FOSSEE Semester Long Internship-Spring Report

On

Software Development of 3psLCCA(Life-Cycle Cost Analysis) app for Osdag

Submitted by

Jawwad Ahmad

2nd Year B.Tech(CSE(Data Science)) Student, Department of Computer Engineering,

Jamia Millia Islamia

New Delhi

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Swastik Gupta

June 11, 2026

Acknowledgments

- Start with a general statement of thanks. Express your overall gratitude to everyone who supported you during your project or research.
- Project staff at the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- Acknowledge your colleagues who worked with you during your internship or project.
- If appropriate, thank your college, department, head, and principal for their support during your studies.

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.

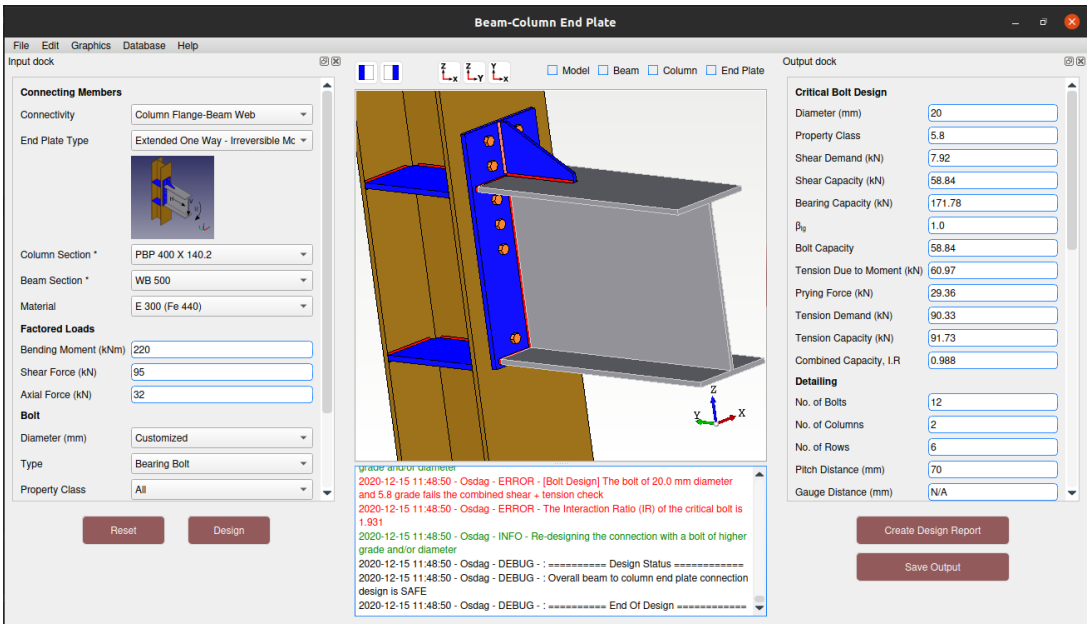


Figure 1.2: Osdag GUI

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

2.1.1 Task 1: Shear Force and Bending Moment Diagrams (2D)

The objective of this task was to extract and visualize the Shear Force Diagram (SFD) and Bending Moment Diagram (BMD) for the central longitudinal girder of a bridge model. The internal force data was provided in a NetCDF dataset (`screening_task1.nc`) containing element-wise values of shear forces and bending moments at both the i and j ends of each finite element.

The program was required to read, process, and visualize the distribution of internal forces along the girder length to aid in structural analysis and design interpretation.

2.1.2 Task 2: 3D Visualization of Internal Forces

The objective of the second task was to visualize SFD and BMD in a 3D format for the longitudinal girders of the bridge structure. Using `xarray` for data handling and `matplotlib`'s 3D plotting tools, the program generates detailed 3D extruded visualizations.

These diagrams help in understanding how internal forces vary spatially along the length and breadth of the bridge, offering an improved visual analysis compared to conventional 2D plots.

2.2 Tasks Done

2.2.1 Theoretical Background

Shear Force (SFD): Shear force is the internal reaction force developed in a structural element to resist transverse loading. It acts perpendicular to the beam's longitudinal axis, and its variation along the beam indicates zones of maximum shear stress.

Bending Moment (BMD): The bending moment is the internal moment that induces bending in a beam. It is the product of the shear force and the perpendicular distance from the section considered. The BMD helps locate points of maximum bending, which are critical for flexural design.

2.2.2 Code Implementation: Task 1 (2D Analysis)

The Python implementation uses the `xarray` library for handling NetCDF data and `matplotlib` for visualization.

Data Loading and Component Detection

Due to variations in solver conventions, internal force components may appear under different label names. A helper function was implemented to robustly detect component labels containing substrings such as "Mz_i" or "Vy_j".

Listing 2.1: Data Loading and Component Detection

```
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt


# Load dataset
ds = xr.open_dataset("screening_task1.nc")
F = ds["forces"] # dims: ('Element', 'Component')


# Helper to find component names robustly
def find_component(cands):
    labels = [str(x) for x in F.coords['Component'].values]
```

```
    low = [s.lower() for s in labels]
    for cand in cands:
        c = cand.lower()
        for lab, l in zip(labels, low):
            if c in l: return lab
    raise KeyError(f"Could not find any of {cands}")

# Detect specific component names
name_Mz_i = find_component(["Mz_i", "Mz i", "Mzi"])
name_Mz_j = find_component(["Mz_j", "Mz j", "Mzj"])
```

Data Extraction and Plotting

The central longitudinal girder is defined by a specific sequence of element IDs. We extract the forces for these elements and construct the chainage (distance) array.

Listing 2.2: Extracting Forces and Plotting BMD

```
# Central longitudinal girder element IDs
girder_ids = [15, 24, 33, 42, 51, 60, 69, 78, 83]
sub = F.sel(Element=girder_ids)

# Pull i/j end values
Mz_i = sub.sel(Component=name_Mz_i).values.astype(float)
Mz_j = sub.sel(Component=name_Mz_j).values.astype(float)

# Construct node-wise sequences
Mz_line = np.concatenate([[Mz_i[0]], Mz_j])
chainage = np.arange(len(girder_ids) + 1, dtype=float)

# --- Plotting BMD ---
plt.figure()
plt.plot(chainage, Mz_line, marker="o", color="red")
for xi, bm in zip(chainage, Mz_line):
    plt.vlines(xi, 0, bm, colors='red', linestyles='solid',
       linewidth=1)
```

```
plt.axhline(0, linewidth=1, color='black')
plt.xlabel("Distance (m)")
plt.ylabel("Bending moment Mz (kN.m)")
plt.title("Bending Moment Diagram - Central Girder")
plt.grid(True)
plt.savefig("BMD_central_girder.png", dpi=220)
```

2.2.3 Code Implementation: Task 2 (3D Visualization)

The 3D implementation visualizes the forces as extrusions from the bridge deck. It uses `Line3DCollection` for performance and color mapping to indicate force magnitude and sign.

Listing 2.3: 3D Visualization using Matplotlib and Xarray

```
from mpl_toolkits.mplot3d.art3d import Line3DCollection

# User Options
MAKE = ["BMD", "SFD"]
COLORMAP = "coolwarm"
EXTRUSION_THICK = 2.2

def make_3d(which):
    """Generates 3D extrusion plots for the specified force type.
       """
    fig = plt.figure(figsize=(11.5, 7.2))
    ax = fig.add_subplot(111, projection="3d")

    # Scaling factors for visualization
    vmax = float(np.nanmax(np.abs(all_vals))) or 1.0
    scale = (max(all_x) / 12.0) / vmax
    cmap = plt.get_cmap(COLORMAP)

    for gname, d in per_girder_data.items():
        xs, zs, vals = d["xs"], d["zs"], d["vals"]
        y0 = np.zeros_like(xs)
```

```
        y1 = vals * scale

        # Extrusions
        segs = np.stack([np.column_stack([xs, y0, zs]),
                         np.column_stack([xs, y1, zs])], axis=1)
        colors = cmap(0.5 + 0.5 * (vals / vmax))
        coll = Line3DCollection(segs, colors=colors, linewidths=
           EXTRUSION_THICK)
        ax.add_collection(coll)

    ax.set_title(f"3D {which} Diagram")
    ax.set_zlabel("Z (m)")

    # Add Colorbar
    mappable = plt.cm.ScalarMappable(cmap=COLORMAP)
    mappable.set_clim(-vmax, vmax)
    plt.colorbar(mappable, ax=ax, shrink=0.75, label=f"{which}
       Magnitude")

    plt.savefig(f"Task2_{which}_3D.png", dpi=260)
```

2.3 Interpretation of Results

2.3.1 Shear Force and Bending Moment Diagrams

The generated 2D diagrams provide critical insights into the structural behavior of the central longitudinal girder.

Shear Force Diagram (SFD): As shown in Figure 2.1, the diagram illustrates the location of maximum shear force, points where the shear force changes sign (zero crossing), and segments where the girder undergoes high shear stress.

Bending Moment Diagram (BMD): Figure 2.2 reveals maximum positive and negative bending moments and points of contraflexure where the bending moment becomes zero. These observations directly influence flexural design and section optimization.

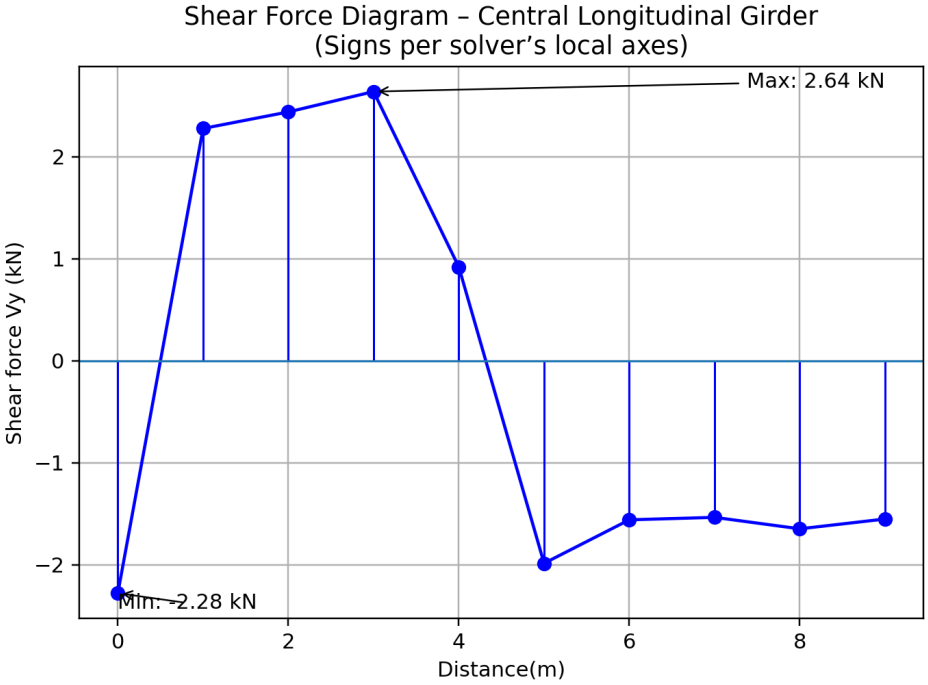


Figure 2.1: Shear Force Diagram (SFD) for the Central Longitudinal Girder

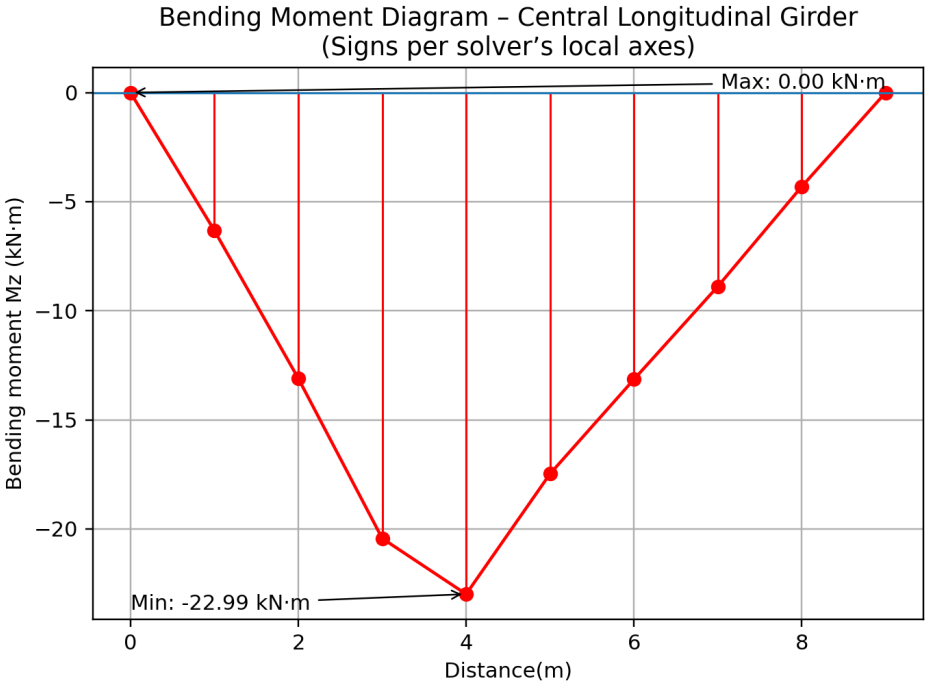


Figure 2.2: Bending Moment Diagram (BMD) for the Central Longitudinal Girder

2.3.2 3D Visualization

The 3D extrusion diagrams provide a spatial context that 2D plots lack. Vertical extrusions along each girder show local internal force magnitudes, where height encodes magnitude scaled to the plan-span. Ribbon surfaces make continuous interpretation easier and expose gradients along the girder length.

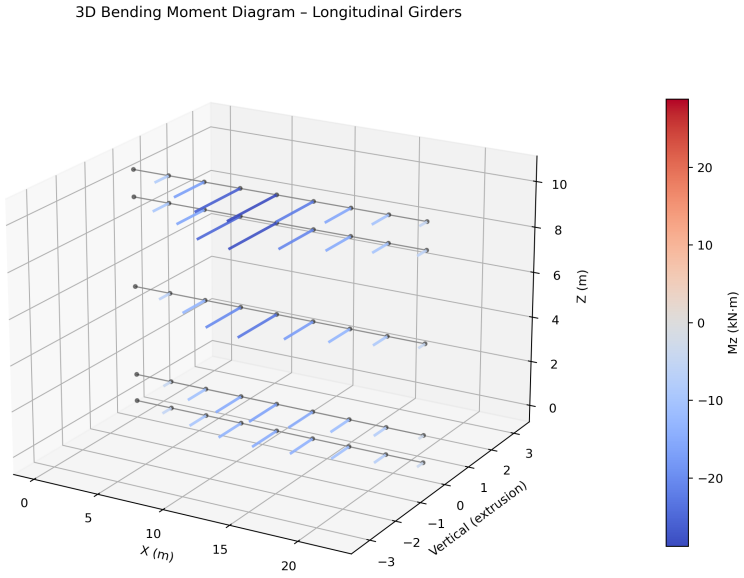


Figure 2.3: 3D Bending Moment Visualization of Longitudinal Girders

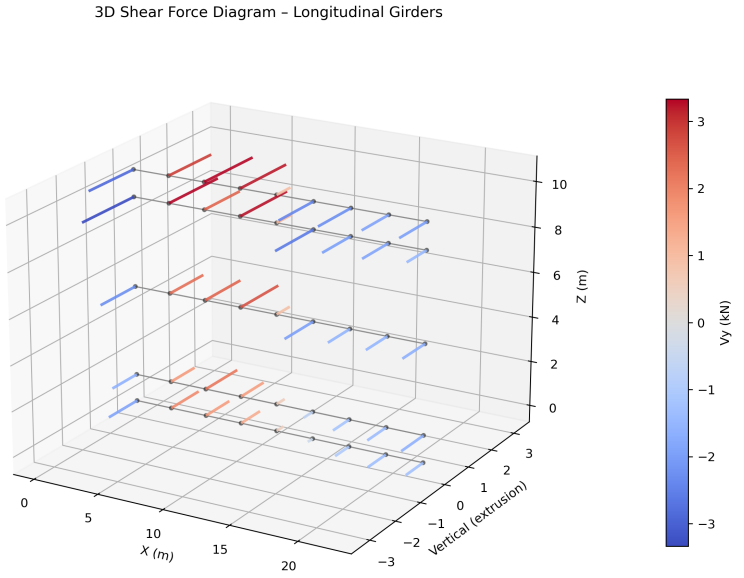


Figure 2.4: 3D Shear Force Visualization of Longitudinal Girders

Chapter 3

Internship Task 1: UI Enhancement, Database Management, and Backend Integration for OsBridgeLCCA

3.1 Task 1: Problem Statement

The primary objective of this task was to develop and enhance the graphical user interface (GUI) and backend data management systems for the OsBridgeLCCA desktop application. The core challenges involved improving the user experience through dynamic input elements, integrating an Excel parser for external data uploads, establishing a robust local database structure for project handling, and implementing real-time state preservation (autosave) mechanisms to prevent data loss across multiple widgets.

3.2 Task 1: Tasks Done

Over the course of the internship, several critical features and optimizations were implemented in the OsBridgeLCCA repository to bridge the frontend UI with backend data processing. The development workflow is highlighted by the following key commits and feature additions:

- **User Interface Improvements & Dynamic Labeling**:
  - Added a new ‘duration of construction‘ input to the financial data form (`Commit`

0cc955b).

  - Replaced outdated terminology by dynamically renaming "Structure Works Data" to "Construction work data" across the left panel widgets (Commit 27fe26c).
  - Reordered the navigation layout, specifically changing the tutorials tab position to the right of the compare tab in main_template.py (Commit e745f22).
  - Updated the custom material popup to include advanced search recommendations and improved window flags, sizing, and title bars (Commits c22a402, e7884cc, 65408f2).

- **Excel Integration and SOR Management**:
  - Implemented an intuitive "Upload Excel" button and integrated the underlying parser logic (Commit 4338598).
  - Added an SOR manager to connect the Excel upload interface seamlessly with the SOR backend (Commit e9269cc).

- **Database Logic & Project Management**:
  - Developed robust database logic to handle user projects and custom materials, specifically creating the user_materials and project database architecture (Commit 1b92e4b).
  - Refined the features for saving to the database and handling recyclability inputs (Commit 8f85635).
  - Transitioned the application to handle comprehensive construction work breakdowns with appropriate alert messages (Commit 8adc017).

- **Real-Time State Preservation & Backend Refactoring**:
  - Engineered and implemented an autosave.json feature to ensure the real-time saving of material data inputs and user progress across all application widgets (Commits b222cac, b4308cc).
  - Optimized backend data handling by improving data_manager.py (Commit 14a301c), removing generated egg-info files (Commit 5719b66), reconfigur-

ing dependencies (`Commit 8fd9ed2`), and standardizing the encoding for carbon emission units (`Commit f241750`).

3.3 Task 1: Python Code

This section presents the Python snippets demonstrating the integration of the enhanced UI components within the OsBridgeLCCA framework, specifically focusing on the dynamic widget generation, UI renaming logic, and the newly added input parameters.

3.3.1 Description of the Scripts

The scripts provided below represent recent updates to the PySide6 application and handle the following operations:

- **Dynamic UI Renaming**: Intercepts legacy system keys (like `KEY_STRUCTURE_WORKS_DATA`) in `ProjectDetailsLeft` and `ProjectDetailsWidget` to safely update user-facing labels to modern terminology ("Construction work data") without breaking backend dependencies.
- **Input Expansion**: Implementation of the new 'Duration of construction' parameter within the PySide6 grid layout, ensuring it correctly passes collected data to the backend storage.

3.3.2 Python Code

Listing 3.1: Dynamic UI Renaming in Left Navigation Panel

```
1 # Excerpt from osbridgelcca/desktop_app/widgets/
      project_details_left_widget.py
2
3 button_data = {
4     KEY_STRUCTURE_WORKS_DATA: [KEY_FOUNDATION, KEY_SUPERSTRUCTURE,
          KEY_SUBSTRUCTURE, KEY_AUXILIARY],
5     KEY_FINANCIAL: [],
6     KEY_CARBON_EMISSION: [KEY_CARBON_EMISSION_COST],
7     KEY_BRIDGE_TRAFFIC: [],
8     KEY_MAINTAINANCE_REPAIR: [],
```

```
    KEY_DEMOLITION_RECYCLE: []
}

for label, sublabels in button_data.items():
    # CHANGE: Check for Structure Works Data and rename it
    display_label = label
    if label == KEY_STRUCTURE_WORKS_DATA:
        display_label = "Construction work data"

    btn = QPushButton(display_label)
    btn.setProperty("class", "category_button")
    btn.setProperty("selected", False)
    btn.setProperty("expanded", False)
    btn.setCursor(Qt.CursorShape.PointingHandCursor)
    btn.setLayoutDirection(Qt.LeftToRight)
    btn.setFocusPolicy(Qt.StrongFocus)
    btn.setIcon(unselected_unexpanded_icon)
    btn.setIconSize(QSize(10, 10))

    # Use display_label for logic so main window receives "Construction
        work data"
    btn.clicked.connect(lambda checked ,b=btn ,name=display_label: self
       .show_structure_widget(name, b))
    scroll_content_layout.addWidget(btn)
    self.all_param_buttons[label] = btn
```

Listing 3.2: Implementation of Duration of Construction Input

```
# Excerpt handling the new financial data input

# 6. Duration of construction
label5 = QLabel("Duration of construction")
label5.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
input5 = QLineEdit()
self.widgets.append(input5)
input5.setAlignment(Qt.AlignmentFlag.AlignLeft)
input5.setFixedWidth(field_width)
input5.setText("")
input5.setStyleSheet(input1.styleSheet())
unit5 = QLabel("(years)")
```

```
13 grid_layout.addWidget(label5, 5, 0, alignment=Qt.AlignVCenter)
14 grid_layout.addWidget(input5, 5, 1, alignment=Qt.AlignVCenter)
15 grid_layout.addWidget(unit5, 5, 2, alignment=Qt.AlignVCenter)
16 grid_layout.addWidget(QWidget(), 5, 3)  # Empty cell for suggested
17
18 # Data Collection Logic
19 def collect_data(self):
20     data = {
21         # ... other keys ...
22         KEY_CONSTR_TIME: 0.0 if not self.widgets[5].text() else float(
               self.widgets[5].text()),
23         KEY_ANALYSIS_PERIOD: 0 if not self.widgets[6].text() else int(
               self.widgets[6].text()),
24     }
25     # Save UI Data to Backend
26     self.database_manager.financial_data = data
27     # calculate Time Cost
28     self.database_manager.calculate_time_cost()
```

3.3.3 Explanation of the Code

- **Listing 3.1, Lines 12-16**: Implements the key UI renaming logic. It intercepts the `KEY_STRUCTURE_WORKS_DATA` constant and dynamically reassigns its `display_label` to "Construction work data" for the user interface, shielding the backend logic from aesthetic changes.

- **Listing 3.1, Lines 28-29**: Binds a lambda function to the button's click signal, passing the dynamically updated `display_label` to ensure proper widget routing.

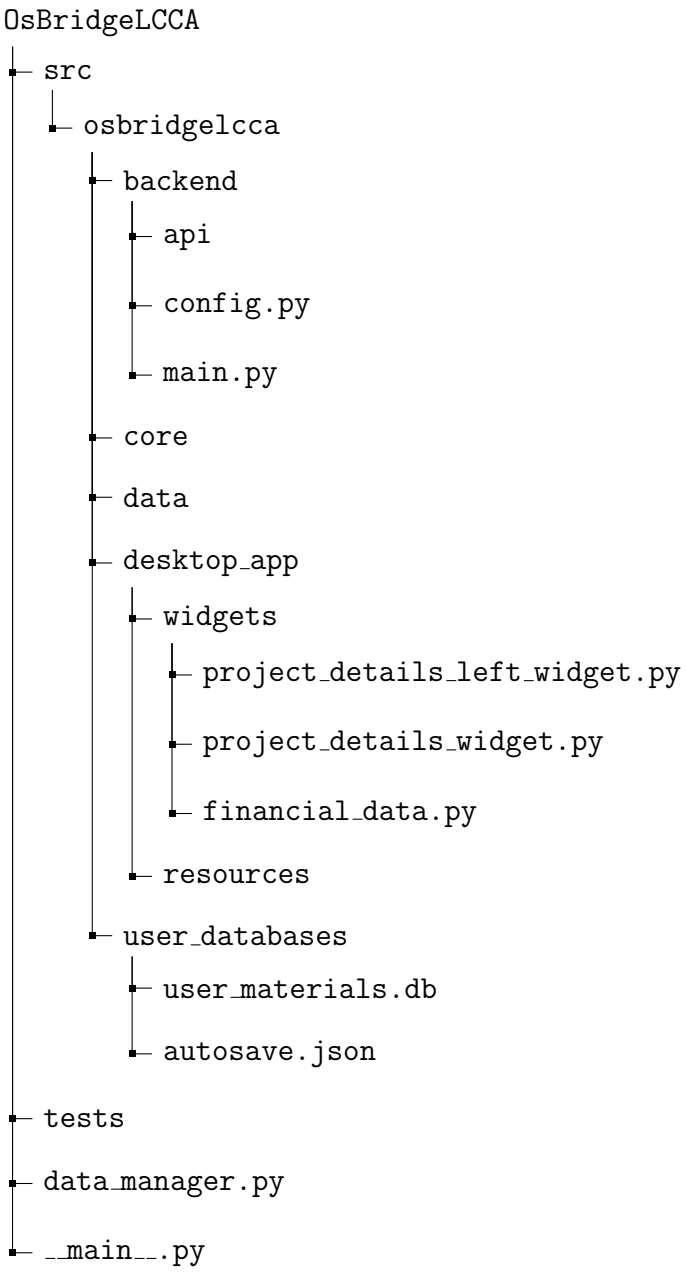
- **Listing 3.2, Lines 3-13**: Instantiates the UI components for the "Duration of construction" parameter, aligning it within the existing `QGridLayout` of the financial data form.

- **Listing 3.2, Lines 16-24**: Demonstrates the updated `collect_data` method, which safely extracts the float value from the new input widget and assigns it to the `KEY_CONSTR_TIME` dictionary key before passing it to the `database_manager`.

3.4 Task 1: Documentation

The structural hierarchy of the repository was updated to accommodate new backend configurations, user databases, and widget designs.

3.4.1 Updated Directory Structure



Program Start Calls The main entry point for the GUI application has been optimized.

To launch the OsBridgeLCCA desktop application, navigate to the project root and execute the following command:

```
$ python -m osbridgelcca.desktop_app.__main__
```

Chapter 4

Internship Task 2: Dynamic GUI Features and Advanced Currency Conversion Logic

4.1 Task 2: Problem Statement

The second phase of the internship focused on enhancing the user experience and ensuring robust data handling within the 3psLCCA (Bridge Life Cycle Cost Analysis) application. The primary challenges addressed were two-fold:

1. **Manual Data Entry Redundancy**: Users were required to manually select their currency after choosing a country during project creation, increasing the likelihood of input errors and reducing UI efficiency.

2. **State Management and Conversion Logic Bugs**: The social cost module suffered from state-management issues where nested signal suppression scopes were overriding each other. Furthermore, the UI lacked dynamic adaptability to hide or show specific currency conversion fields (such as USD or INR) based on the globally synced project currency.

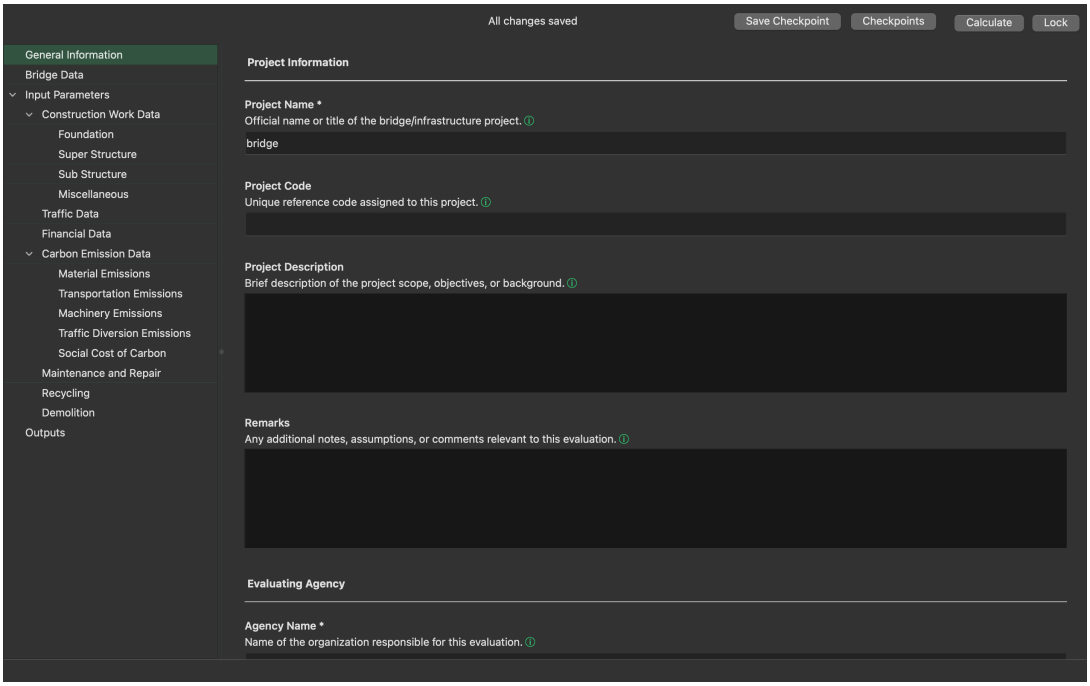


Figure 4.1: General Information View in 3psLCCA after project initialization.

4.2 Task 2: Tasks Done

To resolve these issues, two major features were developed and successfully merged into the repository (PR #7 and PR #11).

4.2.1 Auto-Filling Currency based on Country Selection (PR #7)

A new dynamic feature was implemented in the `new_project_dialog.py` component.

- **Data Mapping**: Updated the `countries_data.py` utility to construct a `COUNTRY_TO_CURRENCY` dictionary, mapping country names directly to their standard currency codes.
- **Signal Integration**: Connected the `currentTextChanged` signal of the country input dropdown to a new handler method.
- **Logic Execution**: Developed the `_auto_fill_currency` method, which intercepts the selected country, fetches the corresponding currency, and automatically updates the `currency_input` index.

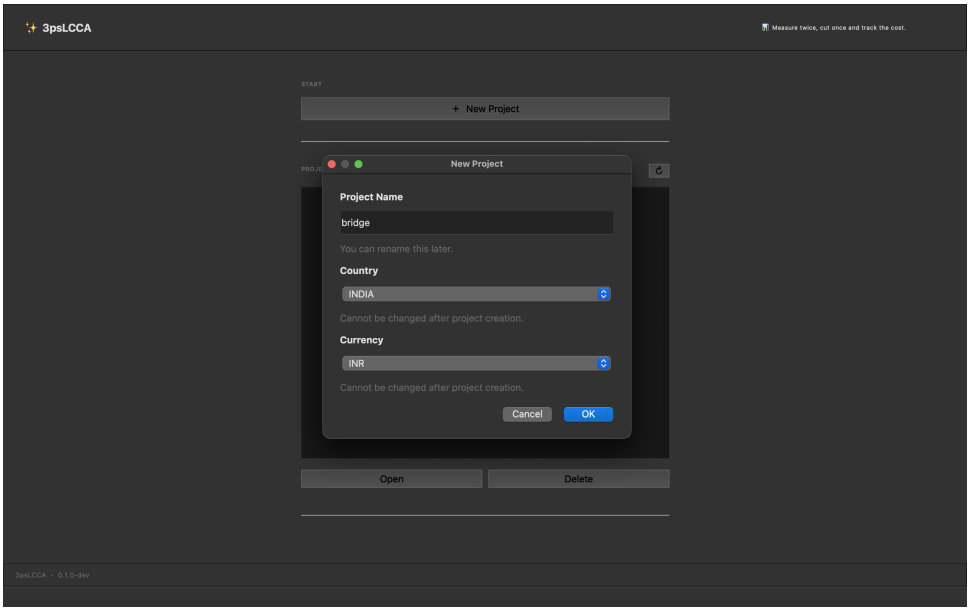


Figure 4.2: New Project Dialog demonstrating the auto-filling currency feature based on Country selection.

4.2.2 Social Cost Currency Conversion and Signal Suppression (PR #11)

Significant backend logic and UI layout handlers were introduced to the `social_cost.py` widget to stabilize currency conversions for carbon emissions.

- **Robust Signal Suppression**: Replaced a simple boolean suppression flag with a counter-based property (`self.__suppress`). This ensured that nested suppression scopes during UI updates did not clobber each other, maintaining the integrity of the `_suppress_signals` check.
- **Recursive Layout Scanning**: Implemented the `_find_row_for_widget` helper function. This function safely and recursively locates the `QFormLayout` row index containing a specific widget, even if deeply nested.
- **Dynamic UI Toggling**: Developed `_toggle_currency_rows()` to utilize the located row indices and dynamically hide the INR or USD conversion input fields if the project's global currency already matches them.

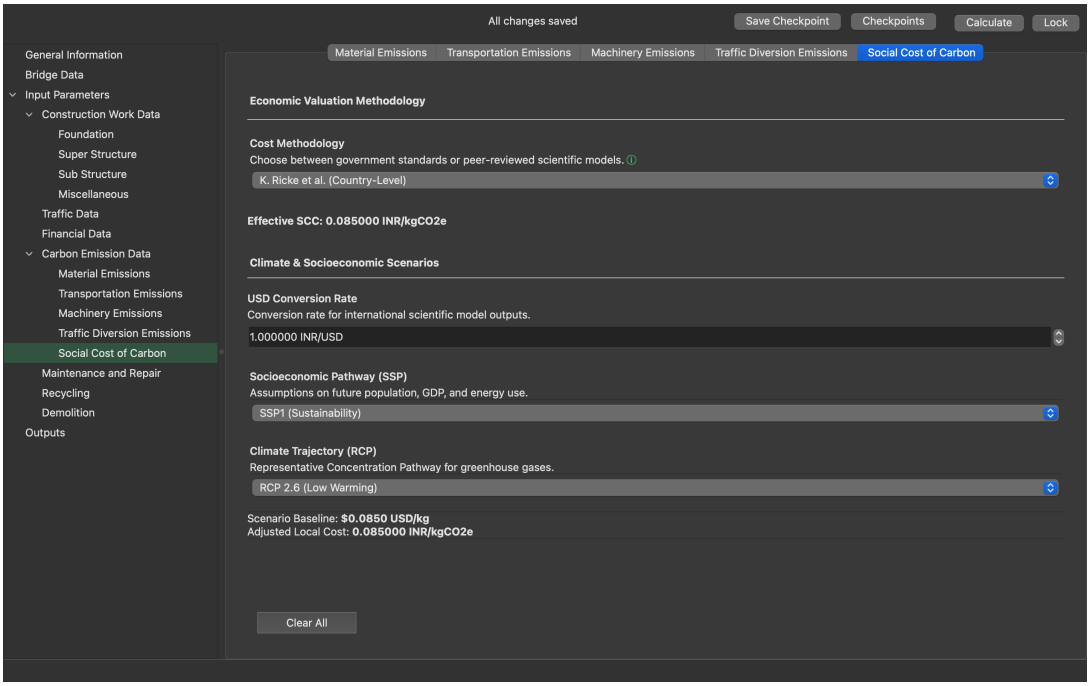


Figure 4.3: Social Cost of Carbon evaluation tab demonstrating dynamic currency layout adjustments.

4.3 Task 2: Python Code

This section presents the Python scripts developed to resolve the aforementioned issues.

4.3.1 Auto-Fill Currency Logic

The following snippet demonstrates the logic integrated into the new project dialog to map and auto-select the currency.

Listing 4.1: Auto-Filling Currency Feature (PR #7)

```
1  # gui/components/new_project_dialog.py
2
3  # 1. Connected the signal in the setup UI phase
4  self.country_input.currentTextChanged.connect(self._auto_fill_currency)
5
6  # 2. Implemented the auto-fill logic
7  def _auto_fill_currency(self, selected_country: str):
8      """Auto-fills the currency combo box based on the country selection
       ."""
9      if selected_country in COUNTRY_TO_CURRENCY:
10         target_currency = COUNTRY_TO_CURRENCY[selected_country]
```

```
11
12          idx = self.currency_input.findText(target_currency)
13          if idx >= 0:
14              self.currency_input.setCurrentIndex(idx)
15      else:
16          self.currency_input.setCurrentIndex(0)
```

4.3.2 Social Cost and Signal Suppression Logic

The script below showcases the implementation of the advanced signal suppression mechanism and recursive layout location.

Listing 4.2: Social Cost Currency Conversion & UI Tracking (PR #11)

```
1  # gui/components/carbon_emission/widgets/social_cost.py
2
3      #         Suppression
4      # Simple counter so nested suppression scopes dont clobber each
           other.
5      # Stored as int; the bool property keeps all `if self.
           _suppress_signals:` checks working.
6
7      @property
8      def _suppress_signals(self):
9          return self.__suppress > 0
10
11     @_suppress_signals.setter
12     def _suppress_signals(self, val):
13         # Accept True/False as increment/decrement so existing call-
               sites work unchanged.
14         if val:
15             self.__suppress += 1
16         else:
17             self.__suppress = max(0, self.__suppress - 1)
18
19     #         Recursive Layout Locator
```

```
20
21      def _find_row_for_widget(self, layout, target_widget):
22          """Helper to safely and recursively locate the QFormLayout row
                index containing a specific widget."""
23          def _contains(item, target):
24              if not item:
25                  return False
26              # Check direct widget match
27              if item.widget() == target:
28                  return True
29              # Check if it's deeply nested inside a wrapper widget
30              if item.widget() and item.widget().isAncestorOf(target):
31                  return True
32              # Check if it's nested inside a sub-layout
33              lay = item.layout()
34              if lay:
35                  for j in range(lay.count()):
36                      if _contains(lay.itemAt(j), target):
37                          return True
38              return False
39
40          # Scan every row in the form layout
41          for i in range(layout.rowCount()):
42              for role in (QFormLayout.FieldRole, QFormLayout.
                    SpanningRole, QFormLayout.LabelRole):
43                  if _contains(layout.itemAt(i, role), target_widget):
44                      return i
45          return -1  # Not found
46
47      #         Dynamic Row Toggling
48
49      def _toggle_currency_rows(self):
50          cur = str(self._project_currency or "").strip().upper()
51
52          # Hide INR Conversion input field if current project currency
                is INR
```

```
53          if hasattr(self, '_inr_row') and self._inr_row >= 0:
54              self._niti_layout.setRowVisible(self._inr_row, cur != "INR"
                    )
55
56          # Hide USD Conversion input field if current project currency
                is USD
57          if hasattr(self, '_usd_row') and self._usd_row >= 0:
58              self._ricke_layout.setRowVisible(self._usd_row, cur != "USD
                    ")
```

4.4 Task 2: Documentation

The directory structure below outlines where the key modifications for these tasks reside within the 3psLCCA graphical user interface repository.

4.4.1 Target Directory Structure

```
3psLCCA-gui
└─ gui
   └─ components
      ├─ carbon_emission
      │  └─ widgets
      │     └─ social_cost.py
      ├─ utils
      │  └─ countries_data.py
      └─ new_project_dialog.py
```

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the fellowship, significant contributions were made to the development and stabilization of the OsBridgeLCCA (3psLCCA) desktop application. The primary focus was on bridging the gap between a highly functional backend engine and an intuitive, user-friendly graphical interface. The key tasks successfully accomplished include:

- **Comprehensive UI/UX Modernization**: Redesigned critical input workflows, including the implementation of dynamic labeling (e.g., transitioning "Structure Works Data" to "Construction work data" seamlessly) and the addition of necessary parameters like construction duration, improving overall user clarity.
- **Advanced State Management**: Engineered a robust, real-time state preservation system utilizing `autosave.json`, preventing data loss across complex, multi-tabbed widgets.
- **Intelligent Automation**: Implemented automated UI behaviors, notably the auto-filling of global currencies based on country selection, which significantly reduced manual entry redundancy and potential user error.
- **Complex Data Integration**: Successfully integrated an Excel parser and Schedule of Rates (SOR) manager, alongside establishing a localized database architecture (`user_materials.db`) for custom material handling.

- **Bug Resolution and Backend Optimization**: Resolved intricate state-management conflicts, specifically addressing nested signal suppression scopes in the social cost module. Furthermore, implemented recursive layout locators to dynamically toggle UI elements without breaking existing form structures.

5.2 Skills Developed

The scope and complexity of this project provided a rigorous environment for both technical and professional growth. The fellowship facilitated the development of the following key skills:

- **Advanced GUI Frameworks (PySide6/Qt)**: Gained deep, practical experience in building dynamic desktop applications. Mastered complex concepts such as signal-slot architecture, dynamic widget generation, recursive layout traversal, and event suppression logic.
- **Python Software Engineering**: Improved proficiency in writing clean, modular, and maintainable Python code. Developed skills in separating frontend views from backend data logic and handling dictionary-based mapping for UI automation.
- **Data Persistence & Database Management**: Learned to effectively manage application state using localized JSON structures and SQLite databases, ensuring seamless data preservation during application runtime.
- **Version Control and Collaborative Development**: Enhanced Git/GitHub proficiency through active repository management. Gained experience in isolating features into focused commits, opening structured pull requests, and navigating collaborative open-source workflows.
- **Algorithmic Problem Solving**: Developed robust debugging techniques, exemplified by engineering custom recursive functions to locate and manipulate deeply nested layout items that standard framework methods could not easily reach.
- **User-Centric Design Thinking**: Cultivated an understanding of how technical backend requirements must be translated into frictionless frontend experiences, keeping the end-user's workflow in mind at every stage of development.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:	Jawwad Ahmad		
Project:	OsBridgeLCCA & 3psLCCA		
Internship:	FOSSEE Winter Internship 2026		
DATE	**DAY**	**TASK**	**Hours Worked**
02-Feb-2026	Monday	Meeting 01: Discussed UI enhancements and backend integration scope for OsBridgeLCCA desktop application.	2.0
05-Feb-2026	Thursday	Added a new `duration of construction` input to the financial data form (Commit `0cc955b`).	3.5
10-Feb-2026	Tuesday	Replaced outdated terminology by dynamically renaming ”Structure Works Data” to ”Construction work data” across left panel widgets (Commit `27fe26c`).	4.0
14-Feb-2026	Saturday	Reordered navigation layout, moving the tutorials tab to the right of the compare tab in `main_template.py` (Commit `e745f22`).	2.5
18-Feb-2026	Wednesday	Updated custom material popup with advanced search recommendations, improved window flags, and sizing (Commits `c22a402`, `e7884cc`, `65408f2`).	5.0
24-Feb-2026	Tuesday	Began Excel integration logic. Implemented an intuitive ”Upload Excel” button and underlying parser logic (Commit `4338598`).	4.5
27-Feb-2026	Friday	Developed an SOR manager to seamlessly connect the Excel upload interface with the SOR backend (Commit `e9269cc`).	4.0
04-Mar-2026	Wednesday	Created local database logic to handle user projects; engineered the `user_materials` and project database architecture (Commit `1b92e4b`).	5.0
09-Mar-2026	Monday	Refined features for saving to the database, particularly handling recyclability inputs (Commit `8f85635`).	3.5
14-Mar-2026	Saturday	Transitioned the application to handle comprehensive construction work breakdowns with appropriate alert messages (Commit `8adc017`).	4.0

DATE	DAY	TASK	Hours Worked
20-Mar-2026	Friday	Engineered `autosave.json` feature for real-time state preservation of material data across widgets (Commits `b222cac`, `b4308cc`).	6.0
26-Mar-2026	Thursday	Backend refactoring: optimized `data_manager.py`, removed generated egg-info, and reconfigured dependencies (Commits `14a301c`, `5719b66`, `8fd9ed2`).	4.5
30-Mar-2026	Monday	Standardized the encoding for carbon emission units in the backend (Commit `f241750`).	2.0
03-Apr-2026	Friday	Meeting 02: Transitioned to 3psLCCA repository. Discussed manual data entry redundancy issues during project creation.	2.0
08-Apr-2026	Wednesday	Updated `countries_data.py` utility to construct a `COUNTRY_TO_CURRENCY` dictionary for data mapping.	3.0
15-Apr-2026	Wednesday	Connected the `currentTextChanged` signal of the country input dropdown to a new handler method in `new_project_dialog.py`.	3.5
20-Apr-2026	Monday	Developed the `_auto_fill_currency` method to automatically update the currency input index based on country selection.	4.0
24-Apr-2026	Friday	Finalized testing for the auto-fill currency logic. Opened and successfully merged PR #7.	2.5
02-May-2026	Saturday	Investigated state-management bugs in the social cost module where nested signal suppression scopes were overriding each other.	4.0
07-May-2026	Thursday	Developed robust signal suppression in `social_cost.py` by replacing the boolean flag with a counter-based property (`self.__suppress`).	5.0
14-May-2026	Thursday	Implemented the `_find_row_for_widget` helper function to recursively locate deeply nested `QFormLayout` row indices.	4.5
21-May-2026	Thursday	Developed `_toggle_currency_rows()` to dynamically hide INR or USD conversion inputs if they match the global project currency.	5.0
27-May-2026	Wednesday	Finalized testing and merged PR #11 for Social Cost Currency Conversion.	2.5

DATE	DAY	TASK	Hours Worked
30-May-2026	Saturday	Compiled final directory structure documentation and drafted the Internship Work Report.	3.0

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.