
        # Deflection series (sign-corrected)
        "deflection": deflection.tolist(),
        # Output dock UI state
        "ui.active_combo":    self._output_dock.active_combo_text(),
        "ui.active_checks":   self._output_dock.active_checkboxes(),
        "ui.active_radio":    self._output_dock.active_radio(),
    }


# --- Dialog Guard Condition (details_dialog.py) ---
def _open_details_dialog(self) -> None:
    """Open the details dialog only when a valid design result exists."""
    if not self._output_dict:
```

```
52          QMessageBox.information(
53              self,
54              "No Design Result",
55              "Please run a design before opening the Details tab."
56          )
57          return
58      self._details_dialog.load(self._output_dict, self._active_member_id
            )
59      self._details_dialog.show()
```

4.3.3 Explanation of the Code

- **Lines 2–18**: `get_steel_design_state()` and `get_output_state()` form the stable backend API. By encapsulating solver internals behind these methods, the UI layer is fully decoupled from solver data structures; future solver refactors only require updating these two functions.
- **Lines 21–46**: `_build_output_dict` uses Python's dictionary unpacking (`**`) to merge inputs, steel state, and utilization outputs into a single flat container. The deflection array is sign-corrected in-place (line 31) using a scalar multiply of -1.0 before being stored, ensuring the correction is applied exactly once at the source rather than at every render site.
- **Lines 49–60**: The guard condition in `_open_details_dialog` checks for a falsy `_output_dict` (empty dict or `None`) before proceeding. This prevents any downstream `KeyError` or `AttributeError` that would arise from accessing result fields on an uninitialised dictionary.

4.4 Task 2: Documentation

The following directory structure illustrates the files created or modified to implement the output state pipeline, Details tab, and report generation base code.

4.4.1 Directory Structure

Implementation Note: The `output_dict` is constructed once per design run inside `template_page.py` and then passed by reference to every downstream consumer—the Details tab, output dock, and report generator. This single-construction, multi-consumer pattern ensures that all parts of the UI always present data from the same solved state, eliminating the class of bugs that previously arose from each component independently re-querying a potentially mutated backend.

Chapter 5

Conclusions

5.1 Tasks Accomplished

Over the course of this fellowship, I addressed critical architectural and usability challenges within OsdagBridge, transforming the Member Properties system and design result pipeline into a coherent, state-aware engineering interface. Key tasks included:

- **Member Properties Refactoring**: Re-engineered the Member Properties tab with full design-type–based field visibility, pruning irrelevant controls from the Cross Bracing and End Diaphragm sub-tabs and unifying the section-properties UI across all four member types. Generalised the Girder and Stiffener components into reusable base classes and removed the redundant `schema_builder.py` module, reducing the codebase by `1,224` lines while adding `4,096` lines of cleaner, maintainable architecture (PR #48, merged Apr 18).

- **Design Tab & Output State Pipeline**: Introduced `get_steel_design_state()` and `get_output_state()` to the backend, establishing a stable API that decouples the UI from internal solver structures. Built the `output_dict` immediately after each design run, consolidating input parameters, steel design state, utilization ratios (DCR checks), output dock UI state, and grillage deflection values into a single serialisable container (PRs #74 and #80).

- **Details Tab Overhaul**: Migrated the Details tab from direct CAD state access to `output_dict`–driven data loading, fixed the design-check tab ID mismatch, corrected the deflection plot direction (sign convention), applied the project-standard

deep purple colour (`#6A1B9A`), and added dialog guard conditions to prevent premature access before a valid design exists.

- **CAD Layout Alignment & Report Generation**: Repositioned the CAD window in the Details tab to mirror the established Girder UI layout (PR #69) and implemented the foundational infrastructure for engineering report generation, including document structure, section scaffolding, and `output_dict` data-binding.

5.2 Skills Developed

This fellowship provided a rigorous environment for applying software engineering practices to structural design tooling. Key skills developed include:

- **Advanced GUI Development**: Deepened expertise in PySide6, specifically in custom widget creation and layout management with `QLayout` and `QPainter`.
- **Architecture & Design Patterns**: Applied the **Factory**, **Guard Clause**, and **Single Source of Truth** patterns to solve scalability, premature-access, and state-consistency issues across the design result pipeline.
- **State Management**: Designed and implemented a post-design state aggregation system using Python dictionary unpacking and two-stage fallback lookups, ensuring all downstream UI components always consume data from the same solved state.
- **Debugging & Bug Triage**: Sharpened systematic debugging skills by identifying and resolving non-trivial issues including member ID mismatches across design-check tabs, incorrect deflection sign conventions, and project-location field resets on design-type switches.
- **Engineering Domain Knowledge**: Gained practical insight into plate girder design workflows, the significance of DCR checks under IS 800:2007, and the importance of sign convention consistency in structural deflection plots and grillage analysis outputs.
- **Version Control & Collaborative Development**:Managed feature branches, coordinated rebases and PRs, and contributed to collaborative development on the OsdagBridge repository within team.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

Name: Sreejesh S

Project: OsdagBridge

Internship: FOSSEE Fellowship 2025-26

DATE	DAY	TASK	Hours Worked
15-Feb-2026	Sunday	Meeting: Task briefing for Phase 2 fellowship	2
16-Feb-2026	Monday	Reviewed OsdagBridge codebase and PR history	4
17-Feb-2026	Tuesday	Meeting: Discussion on Member Properties refactoring scope	2
18-Feb-2026	Wednesday	Studied existing Member Properties tab structure and identified issues	5
19-Feb-2026	Thursday	Meeting: Design-type based behaviour planning with mentor	2
20-Feb-2026	Friday	Started refactoring Member Properties tab layout	6
21-Feb-2026	Saturday	Implemented design-type based field visibility logic	8
22-Feb-2026	Sunday	Meeting: Code review and feedback on field visibility	2
23-Feb-2026	Monday	Pruned irrelevant fields from Cross Bracing tab	7
24-Feb-2026	Tuesday	Meeting: Discussion on End Diaphragm and welded type consistency	2
25-Feb-2026	Wednesday	Removed irrelevant fields from End Diaphragm tab	6
26-Feb-2026	Thursday	Aligned welded-type options in End Diaphragm with Girder Properties	8
27-Feb-2026	Friday	Meeting: Review of welded type alignment changes	2
28-Feb-2026	Saturday	Testing design-type gating logic across all member tabs	6
01-Mar-2026	Sunday	Meeting: Planning section-properties UI unification	2
02-Mar-2026	Monday	Started unifying section-properties UI across Girder tab	7
03-Mar-2026	Tuesday	Exam preparation – partial work on stiffener tab UI	3
04-Mar-2026	Wednesday	Exam	-
05-Mar-2026	Thursday	Meeting: Post-exam sync on progress	2
06-Mar-2026	Friday	Unified section-properties UI for Stiffener tab	8
07-Mar-2026	Saturday	Meeting: Code walkthrough with Aditya on dialog flow	2
08-Mar-2026	Sunday	Unified section-properties UI for Cross Bracing tab	8
09-Mar-2026	Monday	Meeting: Review of Cross Bracing and End Diaphragm dialogs	2
10-Mar-2026	Tuesday	Unified section-properties UI for End Diaphragm tab	8
11-Mar-2026	Wednesday	Improved dialog interaction flow and preview viewer wiring	7
12-Mar-2026	Thursday	Meeting: Discussion on component generalisation strategy	2
13-Mar-2026	Friday	Generalised Girder and Stiffener into reusable base components	9
14-Mar-2026	Saturday	Meeting: Review of generalised Girder/Stiffener components	2

Internship Work Report

Name: Sreejesh S

Project: OsdagBridge

Internship: FOSSEE Fellowship 2025-26

DATE	DAY	TASK	Hours Worked
15-Mar-2026	Sunday	Continued generalisation; removed schema_builder.py module	8
16-Mar-2026	Monday	Standardised all 'Customized' labels to 'Custom' across components	6
17-Mar-2026	Tuesday	Meeting: Code review of terminology changes	2
18-Mar-2026	Wednesday	Updated default member length from 30.0 to 25.0	4
19-Mar-2026	Thursday	Enhanced thickness value retrieval with SAIL_APPROVED_THICKNESS_VALUES fallback	7
20-Mar-2026	Friday	Meeting: Discussion on segment action widget redesign	2
21-Mar-2026	Saturday	Exam preparation – partial review of segment widget code	3
22-Mar-2026	Sunday	Exam	-
23-Mar-2026	Monday	Redesigned segment action widget with add/remove icon integration	8
24-Mar-2026	Tuesday	Meeting: UI review of segment widget layout and icon caching	2
25-Mar-2026	Wednesday	Implemented icon caching for segment action widget rendering	7
26-Mar-2026	Thursday	Renamed _default_dimension_bounds to _default_dimension_bounds_for_field	4
27-Mar-2026	Friday	Meeting: Pre-PR review with Aditya and Nidhi	2
28-Mar-2026	Saturday	Testing and bug fixes for all Member Properties changes	8
29-Mar-2026	Sunday	Meeting: Final review before PR #48 submission	2
30-Mar-2026	Monday	Resolved review comments and prepared PR #48 branch	7
31-Mar-2026	Tuesday	Raised PR #48: refine additional-inputs CAD section property dialogs	5
01-Apr-2026	Wednesday	Meeting: PR #48 code review discussion	2
02-Apr-2026	Thursday	Addressed review feedback on PR #48 – layout and wiring fixes	8
03-Apr-2026	Friday	Meeting: Follow-up review on PR #48 changes	2
04-Apr-2026	Saturday	Force-pushed rebased branch for PR #48; resolved merge conflicts	7
05-Apr-2026	Sunday	Meeting: Discussion on CAD layout alignment (PR #69 scope)	2
06-Apr-2026	Monday	Started CAD layout alignment – repositioning CAD window in Details tab	8
07-Apr-2026	Tuesday	Aligned CAD previews with captions in Details tab	7
08-Apr-2026	Wednesday	Meeting: Review of Details tab CAD layout changes	2
09-Apr-2026	Thursday	Added shared stiffener table constants across tabs	6
10-Apr-2026	Friday	Meeting: PR #69 walkthrough and testing	2
11-Apr-2026	Saturday	Testing CAD layout changes and visual verification	7

Internship Work Report

Name: Sreejesh S

Project: OsdagBridge

Internship: FOSSEE Fellowship 2025-26

DATE	DAY	TASK	Hours Worked
12-Apr-2026	Sunday	Meeting: Nidhi review – Details tab layout approval	2
13-Apr-2026	Monday	Raised PR #69: Steel Design Details mapping and CAD layout	5
14-Apr-2026	Tuesday	Meeting: Discussed design tab and output state pipeline plan	2
15-Apr-2026	Wednesday	Studied backend structure to plan get_steel_design_state()	5
16-Apr-2026	Thursday	Meeting: Final review before merging PR #48	2
17-Apr-2026	Friday	Final testing and documentation for PR #48	6
18-Apr-2026	Saturday	PR #48 merged – Member Properties refactor complete	3
19-Apr-2026	Sunday	Meeting: Sprint planning for output state pipeline	2
20-Apr-2026	Monday	Implemented get_steel_design_state() in backend	8
21-Apr-2026	Tuesday	Implemented get_output_state() with DCR check categories	8
22-Apr-2026	Wednesday	Meeting: Code review of backend API methods	2
23-Apr-2026	Thursday	Added placeholder DCR checks: flexure, shear, LTB, fatigue, interaction	8
24-Apr-2026	Friday	Meeting: Discussion on output_dict structure	2
25-Apr-2026	Saturday	Implemented steel design state retrieval in template page	8
26-Apr-2026	Sunday	Meeting: Review of template page state sync logic	2
27-Apr-2026	Monday	Testing get_steel_design_state() and get_output_state() integration	7
28-Apr-2026	Tuesday	Raised PR #74: steel design state retrieval and DCR checks	5
29-Apr-2026	Wednesday	Meeting: PR #74 review discussion	2
30-Apr-2026	Thursday	Built output_dict construction logic in template_page.py	9
01-May-2026	Friday	Integrated input_dict, steel state and utilization outputs into output_dict	8
02-May-2026	Saturday	Meeting: Review of output_dict structure and data sources	2
03-May-2026	Sunday	Extracted deflection values from grillage xarray Dataset into output_dict	8
04-May-2026	Monday	Meeting: Discussion on Details tab migration plan	2
05-May-2026	Tuesday	Migrated Details tab to load data from output_dict instead of CAD state	9
06-May-2026	Wednesday	Fixed design check tab ID mismatch bug	7
07-May-2026	Thursday	Meeting: Code review for Details tab and ID fix	2
08-May-2026	Friday	Fixed deflection plot direction – multiplied values by -1	7
09-May-2026	Saturday	Applied deep purple (#6A1B9A) colour to deflection plot	5

Internship Work Report

Name: Sreejesh S

Project: OsdagBridge

Internship: FOSSEE Fellowship 2025-26

DATE	DAY	TASK	Hours Worked
10-May-2026	Sunday	Meeting: Review of deflection plot changes and output dock loading	2
11-May-2026	Monday	Fixed project location bug on design type switch	6
12-May-2026	Tuesday	Loaded output state in dialogs; fixed active_member_id tracking	7
13-May-2026	Wednesday	Added dialog guard conditions to prevent premature access	6
14-May-2026	Thursday	Meeting: Pre-PR #80 final review with Nidhi	2
15-May-2026	Friday	Raised PR #80: Details tab work and output dictionary	5

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.