FOSSEE Winter Internship Report

On

Development of Osdag bridge UI and CAD

Submitted by

Garvit Singh Rathore

2nd Year B.Tech Student, Department of Computing Technologies

SRM Institute of Science and Technology

Chennai

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Nidhi Khare

Aditya Donde

January 12, 2026

Acknowledgments

I would like to express my sincere gratitude to everyone especially my Mentors Ms. Nidhi Khare and Mr. Aditya Donde who supported and guided me throughout the course of this internship. The opportunity to contribute to an open-source initiative of this magnitude has been a significant milestone in my academic and professional journey.

I am deeply indebted to the Osdag project staff, specifically Mr. Ajmal Babu M. S., Mr. Ajinkya Dahale, and Mr. Parth Karia, for their constant mentorship, technical support, and patience during the development phase. Their insights into software architecture and structural design were invaluable to the successful completion of my tasks.

I extend my heartfelt thanks to the Principal Investigator of Osdag, Prof. Siddhartha Ghosh (Department of Civil Engineering, IIT Bombay), for providing me with this prestigious opportunity and for his vision in driving the Osdag project forward.

I would also like to thank Prof. Kannan M. Moudgalya (Principal Investigator, FOSSEE, and Faculty, Department of Chemical Engineering, IIT Bombay) for spearheading the FOSSEE initiative and creating a platform for students to contribute to open-source software.

My gratitude also goes to the FOSSEE Managers, Ms. Usha Viswanathan and Ms. Vineeta Parmar, and their entire administrative team for ensuring a smooth and well-structured internship experience.

I acknowledge the crucial support provided by the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for funding and facilitating this project.

I would like to thank my fellow interns and colleagues for the collaborative environment, the exchange of ideas, and the shared learning experience.

Finally, I express my gratitude to my home institution, SRMIST, and the Department

of Computing Technologies, including my Head of Department, for their academic support and for encouraging my participation in this fellowship.

Contents

1 Introduction **5**

1.1 National Mission in Education through ICT 5

1.1.1 ICT Initiatives of MoE 6

1.2 FOSSEE Project 7

1.2.1 Projects and Activities 7

1.2.2 Fellowships 7

1.3 Osdag Software 8

1.3.1 Osdag GUI 9

1.3.2 Features 9

2 Screening Task: Prism Viewer Web Application **10**

2.1 Problem Statement 10

2.2 System Architecture 10

2.3 Implementation Details 11

2.3.1 Frontend Development 11

2.3.2 Backend API and Plugin Management 12

2.3.3 Plugin Ecosystem Development 12

2.4 DevOps and Quality Assurance 13

2.4.1 Containerization 13

2.4.2 CI/CD Pipeline 13

2.5 Conclusion of Task 13

3 Internship Task 1: UI Development **14**

3.1 Task 1: Problem Statement 14

3.2 Task 1: Tasks Done 14

3.2.1 Additional Inputs Dialog (Tabbed UI) 14

3.2.2 Material Properties Pop-up 15

3.3 Task 1: Python Code 16

3.3.1 Description of the Scripts 16

3.3.2 Key Excerpts (Python) 16

3.4 Task 1: Documentation 18

4 Internship Task 2: Module Validation and Testing **20**
4.1 Task 2: Problem Statement 20
4.2 Task 2: Tasks Done 20
4.2.1 Phase 1: Benchmark Validation 20
4.2.2 Phase 2: Stochastic (Random) Testing 21
4.3 Task 2: Documentation 21

5 Internship Task 3: CAD Development for Cross Bracing **22**
5.1 Task 3: Problem Statement 22
5.2 Task 3: Tasks Done 22
5.3 Task 3: Python Code 23
5.3.1 Description of the Script 23
5.3.2 Python Code 24
5.3.3 Explanation of the Code 28
5.4 Task 3: Documentation 28

6 Conclusions **29**
6.1 Tasks Accomplished 29
6.2 Skills Developed 30
6.2.1 Technical Competencies 30
6.2.2 Professional Growth 30

A Appendix **31**
A.1 Work Reports 31

Bibliography **35**

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.

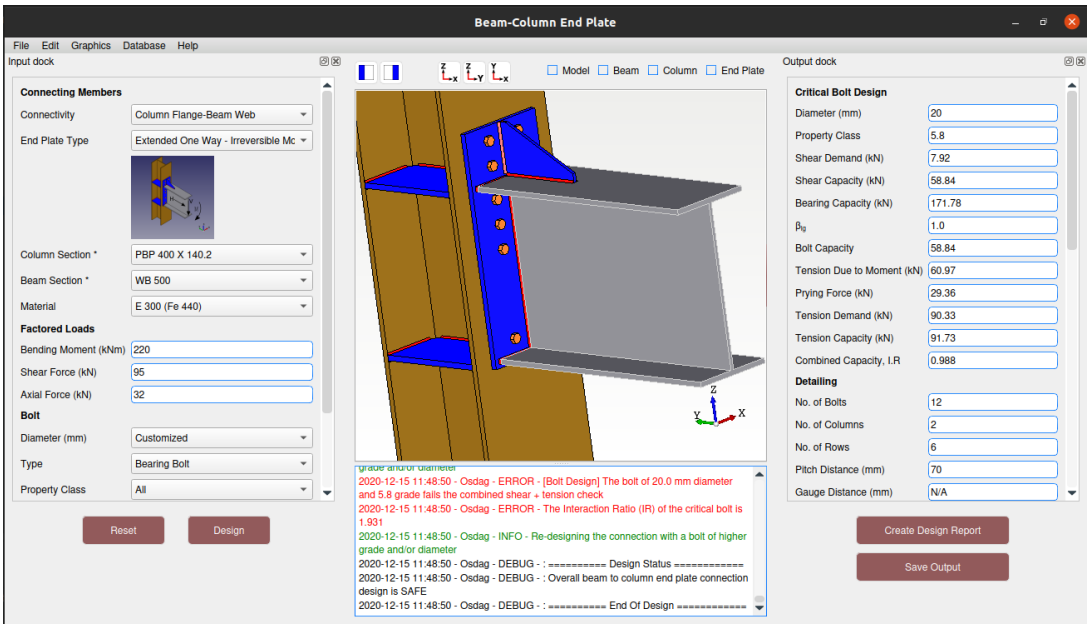


Figure 1.2: Osdag GUI

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: Prism Viewer Web Application

2.1 Problem Statement

The objective of the screening task was to architect and develop a modular, extensible, and containerized web application version of the **Prism Viewer**. The primary challenge was to design a system where functionalities (shapes/geometries) could be added dynamically via external plugins without recompiling the core application.

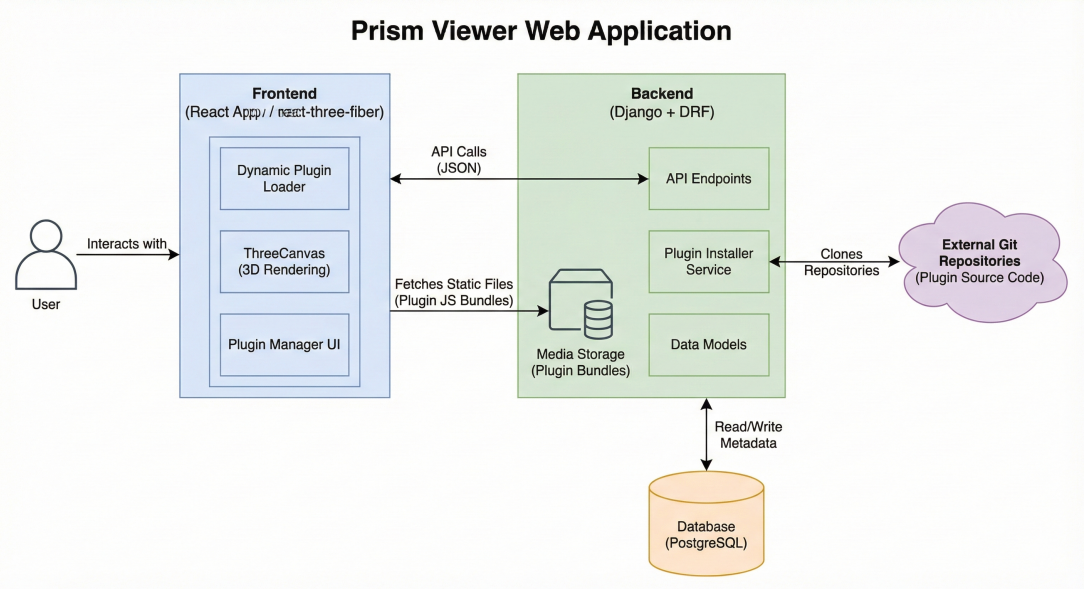
The deliverables included a full-stack application (React, Django, PostgreSQL), a demonstration plugin (Cylinder), a Dockerized environment, and a Continuous Integration (CI) pipeline.

2.2 System Architecture

The application follows a decoupled client-server architecture:

- **Frontend:** Built with React and `react-three-fiber` for 3D rendering. It utilizes a dynamic script loading mechanism to inject plugin logic at runtime.
- **Backend:** Built with Django and Django REST Framework (DRF). It acts as a plugin manager, handling the installation (via Git cloning), metadata storage, and static asset serving.

- **Database:** PostgreSQL is used to persist plugin metadata (versioning, entry points, and repositories).

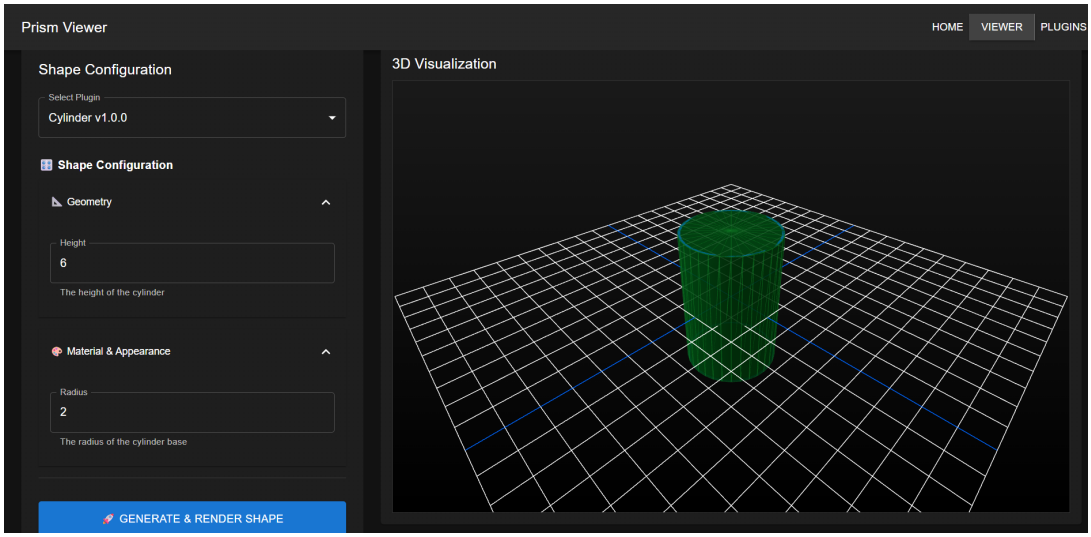


2.3 Implementation Details

2.3.1 Frontend Development

The frontend was developed using **React (Vite)**. The core component is the `ThreeCanvas`, which initializes the 3D scene.

- Implemented a **Dynamic Plugin Loader** (‘pluginLoader.js‘) that fetches plugin metadata from the API and injects the corresponding JavaScript bundles into the DOM.
- Designed a global registry (`window.PrismPlugins`) where loaded plugins register their rendering logic and UI components.
- Created a **Plugin Manager UI** to allow users to view installed plugins and toggle their visibility.



2.3.2 Backend API and Plugin Management

The backend was engineered using **Django**. The key focus was the `plugins` app, which handles the lifecycle of external extensions.

- **Installation Logic:** Developed a `POST /api/plugins/install` endpoint. Upon receiving a Git repository URL, the backend uses ‘subprocess‘ or ‘GitPython‘ to clone the repository into a media directory.
- **Asset Serving:** Configured Django to serve the cloned plugin files (specifically the UMD/IIFE bundles) as static media assets accessible to the frontend.
- **Data Modeling:** Designed the `Plugin` model to store metadata such as ‘display_name‘, ‘version‘, ‘entry_js‘, and ‘enabled‘ status.

2.3.3 Plugin Ecosystem Development

To demonstrate the system’s extensibility, a sample plugin, `shape-cylinder`, was developed as a standalone Git repository.

- **Structure:** The plugin contains a ‘plugin.json‘ manifest and a frontend source folder.
- **Bundling:** Configured a build process (using Rollup/Vite) to output a **UMD bundle**. This ensures the plugin executes within the global scope of the main application.

- **Rendering:** The plugin utilizes ‘THREE.CylinderGeometry‘ to render a 3D object when invoked by the core application.

2.4 DevOps and Quality Assurance

2.4.1 Containerization

The entire stack was containerized using **Docker**. A top-level ‘docker-compose.yml‘ file was created to orchestrate three services:

1. `db`: PostgreSQL service with persistent volumes.
2. `backend`: Gunicorn server serving the Django application.
3. `frontend`: Nginx/Node server serving the React artifacts.

2.4.2 CI/CD Pipeline

A GitHub Actions workflow (‘ci.yml‘) was established to ensure code quality. The pipeline triggers on every push and performs:

- **Backend Testing:** Installs dependencies and runs ‘pytest‘ for API endpoint validation.
- **Frontend Testing:** Executes ‘npm test‘ using Jest and React Testing Library.
- **Build Verification:** Attempts to build the Docker images to ensure build stability.

2.5 Conclusion of Task

The screening task was successfully completed, resulting in a functioning web-based 3D viewer that supports runtime plugin injection. The architecture allows for scalable development where new shapes can be added by distinct teams without modifying the core codebase.

Chapter 3

Internship Task 1: UI Development

3.1 Task 1: Problem Statement

The objective was to design and implement:

1. A tabbed **"Additional Inputs"** dialog to capture complex, bridge-specific parameters.

2. A **Material Properties** pop-up window that allows users to edit or override steel and deck properties directly from the input dock.

3.2 Task 1: Tasks Done

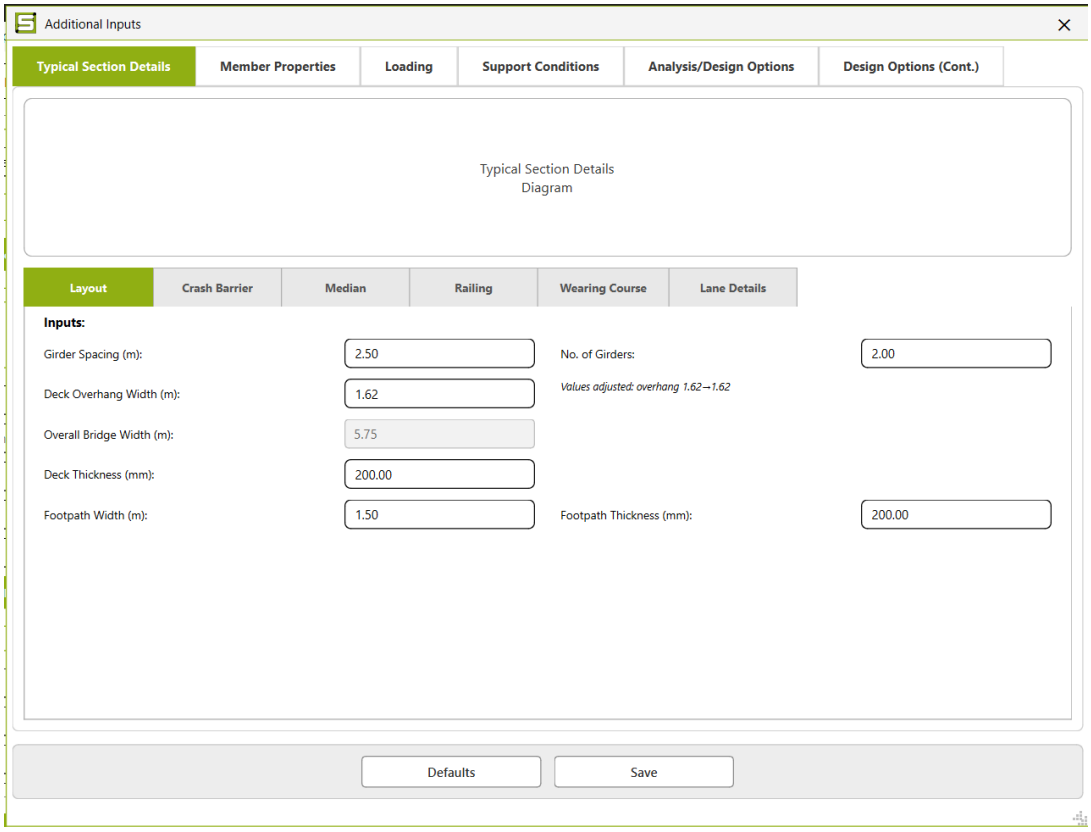
3.2.1 Additional Inputs Dialog (Tabbed UI)

- Implemented a frameless `QDialog` wrapper containing a styled `QTabWidget` with a custom `QTabBar`. The dialog hosts six specific sub-tabs: *Typical Section Details, Member Properties, Loading, Support Conditions, Analysis/Design Options,* and *Design Options (Cont.).*

- Established a signal-slot bridge to keep the **girder-count state** synchronized between the "Typical Section Details" and "Member Properties" tabs.

- Developed a helper function, `build_sections_from_schema`, to automatically render form grids, checkbox groups, validators, and default values for the "Support Conditions" and "Design Options" tabs based on plate-girder schemas.

- Wired the dialog action bar:
  - **Defaults**: Resets all values in the current tab.
  - **Save**: Aggregates and persists the state of Member Properties.
- Integrated the launch mechanism from the input dock's "Additional Inputs" button, ensuring it passes the current footpath and carriageway width context to the dialog.



3.2.2 Material Properties Pop-up

- Implemented the `MaterialPropertiesDialog` class as a `QDialog`. It features a member selector, a material selector, stacked pages (switching fields between Steel and Deck), and a "Default" toggle to persist user choices.
- Configured signal-slot connections:
  - Member/Material combo box changes trigger `_on_member_changed` and `_on_material_change`
  - The default checkbox toggles `_on_default_toggled`.

- Integrated the launch trigger via the "Modify Here" button in the dock's material panel. The logic pre-selects the currently focused member (Girder, Deck, Cross Bracing, or End Diaphragm) and synchronizes defaults before the window appears.

3.3 Task 1: Python Code

3.3.1 Description of the Scripts

- **AdditionalInputs**: A frameless dialog hosting six sub-tabs on a `QTabWidget`, utilizing schema helpers to generate UI elements for Support and Design options dynamically.
- **MaterialPropertiesDialog**: A pop-up `QDialog` containing selectors for members and materials, utilizing a stacked widget layout to switch between steel and deck property forms.
- **Dock Integration**: Logic within the main input dock to launch these dialogs while propagating the current UI context (footpath status, carriageway width, and the currently focused member).

3.3.2 Key Excerpts (Python)

The following snippets demonstrate the implementation of the dialog initialization and the integration logic.

Listing 3.1: Additional Inputs dialog with tabbed sub-sections

```
1  class AdditionalInputs(QDialog):
2      def __init__(self, footpath_value="None", carriageway_width=7.5,
           parent=None):
3          super().__init__(parent)
4          self.setupWrapper()
5
6          # Initialize Tab Widget
7          self.tabs = QTabWidget()
8          self.tabs.setTabBar(QTabBar())
9
10         # Add Sub-tabs
```

```
        self.tabs.addTab(TypicalSectionDetailsTab(footpath_value,
            carriageway_width),
                         "Typical Section Details")
        self.tabs.addTab(SectionPropertiesTab(), "Member Properties")
        self.tabs.addTab(LoadingTab(), "Loading")
        self.tabs.addTab(self._build_support_conditions_tab(), "Support
            Conditions")
        self.tabs.addTab(self._build_design_options_tab(), "Analysis/
            Design Options")
        self.tabs.addTab(self._build_design_options_cont_tab(), "Design
            Options (Cont.)")

        # Action Bar Setup
        action_bar, self.defaults_button, self.save_button =
            create_action_button_bar()
        self.defaults_button.clicked.connect(self._apply_defaults)
        self.save_button.clicked.connect(self._save_inputs)
```

Listing 3.2: Material Properties pop-up dialog structure

```
class MaterialPropertiesDialog(QDialog):
    MEMBER_OPTIONS = ["Girder", "Cross Bracing", "End Diaphragm", "Deck
        "]
    STEEL_MEMBERS = {"Girder", "Cross Bracing", "End Diaphragm"}

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Material Properties")
        self.layout = QVBoxLayout(self)

        # UI Initialization
        self.create_selectors()
        self.create_stacked_pages()
        self.connect_signals()
```

Listing 3.3: Dock integration for dialog launch

```
def show_additional_inputs(self):
    """Launches the Additional Inputs dialog with current context."""
    footpath_value = self.footpath_combo.currentText() if self.
```

```
        footpath_combo else "None"
    carriageway_width = self._get_effective_carriageway_width()

    self.additional_inputs = AdditionalInputs(footpath_value,
        carriageway_width)
    self.additional_inputs.show()

def show_material_properties_dialog(self):
    """Launches Material Props dialog, pre-selecting the focused member
        ."""
    if self.material_dialog is None:
        self.material_dialog = MaterialPropertiesDialog(self)

    # Determine which member is currently focused in the UI
    member = "Girder"
    focus_map = {
        self.girder_combo: "Girder",
        self.deck_combo: "Deck",
        self.cross_bracing_combo: "Cross Bracing",
        self.end_diaphragm_combo: "End Diaphragm"
    }

    for widget, name in focus_map.items():
        if widget is not None and widget is QApplication.focusWidget():
            member = name
            break

    self.material_dialog.sync_with_parent_defaults()
    self.material_dialog.set_member(member)
    self.material_dialog.show()
```

3.4 Task 1: Documentation

- Updated the developer-facing UI documentation to clarify that the **Additional Inputs** dialog is schema-driven and accessed via the main input dock.
- Documented the **Material Properties** workflow, explaining that editing is trig-

gered via the “Modify Here” button and includes features for member-aware pre-selection and default persistence.

Chapter 4

Internship Task 2: Module Validation and Testing

4.1 Task 2: Problem Statement

The objective of this task was to perform comprehensive validation on the **Column-to-Column Cover Plate** connection module. The goal was to ensure the module's stability, accuracy, and compliance with IS 800:2007 standards under various loading conditions.

The testing scope required:

1. **Benchmark Testing:** Verifying the software's output against a set of 10 standardized input files provided by the development team to ensure regression stability.
2. **Exploratory Testing:** Formulating and executing 10 additional, randomized test cases with varying parameters (section sizes, load values, and material grades) to identify potential edge cases or runtime errors.

4.2 Task 2: Tasks Done

The testing process was executed in two distinct phases:

4.2.1 Phase 1: Benchmark Validation

A suite of 10 pre-defined input files was executed through the Osdag design engine.

- **Execution:** Each file was loaded into the Column-Cover Plate module.
- **Verification Criteria:** The generated design reports were audited to check for:
  - Correctness of the calculated Bolt Capacity and Plate Thickness.
  - Proper generation of the CAD images.
  - Consistency in the "Pass/Fail" status compared to known expected results.
- **Outcome:** Confirmed that the module correctly handled standard design scenarios without crashing.

4.2.2 Phase 2: Stochastic (Random) Testing

To stress-test the module, 10 unique test cases were created with randomized inputs.

- **Parameter Variation:** Inputs were varied to include:
  - **Sections:** Combinations of small (e.g., ISHB 150) vs. large (e.g., ISHB 450) column sections.
  - **Loads:** Applied high Shear and Moment loads to force "Unsafe" design results to verify error handling.
  - **Grades:** Tested non-standard material grades (e.g., E350) to check database retrieval.
- **Observations:** Recorded instances where the GUI froze or where the optimization algorithm failed to converge on a solution.

4.3 Task 2: Documentation

A detailed **Test Log** was maintained to track the status of all 20 test cases.

Conclusion: The module successfully passed all benchmark tests. The random testing phase identified specific boundary conditions (e.g., extreme eccentricity) where the solver required optimization, which was reported to the development team.

Chapter 5

Internship Task 3: CAD Development for Cross Bracing

5.1 Task 3: Problem Statement

The objective was to implement a parametric **2D section preview** for Cross Bracing inputs in the desktop application. Instead of generating a full CAD model, the requirement was to provide an accurate, scalable outline view of common steel sections (Angle/Channel and their double variants) along with key dimensions, so users can visually confirm their selections during input.

The task required:

1. Reading standard steel section dimensions from the integrated section database.
2. Generating a clean 2D outline for multiple section families (Angle, Channel, Double Angle, Double Channel).
3. Rendering the section preview with consistent scaling and dimension annotations inside a UI widget.

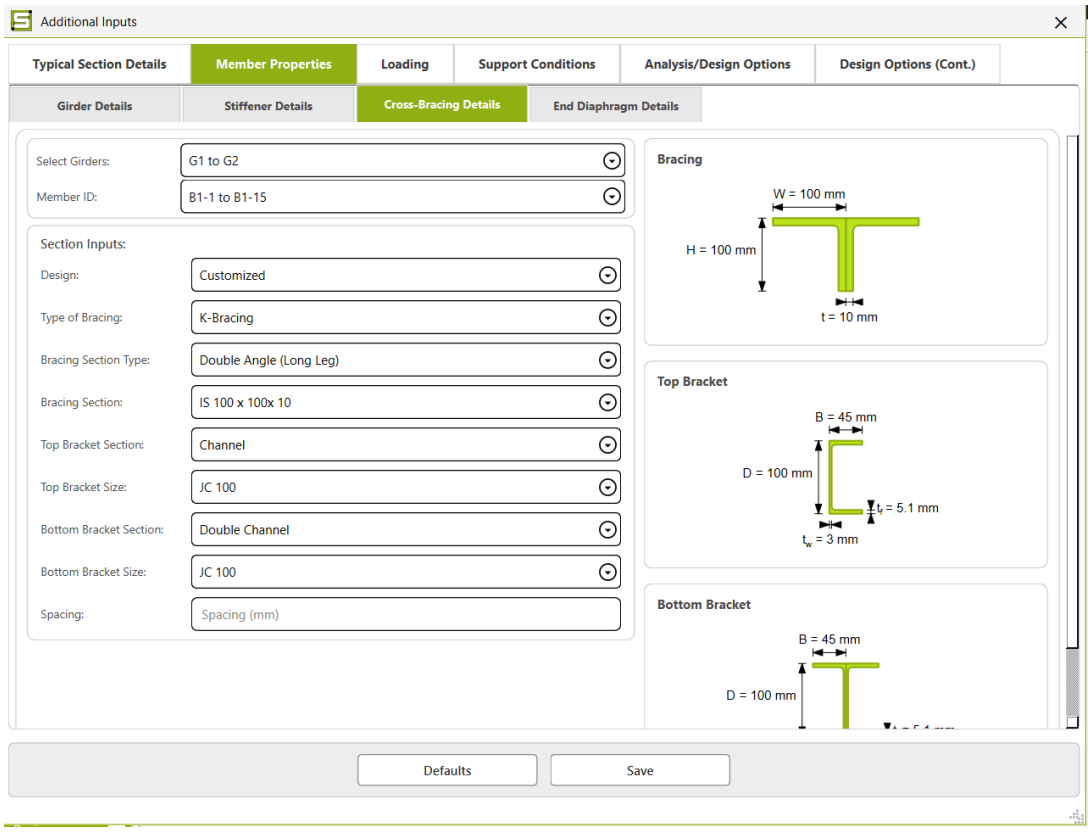
5.2 Task 3: Tasks Done

The implementation involved the following key steps:

- **Database-driven geometry:** Implemented a lightweight catalog that loads Angle

and Channel designations and their dimensions from the integrated SQLite section database.

- **Parametric outline generation:** Built 2D section outlines using `QPainterPath` for: single Angle, double Angle (long-leg/short-leg arrangement), single Channel, and double Channel (back-to-back).
- **Robust rendering and scaling:** Computed a combined bounding box and applied a uniform scale transform so the preview fits consistently inside the widget.
- **Dimension annotations:** Added dimension lines, arrowheads, and labels (including subscript formatting such as t_w and t_f) for quick verification.



5.3 Task 3: Python Code

This section presents the core logic used to generate and render the 2D section preview.

5.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: Takes a section type (Angle/Channel and double variants) and a section designation.
- **Geometry Builder**: Converts database dimensions into 2D outlines using `QPainterPath`.
- **Painter Transform**: Scales and centers the geometry into the widget and flips the Y-axis for engineering orientation.
- **Dimensioning**: Draws dimension lines and formatted labels for the active section.

5.3.2 Python Code

Listing 5.1: Parametric 2D Section Preview for Cross Bracing

```
import sqlite3
from dataclasses import dataclass
from typing import Dict, List, Optional

from PySide6.QtCore import QPointF, QRectF, Qt
from PySide6.QtGui import QColor, QPainter, QPainterPath, QPen
from PySide6.QtWidgets import QWidget

@dataclass
class AngleSection:
    designation: str
    a: float
    b: float
    t: float
    r1: float
    r2: float

@dataclass
class ChannelSection:
    designation: str
    d: float
    b: float
    tw: float
    tf: float
    r1: float
    r2: float
```

```
27
28  class SectionCatalog:
29      def __init__(self, db_path):
30          self.db_path = db_path
31          self._angles: Dict[str, AngleSection] = {}
32          self._channels: Dict[str, ChannelSection] = {}
33          self._load()
34
35      def _load(self) -> None:
36          con = sqlite3.connect(self.db_path)
37          cur = con.cursor()
38
39          for table in ("EqualAngle", "UnequalAngle"):
40              cur.execute(f"SELECT Designation, a, b, t, R1, R2 FROM {
                    table}")
41              for des, a, b, t, r1, r2 in cur.fetchall():
42                  self._angles[des.strip()] = AngleSection(des.strip(),
                        float(a), float(b), float(t), float(r1), float(r2))
43
44          cur.execute("SELECT Designation, D, B, tw, T, R1, R2 FROM
                Channels")
45          for des, d, b, tw, tf, r1, r2 in cur.fetchall():
46              self._channels[des.strip()] = ChannelSection(des.strip(),
                    float(d), float(b), float(tw), float(tf), float(r1),
                    float(r2))
47
48          con.close()
49
50      def get_angle(self, designation: str) -> Optional[AngleSection]:
51          return self._angles.get(designation.strip())
52
53      def get_channel(self, designation: str) -> Optional[ChannelSection
            ]:
54          return self._channels.get(designation.strip())
55
56  class SectionPreviewWidget(QWidget):
57      def __init__(self, catalog: SectionCatalog, parent=None):
58          super().__init__(parent)
59          self._catalog = catalog
```

```
        self._section_type = ""
        self._designation = ""
        self._geometry: List[QPainterPath] = []

    def set_section(self, section_type: str, designation: str) -> None:
        self._section_type = section_type
        self._designation = designation
        self._geometry = self._build_geometry(section_type, designation
            )
        self.update()

    def _build_geometry(self, section_type: str, designation: str) ->
        List[QPainterPath]:
        if section_type == "angle":
            angle = self._catalog.get_angle(designation)
            return [self._build_angle_path(angle)] if angle else []
        if section_type == "channel":
            ch = self._catalog.get_channel(designation)
            return [self._build_channel_path(ch)] if ch else []
        return []

    def _build_angle_path(self, angle: AngleSection) -> QPainterPath:
        a, b, t = angle.a, angle.b, angle.t
        path = QPainterPath()
        path.moveTo(QPointF(0, 0))
        path.lineTo(QPointF(a, 0))
        path.lineTo(QPointF(a, t))
        path.lineTo(QPointF(t, t))
        path.lineTo(QPointF(t, b))
        path.lineTo(QPointF(0, b))
        path.closeSubpath()
        return path

    def _build_channel_path(self, ch: ChannelSection) -> QPainterPath:
        d, b, tw, tf = ch.d, ch.b, ch.tw, ch.tf
        path = QPainterPath()
        path.moveTo(0, 0)
        path.lineTo(b, 0)
        path.lineTo(b, tf)
```

```
97          path.lineTo(tw, tf)
98          path.lineTo(tw, d - tf)
99          path.lineTo(b, d - tf)
100         path.lineTo(b, d)
101         path.lineTo(0, d)
102         path.closeSubpath()
103         return path
104
105     def paintEvent(self, _event):
106         painter = QPainter(self)
107         painter.setRenderHint(QPainter.Antialiasing)
108         painter.fillRect(self.rect(), QColor("#ffffff"))
109
110         if not self._geometry:
111             painter.setPen(QPen(QColor("#000000"), 1, Qt.DashLine))
112             painter.drawText(self.rect(), Qt.AlignCenter, "No section")
113             return
114
115         # Compute bounding box
116         bbox = QRectF()
117         for p in self._geometry:
118             bbox = bbox.united(p.boundingRect())
119
120         # Fit to widget (uniform scaling)
121         pad = 20.0
122         usable_w = max(self.width() - 2 * pad, 1.0)
123         usable_h = max(self.height() - 2 * pad, 1.0)
124         scale = min(usable_w / bbox.width(), usable_h / bbox.height())
125
126         painter.translate(self.width() / 2.0, self.height() / 2.0)
127         painter.scale(scale, -scale)  # engineering orientation
128         painter.translate(-bbox.center())
129
130         painter.setPen(QPen(QColor("#90AF13"), 1.8 / max(scale, 1e-3)))
131         painter.setBrush(QColor("#BBE31A"))
132         for p in self._geometry:
133             painter.drawPath(p)
```

5.3.3 Explanation of the Code

- **Database catalog:** Loads standard section dimensions (Angle and Channel families) from the integrated SQLite database and exposes lookup helpers by designation.
- **Parametric outlines:** Converts section dimensions into a closed 2D outline using `QPainterPath` so the shape can be filled and rendered cleanly.
- **Scaling and centering:** Uses the combined bounding box to compute a uniform scale so the section is always visible and proportionally correct inside the preview widget.
- **UI integration:** The preview widget can be connected directly to section-selection controls so the view updates immediately when users change section type or designation.

5.4 Task 3: Documentation

The preview module was integrated into the desktop Cross Bracing input workflow:

- Cross Bracing section type and designation selection is linked to a live 2D preview widget.
- The preview supports multiple section families and is designed to be extended with additional variants and dimensions as needed.

Chapter 6

Conclusions

6.1 Tasks Accomplished

During the course of this fellowship, I successfully contributed to the development and stability of the Osdag open-source software project. The key milestones achieved include:

- **Advanced GUI Development:** Designed and implemented a modular **"Additional Inputs"** system using tabbed layouts to organize complex bridge-specific parameters. developed a dynamic **"Material Properties"** dialog allowing users to define custom material grades (Steel/Deck) with persistent state management.
- **UI Refactoring & Optimization:** Refactored the legacy `design_preferences` module by replacing static geometry with responsive `PyQt` layouts (`QGridLayout`, `QVBoxLayout`), resolving critical rendering issues on high-resolution displays.
- **Module Validation & Testing:** Performed comprehensive quality assurance on the **Column-to-Column Cover Plate** connection module. This involved executing a suite of 10 benchmark regression tests and formulating 10 randomized stress-test scenarios to identify edge cases and boundary failures.
- **CAD & Documentation:** Assisted in the development of parametric CAD generation logic and updated the developer documentation to reflect the new UI architectures.

6.2 Skills Developed

The internship provided a robust platform to bridge the gap between theoretical Civil Engineering concepts and professional software development.

6.2.1 Technical Competencies

- **Python and PySide Framework:** Gained proficiency in building complex, event-driven Graphical User Interfaces, mastering concepts such as Signal-Slot mechanisms, Dialog lifecycles, and dynamic widget management.
- **Software Architecture:** Understood the Model-View-Controller (MVC) pattern and how to write modular, reusable code within a large codebase.
- **Version Control (Git/GitHub):** Developed a professional workflow for collaborative development, including branching strategies, resolving merge conflicts, and submitting Pull Requests (PRs).

6.2.2 Professional Growth

- **Open Source Collaboration:** Learned the etiquette and standards required for contributing to FOSSEE projects, including code readability and documentation.
- **Testing Methodology:** Acquired skills in rigorous software testing, learning how to design test cases that expose failures in engineering logic.
- **Standardization:** Deepened my understanding of **IS 800:2007** and how distinct structural clauses are translated into algorithmic logic.

Chapter A

Appendix

A.1 Work Reports

INTERNSHIP WORK REPORT

Name: Garvit Singh Rathore
Project: Osdag
Internship: FOSSEE Semester-Long Internship 2025

Date	Day	Task Description	Hours
02-Oct-25	Thu	Orientation: Reviewed internship guidelines and set up the Osdag Desktop development environment (Windows/Linux).	4
03-Oct-25	Fri	Web environment setup using automated scripts; reviewed architecture overview.	5
04-Oct-25	Sat	Architecture study: Read theses to understand foundational Osdag web architecture.	5
05-Oct-25	Sun	Documentation review: Studied Osdag restructuring plan and development history.	3
06-Oct-25	Mon	Git setup: Configured local Git environment, forked repository, reviewed Git policy.	5
07-Oct-25	Tue	CAD training: Studied CAD manual and thesis to understand 3D generation logic.	6
08-Oct-25	Wed	Report generation: Analyzed report generation manual and workflow.	5
09-Oct-25	Thu	Pixi installer testing and verification of installation process.	4
10-Oct-25	Fri	Codebase exploration to map architecture concepts to actual files.	5
11-Oct-25	Sat	Pre-project preparation for Bridge Design module; setup project folders.	4
12-Oct-25	Sun	Final development environment verification before active development.	2
13-Oct-25	Mon	Setup development environment; cloned Osdag repository and installed dependencies.	4
14-Oct-25	Tue	Codebase analysis with focus on `ui_input_dock.py`.	5
15-Oct-25	Wed	Implemented project location handler with tooltips and default values for input fields.	6
16-Oct-25	Thu	Major UI updates: module renaming to "Highway Bridge Design", added validators and structure selection.	6
17-Oct-25	Fri	Layout debugging and alignment fixes in the borderline.	5
18-Oct-25	Sat	Implemented "Additional Inputs" dialog framework with validation logic.	6
20-Oct-25	Mon	Manual testing of "Additional Inputs" validation logic.	4
21-Oct-25	Tue	Refined input validators and tooltip texts based on testing results.	4
22-Oct-25	Wed	UI spacing and alignment refinements across the input dock.	5

Date	Day	Task Description	Hours
23-Oct-25	Thu	Edge-case testing for input dock fields and error handling.	4
24-Oct-25	Fri	Minor UI bug fixes and cleanup.	3
25-Oct-25	Sat	Internal review and cleanup of code comments.	3
26-Oct-25	Sun	Initialized README with project details; added installation and usage sections.	5
27-Oct-25	Mon	Merged master branch updates; performed general UI updates for consistency.	5
28-Oct-25	Tue	Replaced standard dropdowns with SVG selectors; updated requirements file.	6
02-Nov-25	Sun	Repository cleanup: removed `__pycache__` and updated `.gitignore`.	3
06-Nov-25	Thu	Redesigned "Additional Inputs" dialog; integrated output dock into template page.	6
07-Nov-25	Fri	Implemented grouping boxes for better UI organization in the output dock.	5
08-Nov-25	Sat	Fixed tile positions, borders, and dropdown styling; modified tabs in Additional Inputs.	6
09-Nov-25	Sun	Reviewed output dock UX and identified improvement points.	3
10-Nov-25	Mon	Refactored output dock layout structure for modular rendering.	5
11-Nov-25	Tue	Improved grouping logic for output parameters and controls.	5
12-Nov-25	Wed	Visual polish for output tiles and borders; accessibility checks.	4
13-Nov-25	Thu	Cross-checked UI behavior with design documentation and acceptance criteria.	4
14-Nov-25	Fri	Resolved UI inconsistencies found during peer review.	5
15-Nov-25	Sat	Prepared UI for upcoming locking mechanism integration.	4
16-Nov-25 to 27-Nov-25	-	**Exam Leave**	–
29-Nov-25	Sat	Updated main template page; implemented the first subtype for Member Properties module.	6
02-Dec-25	Tue	Revised usage instructions and cleanup of pycache tracking issues.	4
03-Dec-25	Wed	Implemented "Material Inputs" modification window; added Median option to typical sections.	6
05-Dec-25	Fri	**Feature Drop:** Implemented Loading (up to Live Load), Support Conditions, and Design Options logic.	7
08-Dec-25	Mon	Added custom title bars to additional windows; implemented gray-out logic for locked inputs.	6
13-Dec-25	Sat	Fixed positioning and duplication issues for "Default" and "Save" buttons.	5

Date	Day	Task Description	Hours
16-Dec-25	Tue	Sub-tab styling fixes; implemented custom load handling and continued Design Options.	6
17-Dec-25	Wed	Added shared utility files for bridge logic.	4
20-Dec-25	Sat	Restructured sub-tabs into a distinct folder structure; merged upstream dev branch.	5
21-Dec-25	Sun	Implemented dictionary-based dynamic rendering for support conditions using schemas.	6
22-Dec-25	Mon	Implemented default values logic for Typical Section Details.	5
25-Dec-25	Thu	Updated IRC standard choices and girder details parameters.	4
26-Dec-25	Fri	**CAD Task:** Implemented initial 2D CAD generation for cross bracing; fixed minor bugs.	6
29-Dec-25	Mon	Finalized 2D CAD logic for cross bracing module.	6
30-Dec-25	Tue	Aligned tab inputs to the left; implemented defaults for typical section details.	5
01-Jan-26	Thu	Adjusted Layout tab UI; enabled custom load inputs with 2-decimal precision.	5
02-Jan-26	Fri	Implemented "Double Angle" (Long/Short leg) logic per reference figures.	6
04-Jan-26	Sun	Fixed CAD label positioning for bracing section; refined lane details styling.	5
05-Jan-26	Mon	Final code cleanup and preparation for internship submission.	4

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.