



FOSSEE Winter Internship Report

On

Importing XLSX to the LCC App for Osdag

Submitted by

Ritik Kumar

3rd Year B.Tech Student, Department of Computer Science

Vellore Institute Of Technology

Bhopal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Parth Karia

Swastik Gupta

February 20, 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, Parth Karia and Swastik Gupta,
- 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

1	Introduction	4
1.1	National Mission in Education through ICT	4
1.1.1	ICT Initiatives of MoE	5
1.2	FOSSEE Project	6
1.2.1	Projects and Activities	6
1.2.2	Fellowships	6
1.3	Osdag Software	7
1.3.1	Osdag GUI	8
1.3.2	Features	8
2	Screening Task	9
2.1	Problem Statement	9
2.2	Tasks Done	9
3	Internship Task 1 Title	11
3.1	Task 1: Problem Statement	11
3.2	Task 1: Tasks Done	11
3.2.1	Creation of Excel Template for LCCA Application	11
3.3	Task 1: Python Code	13
3.4	Excel Sheet Designs	13
4	Internship Task 2 Title	14
4.1	Task 2: Problem Statement	14
4.2	Task 2: Tasks Done	14
4.2.1	Creation of Excel Data Parser for LCCA Application	14
4.3	Task 2: Documentation	16
4.3.1	Creation of Excel Data Parser for LCCA Application	16
4.3.2	Python Code	17
4.3.3	Explanation of the Code	24
4.4	Validation Report	26
4.5	Validation Report Documentation	28

5	Internship Task 3	29
5.1	Task 3 Title	29
5.2	Tasks Done	29
5.2.1	Creation of Excel Data Parser for LCCA Application	29
5.3	Task 3: Documentation	30
5.3.1	Overview of Excel Parser Integration in LCCA	30
5.3.2	Excel Reading and Multi-Sheet Handling	30
5.3.3	Header Normalization and Field Mapping	31
5.3.4	Validation Mechanism	31
5.3.5	Duplicate Detection Logic	32
5.3.6	Data Persistence and JSON Storage	32
5.3.7	Cost Calculation from Parsed Data	32
5.3.8	UI Distribution and Widget Mapping	33
5.3.9	Construction Locking Mechanism	33
5.3.10	Conclusion	33
5.3.11	Python Code	34
5.4	Code Explanation	64
5.4.1	Entry Point – <code>open_excel_dialog()</code>	64
5.4.2	Core Parsing Function – <code>parse_and_validate_excel(file_path)</code>	64
5.4.3	Header Normalization – <code>clean_header()</code>	65
5.4.4	Validation Mechanism	66
5.4.5	Duplicate Detection Logic	66
5.4.6	Data Storage and JSON Generation	67
5.4.7	Cost Computation – <code>calculate_cost_from_json()</code>	67
5.4.8	UI Mapping – <code>distribute_imported_data()</code>	67
5.4.9	Construction Locking – <code>calculate_and_lock_construction_data()</code>	68
5.4.10	Summary	68
A	Appendix	69
A.1	Work Reports	69
	Bibliography	73

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	SWAYAMPBABHA	Access 24x7 TV programs	Enable SWAYAMPBABHA 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.

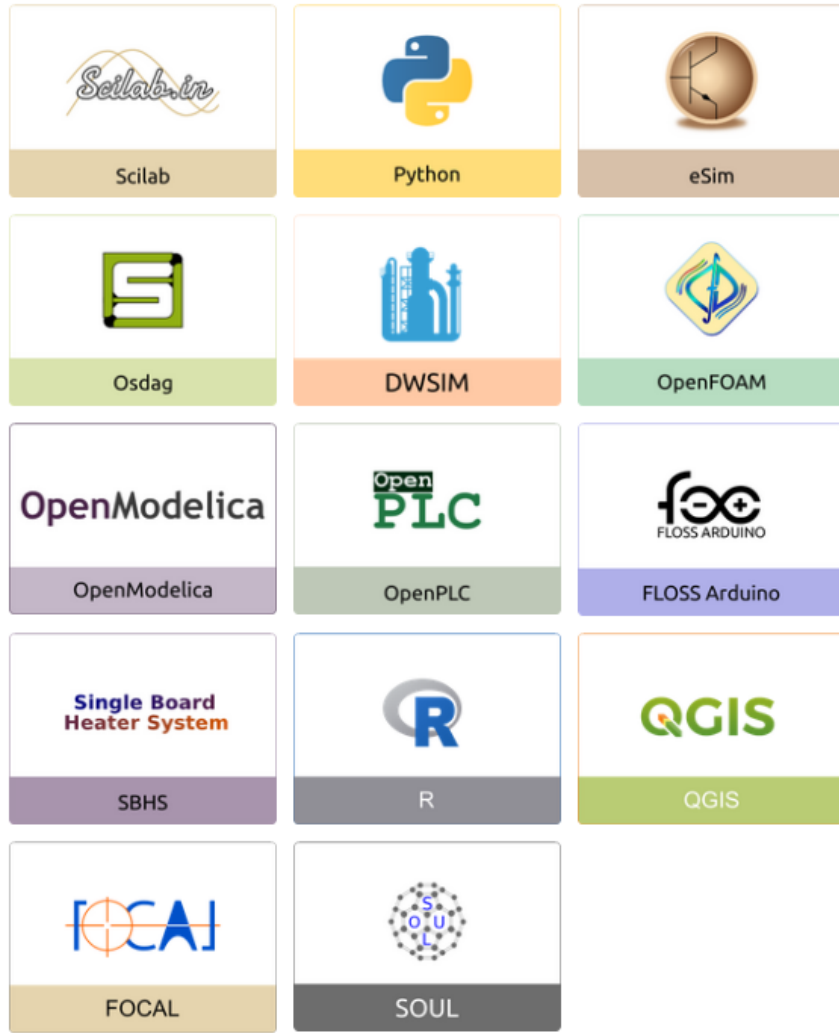


Figure 1.1: FOSSEE Projects and Activities

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.

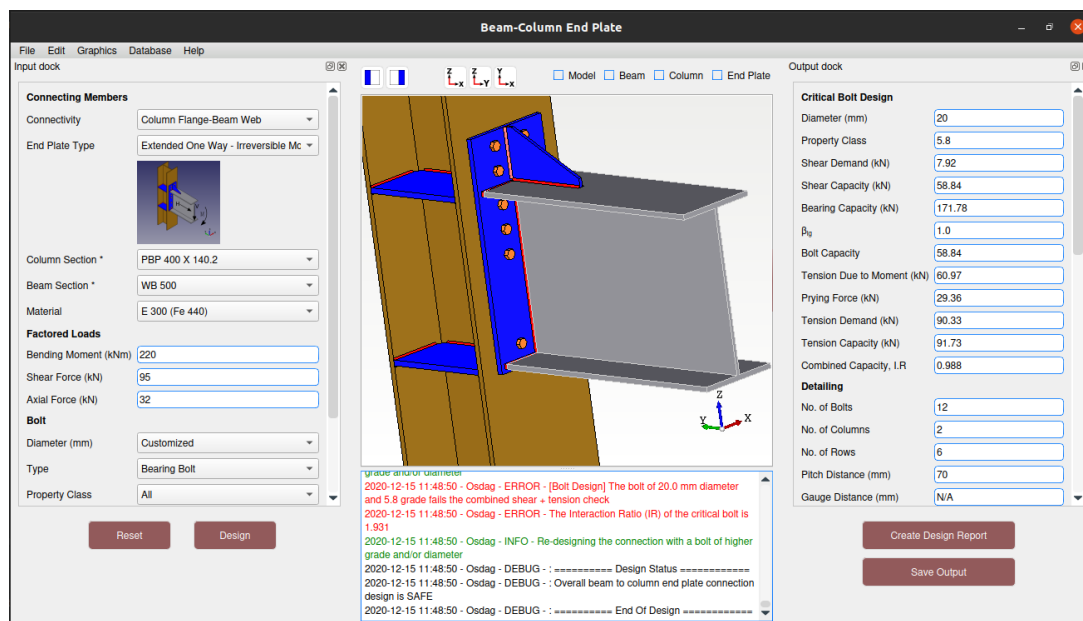


Figure 1.2: Osdag GUI

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

1. Create a shear force (SFD) and bending moment diagram (BMD) from the values provided in the Xarray dataset using Plotly/PyPlot for the central longitudinal girder.
2. Generate 3D shear force and bending moment diagrams (SFD and BMD) for all the girders using Plotly/PyPlot, similar to the visualization style in MIDAS

2.2 Tasks Done

In this task, shear force and bending moment values were extracted from the given Xarray dataset for the central longitudinal girder. The dataset contained structural analysis results organized with respect to longitudinal position and girder identification.

First, the data corresponding to the central girder was filtered from the dataset. The longitudinal coordinates along the span were used as the X-axis values. The corresponding shear force and bending moment values were obtained as functions of span length.

Using PyPlot/Plotly, two separate 2D plots were generated:

1. Shear Force Diagram (SFD): The longitudinal distance was plotted on the horizontal axis, and the shear force values were plotted on the vertical axis. The diagram clearly represents the variation of shear force along the girder span. Positive and negative shear regions were visualized to identify support reactions and critical sections.

2. Bending Moment Diagram (BMD): Similarly, bending moment values were plotted against the longitudinal position. The diagram shows the distribution of sagging and

hogging moments along the girder. The maximum bending moment and its location were identified from the graph.

Appropriate labels, grid lines, legends, and titles were added to ensure clarity and proper interpretation of results.

To visualize the force distribution across the entire bridge deck, three-dimensional surface plots were generated for all longitudinal girders.

In the 3D representation:

The X-axis represents the longitudinal position along the span.

The Y-axis represents the girder number (transverse direction).

The Z-axis represents either shear force or bending moment values.

A meshgrid was created using the longitudinal coordinates and girder indices. The corresponding shear force and bending moment matrices were used as the surface height values (Z-axis).

Using Plotly/PyPlot surface plotting functions, 3D surface diagrams were generated. Color mapping was applied to enhance visualization and clearly indicate regions of high and low force intensity. This visualization style is similar to that used in structural analysis software such as MIDAS Civil.

The 3D diagrams allow clear observation of:

Distribution of shear forces across girders

Distribution of bending moments across girders

Identification of critical girders

Maximum force regions in both longitudinal and transverse directions

Chapter 3

Internship Task 1 Title

3.1 Task 1: Problem Statement

Define the problem for the first task undertaken.

3.2 Task 1: Tasks Done

3.2.1 Creation of Excel Template for LCCA Application

An Excel template was developed to enable structured data entry for the Life Cycle Cost Analysis (LCCA) application. The objective was to design a user-friendly format in which users can input construction, cost, and environmental parameters in a standardized manner for direct import into the application.

Excel File Structure

The Excel sheet was designed in an item-based format, where each row represents a construction activity. The columns were structured to align with the internal data mapping of the LCCA system.

The following fields were included:

- SOR Item No.
- Material Description
- Quantity (Unit_A)

- Unit_A
- Rupees per Unit_A
- Rate Source
- Carbon Emission (kgCO₂e / Unit_B)
- Carbon Emission Units
- Conversion Factor (Unit_A → Unit_B)
- Carbon Factor Source
- % Recyclability
- Scrap Rate

Data Formatting and Standardization

To ensure compatibility with the LCCA application:

- Numeric formatting was applied to quantity, rate, and carbon emission values.
- Units were clearly specified to avoid ambiguity.
- Conversion factors were incorporated to handle differences between cost units and carbon emission units.
- Percentage fields such as recyclability and scrap rate were defined in numeric format.

System Integration Considerations

The column arrangement was aligned with the application's internal CID-based mapping system. This ensures that upon importing the Excel file, each parameter is automatically recognized and mapped correctly within the LCCA framework.

Outcome

The developed Excel template provides a standardized and structured format for entering construction, financial, and environmental data. It ensures clean data entry, reduces errors, and enables seamless integration with the LCCA application.

3.3 Task 1: Python Code

So this task was Completely designing based so in this i have focussed on the no code part development i.e only on Designing of the Excel Sheet.

3.4 Excel Sheet Designs

Excavation									
SOR Item No.	Material	Quantity (Unit A)	Unit A	Rupees/Unit A	Rate source	Carbon emission (kgCO ₂ e/Unit B)	Carbon emission units	Conversion factor (Unit A * Unit B)	Carbon factor
CIDItemNo	CIDItemName	CIDQuantity	CIDUnit	CIDRate	CIDRate_Src	CIDCarbon_Emission	CIDCarbon_Emission_units	CIDConversion_Factor	CIDCarbon_Factor
12.02	All type(manual) (0 to 1.5m)	500	cum	239	Maha PWD SOR	not_available	not_available	not_available	not_available
12.02	All type(manual) (1.5 to 3.0m)	213	cum	262.5	Maha PWD SOR	not_available	not_available	not_available	not_available
12.02	All type(manual) (3.0 to 4.5m)		cum	286.8	Maha PWD SOR	not_available	not_available	not_available	not_available
12.02	All type(manual) (4.5 to 6)		cum	310.7	Maha PWD SOR	not_available	not_available	not_available	not_available
12.02	All type(manual) (Above 6 m)		cum	310.7	Maha PWD SOR	not_available	not_available	not_available	not_available
12.03	All type(Mechanical) (0 to 1.5m)		cum	90	Maha PWD SOR	not_available	not_available	not_available	not_available
12.03	All type(Mechanical) (1.5 to 3.0m)		cum	99	Maha PWD SOR	not_available	not_available	not_available	not_available
12.03	All type(Mechanical) (3.0 to 4.5m)		cum	108	Maha PWD SOR	not_available	not_available	not_available	not_available
12.03	All type(Mechanical) (4.5 to 6)		cum	117	Maha PWD SOR	not_available	not_available	not_available	not_available
12.03	All type(Mechanical) (Above 6 m)		cum	117	Maha PWD SOR	not_available	not_available	not_available	not_available
12.04	Hard Murrum (0 to 1.5m)		cum	241	Maha PWD SOR	not_available	not_available	not_available	not_available
12.04	Hard Murrum (1.5 to 3.0m)		cum	265.1	Maha PWD SOR	not_available	not_available	not_available	not_available
12.04	Hard Murrum (3.0 to 4.5m)		cum	289.2	Maha PWD SOR	not_available	not_available	not_available	not_available
12.04	Hard Murrum (4.5 to 6)		cum	313.3	Maha PWD SOR	not_available	not_available	not_available	not_available
12.04	Hard Murrum (Above 6 m)		cum	313.3	Maha PWD SOR	not_available	not_available	not_available	not_available
12.06	Hard Murrum and Boulders (0 to 1.5m)		cum	286	Maha PWD SOR	not_available	not_available	not_available	not_available
12.06	Hard Murrum and Boulders (1.5 to 3.0m)		cum	314.6	Maha PWD SOR	not_available	not_available	not_available	not_available
12.06	Hard Murrum and Boulders (3.0 to 4.5m)		cum	343.2	Maha PWD SOR	not_available	not_available	not_available	not_available
12.06	Hard Murrum and Boulders (4.5 to 6)		cum	371.8	Maha PWD SOR	not_available	not_available	not_available	not_available
12.06	Hard Murrum and Boulders (Above 6 m)		cum	371.8	Maha PWD SOR	not_available	not_available	not_available	not_available
12.08	Soft Rock (0 to 1.5m)		cum	67	Maha PWD SOR	not_available	not_available	not_available	not_available
12.08	Soft Rock (1.5 to 3.0m)		cum	612.7	Maha PWD SOR	not_available	not_available	not_available	not_available
12.08	Soft Rock (3.0 to 4.5m)		cum	668.4	Maha PWD SOR	not_available	not_available	not_available	not_available
12.08	Soft Rock (4.5 to 6)		cum	724.1	Maha PWD SOR	not_available	not_available	not_available	not_available
12.08	Soft Rock (Above 6 m)		cum	724.1	Maha PWD SOR	not_available	not_available	not_available	not_available
12.09	Hard Rock (Blasting) (0 to 1.5m)		cum	1232	Maha PWD SOR	not_available	not_available	not_available	not_available
12.09	Hard Rock (Blasting) (1.5 to 3.0m)		cum	1355.2	Maha PWD SOR	not_available	not_available	not_available	not_available

Figure 3.1: Excel Template Designed for LCCA Application Data Entry

Chapter 4

Internship Task 2 Title

4.1 Task 2: Problem Statement

Creation of a Parser which parses the data present in the excel sheet

4.2 Task 2: Tasks Done

4.2.1 Creation of Excel Data Parser for LCCA Application

A data parser was developed to extract and process structured information from the Excel template designed for the LCCA application. The purpose of the parser is to read user-entered data from the Excel sheet and convert it into a structured format compatible with the internal data model of the application.

Objective

The primary objective of developing the parser was to:

- Automatically read item-wise construction data from the Excel file.
- Validate numerical and textual inputs.
- Map Excel columns to corresponding LCCA system parameters.
- Prepare structured data for further cost and carbon analysis.

Data Extraction Process

The parser reads the Excel sheet row by row. Each row corresponds to a construction activity and contains cost, quantity, carbon emission, and recyclability parameters.

The following steps were performed during parsing:

- Identification of required columns based on predefined headers.
- Extraction of numeric fields such as quantity, rate, carbon emission, conversion factor, and scrap rate.
- Extraction of descriptive fields such as SOR Item Number, material description, and source references.
- Conversion of percentage values into numerical format for computation.

Data Validation

Basic validation checks were incorporated to ensure:

- Mandatory fields are not empty.
- Numeric fields contain valid numerical values.
- Units are consistent with expected format.
- Conversion factors are properly defined where required.

Invalid or incomplete entries are flagged before further processing to prevent incorrect LCCA calculations.

Data Mapping and Structuring

After extraction and validation, the parsed data is mapped to the internal data structure of the LCCA application. Each Excel column is associated with a corresponding system parameter, ensuring seamless integration with cost computation and carbon emission analysis modules.

Outcome

The developed parser enables automated processing of user-uploaded Excel files. It eliminates manual data entry into the application, reduces errors, and ensures that structured, validated data is available for life cycle cost and environmental impact assessment.

4.3 Task 2: Documentation

4.3.1 Creation of Excel Data Parser for LCCA Application

Task Objective

The objective of this task was to develop a parser capable of reading and processing structured data from the Excel template designed for the LCCA application. The parser enables automatic extraction of user-entered data and prepares it for further life cycle cost and carbon emission analysis.

Description of Work Performed

A parsing module was implemented to read the Excel sheet containing item-wise construction data. The parser systematically processes each row of the sheet, where every row represents a construction activity along with its financial and environmental parameters.

The parser performs the following operations:

- Reads the Excel file and identifies predefined column headers.
- Extracts numerical values such as quantity, rate per unit, carbon emission factor, conversion factor, recyclability percentage, and scrap rate.
- Extracts descriptive fields such as SOR item number, material description, and data sources.
- Converts percentage and unit-based values into standardized numerical formats for computation.

Data Validation and Error Handling

To ensure reliability and accuracy, validation checks were incorporated:

- Verification of mandatory fields to prevent missing data.
- Validation of numeric entries in cost and emission-related columns.
- Consistency checks for units and conversion factors.
- Identification and flagging of incomplete or invalid records before further processing.

These validation steps help prevent incorrect inputs from affecting LCCA computations.

Data Structuring and Integration

After successful extraction and validation, the parsed data is structured according to the internal data model of the LCCA application. Each Excel column is mapped to its corresponding system parameter, ensuring seamless integration with cost analysis and carbon emission calculation modules.

Outcome

The developed parser automates the process of importing Excel-based user inputs into the LCCA application. It reduces manual effort, minimizes data entry errors, and ensures that structured and validated data is available for life cycle cost assessment.

4.3.2 Python Code

The Python script is shown below. Each section is commented for clarity.

Listing 4.1: Parser Code which Parses the Excel sheet

```

1 %-----begin code-----
2
3 import pandas as pd
4 import json
5 import numpy as np
6
7 # CONFIGURATION & VALIDATION RULES
8
9 ALLOWED_UNITS = {'cum', 'rmt', 'm2', 'mt', 'nos', 'kg'}
10 FLEXIBLE_FIELD_CONFIG = ['carbon_emission', 'conversion_factor']

```

```

11 ALLOWED_FLEXIBLE_VALUES = {'not_available'}
12
13 # HELPER FUNCTIONS
14
15 def clean_header(header_val):
16     if not isinstance(header_val, str):
17         return str(header_val)
18     # Converts 'CID#Item' -> 'item'
19     return header_val.replace('CID#', '').lower().strip()
20
21 def is_valid_number(val):
22     if val is None or val == "":
23         return False
24     try:
25         float(val)
26         return True
27     except (ValueError, TypeError):
28         return False
29
30 def get_row_identifier(sheet, type_name, row_idx, item_code=None):
31     # Pandas index 0 = Excel Row 1
32     base_id = f"Sheet: '{sheet}', Component: '{type_name}', Row: {
33         row_idx + 1}"
34
35     # Optional: Use item code for easier debugging if it exists,
36     # but this is not a validation rule.
37     if item_code:
38         return f"{base_id} [Item: {item_code}]"
39     return base_id
40
41 # MAIN PARSER
42
43 def parse_and_validate_excel(file_path):
44     try:
45         all_sheets = pd.read_excel(file_path, sheet_name=None, header=
46             None)
47     except Exception as e:
48         return {"fatal_error": f"Failed to read file: {str(e)}"}

```

```

48
49     parsed_sections = []
50     global_errors = []
51     global_warnings = []
52
53     for sheet_name, df in all_sheets.items():
54
55         current_section = None
56         current_keys = None
57
58         # New Tracker: Maps 'material_name' -> { 'data': row_dict, 'row_id': identifier }
59         section_seen_data = {}
60
61         for idx, row in df.iterrows():
62             if row.isnull().all():
63                 continue
64
65             first_val = str(row[0]).strip() if pd.notna(row[0]) else ""
66
67             # --- 1. Identify Header (CID#) ---
68             if first_val.startswith('CID#'):
69                 current_keys = []
70                 for i, val in enumerate(row):
71                     if pd.notna(val) and str(val).strip() != "":
72                         # Dynamic extraction: 'CID#Item' becomes 'item'
73                         current_keys.append((i, clean_header(str(val).strip())))
74                 continue
75
76             # --- Skip Visual Headers ---
77             # 'SOR Item No.' is usually the label above 'CID#Item', we skip it to avoid parsing it as data
78             if first_val in ['Material', 'SOR Item No.']:
79                 continue
80
81             # --- 2. Identify Section ---
82             col2 = row[2] if len(row) > 2 else np.nan
83             col3 = row[3] if len(row) > 3 else np.nan

```

```

84
85     if pd.notna(row[0]) and pd.isna(col2) and pd.isna(col3):
86         if current_section:
87             parsed_sections.append(current_section)
88
89         current_section = {
90             "sheetName": sheet_name,
91             "type": first_val,
92             "data": []
93         }
94         # Reset duplicate tracker for new component
95         section_seen_data = {}
96         continue
97
98     # --- 3. Process Data Row ---
99     if current_section and current_keys:
100         raw_row_data = {}
101         for col_idx, key in current_keys:
102             if col_idx < len(row):
103                 val = row[col_idx]
104                 if pd.isna(val) or str(val).strip() == "":
105                     raw_row_data[key] = None
106                 else:
107                     raw_row_data[key] = val
108             else:
109                 raw_row_data[key] = None
110
111         # === DATA EXTRACTION ONLY ===
112         # 'item' is already inside raw_row_data because of the
113         loop above.
114         # We do NOT validate it.
115
116         # === VALIDATION (Other Fields) ===
117         qty_val = raw_row_data.get('quantity')
118         if qty_val is None:
119             continue
120
121         final_row_data = raw_row_data.copy()
122         row_has_critical_error = False

```

```

122         is_duplicate = False
123
124         # We grab item just for the error log label, no
125         validation logic applied.
126         item_code = final_row_data.get('item')
127         row_context = get_row_identifier(sheet_name,
128                                         current_section['type'], idx, item_code)
129
130         # Validate Quantity
131         if not is_valid_number(qty_val):
132             global_errors.append(f"{row_context} - Error:
133                                 Quantity '{qty_val}' is not a valid number.")
134             row_has_critical_error = True
135
136         # Validate Rate
137         rate_val = final_row_data.get('rate')
138         if rate_val is None:
139             global_errors.append(f"{row_context} - Error:
140                                 Quantity is present but 'rate' is missing.")
141             row_has_critical_error = True
142         elif not is_valid_number(rate_val):
143             global_errors.append(f"{row_context} - Error: Rate
144                                 '{rate_val}' is not a valid number.")
145             row_has_critical_error = True
146
147         # Validate Sources (Warning only)
148         if final_row_data.get('rate_src') is None:
149             global_warnings.append(f"{row_context} - Warning: '
150                                   rate_src' is missing.")
151         if final_row_data.get('carbon_emission_src') is None:
152             global_warnings.append(f"{row_context} - Warning: '
153                                   carbon_emission_src' is missing.")
154
155         # Validate Flexible Numbers
156         for field in FLEXIBLE_FIELD_CONFIG:
157             val = final_row_data.get(field)
158             if val is not None:
159                 val_str = str(val).strip().lower()
160                 if not is_valid_number(val) and val_str not in

```

```

154         ALLOWED_FLEXIBLE_VALUES:
            global_errors.append(f"{row_context} -
                Error: Field '{field}' value '{val}' is
                invalid. Must be number or {
                ALLOWED_FLEXIBLE_VALUES}.")
155         row_has_critical_error = True
156
157     # Validate Units
158     unit_val = final_row_data.get('unit')
159     if unit_val:
160         if str(unit_val).strip().lower() not in
            ALLOWED_UNITS:
161             global_errors.append(f"{row_context} - Error:
                Unit '{unit_val}' is not valid. Allowed: {
                ALLOWED_UNITS}")
162             row_has_critical_error = True
163     else:
164         global_errors.append(f"{row_context} - Error: Unit
            is missing.")
165         row_has_critical_error = True
166
167     # Validate Recyclability Percentage
168     perc_val = final_row_data.get('recyclability_percentage
        ')
169     if perc_val is None or str(perc_val).strip() == "":
170         global_warnings.append(f"{row_context} - Warning:
            'recyclability_percentage' is missing.")
171     else:
172         if not is_valid_number(perc_val):
173             global_errors.append(f"{row_context} - Error:
                'recyclability_percentage' '{perc_val}' is
                not a valid number.")
174             row_has_critical_error = True
175         else:
176             if float(perc_val) > 100:
177                 global_errors.append(f"{row_context} -
                    Error: 'recyclability_percentage' ({
                    perc_val}) cannot be greater than 100.")
178                 row_has_critical_error = True

```

```

179         elif float(perc_val) < 0:
180             global_errors.append(f"{row_context} -
                Error: 'recyclability_percentage' ({
                    perc_val}) cannot be negative.")
181             row_has_critical_error = True
182
183     # Validate Scrap Rate
184     scrap_val = final_row_data.get('scrap_rate')
185     if scrap_val is None or str(scrap_val).strip() == "":
186         global_warnings.append(f"{row_context} - Warning: '
            scrap_rate' is missing.")
187     else:
188         if not is_valid_number(scrap_val):
189             global_errors.append(f"{row_context} - Error:
                'scrap_rate' '{scrap_val}' is not a valid
                number.")
190             row_has_critical_error = True
191
192     # === DUPLICATE CHECK ===
193     name_val = final_row_data.get('name')
194     if name_val:
195         if name_val in section_seen_data:
196             original_entry = section_seen_data[name_val]
197             original_data = original_entry['data']
198             original_loc = original_entry['row_id']
199
200             if final_row_data == original_data:
201                 global_warnings.append(f"{row_context} -
                    Warning: Duplicate material '{name_val}'
                    found with identical values. Rows have
                    been merged.")
202                 is_duplicate = True
203             else:
204                 global_errors.append(f"{row_context} -
                    Error: Duplicate material '{name_val}'
                    found with conflicting values compared
                    to {original_loc}. User selection
                    required.")
205                 is_duplicate = True

```

```

206         else:
207             section_seen_data[name_val] = {
208                 'data': final_row_data,
209                 'row_id': row_context
210             }
211
212         if not row_has_critical_error and not is_duplicate:
213             current_section['data'].append(final_row_data)
214
215     if current_section:
216         parsed_sections.append(current_section)
217
218     return {
219         "validation_report": {
220             "errors": global_errors,
221             "warnings": global_warnings
222         },
223         "parsed_data": parsed_sections
224     }
225
226 if __name__ == "__main__":
227     file_path = r"C:\Users\Ritik\Downloads\5C227B00.xlsx"
228     result = parse_and_validate_excel(file_path)
229     print(json.dumps(result, indent=2, ensure_ascii=False))
230
231 %----- end code -----

```

4.3.3 Explanation of the Code

- **Line 1–3:** Import required libraries. `pandas` is used for reading Excel files, `json` for structured output formatting, and `numpy` for handling missing values (NaN).
- **Line 5–10:** Define configuration rules such as allowed measurement units, flexible fields, and permitted special values.
- **Line 12–17:** `clean_header()` removes the “CID#” prefix and converts header names to lowercase for uniform processing.

- **Line 19–27:** `is_valid_number()` checks whether a value can be converted to a valid floating-point number.
- **Line 29–38:** `get_row_identifier()` generates a readable row reference (sheet name, component type, row number, item code) for error tracking.
- **Line 42–47:** Begin main function `parse_and_validate_excel()` and load all Excel sheets without predefined headers.
- **Line 49–52:** Initialize containers for parsed sections, global errors, and warnings.
- **Line 54–61:** Iterate through each sheet and initialize trackers for sections and duplicate detection.
- **Line 63–66:** Skip completely empty rows.
- **Line 68–78:** Detect header rows starting with “CID#” and dynamically extract column keys.
- **Line 80–83:** Skip visual header rows such as “Material” and “SOR Item No.”.
- **Line 85–101:** Identify new component sections and initialize structured storage for each section.
- **Line 103–124:** Extract row data dynamically based on detected headers and store cleaned values.
- **Line 126–135:** Skip rows with missing quantity and prepare row context for validation messages.
- **Line 137–146:** Validate quantity (must be numeric).
- **Line 148–156:** Validate rate (mandatory and numeric).
- **Line 158–163:** Check for missing rate source and carbon emission source (warning only).
- **Line 165–175:** Validate flexible numeric fields (must be numeric or “not_available”).
- **Line 177–186:** Validate measurement unit against predefined allowed units.

- **Line 188–205:** Validate recyclability percentage (must be numeric and between 0 and 100).
- **Line 207–216:** Validate scrap rate (must be numeric, warning if missing).
- **Line 218–247:** Perform duplicate material detection. Identical duplicates generate warnings, conflicting duplicates generate errors.
- **Line 249–252:** Add row to section if no critical errors or duplication conflicts exist.
- **Line 254–256:** Append completed section to parsed sections list.
- **Line 258–266:** Return final JSON output containing validation report and parsed structured data.
- **Line 268–273:** Execute main block: specify file path, call parser function, and print formatted JSON result.

4.4 Validation Report

The validation module generates a structured report containing two main categories:

1. Errors

Errors indicate critical issues that prevent a row from being processed. Rows containing errors are excluded from the final parsed output.

Examples of error conditions:

- Quantity is missing or not a valid number.
- Rate is missing or not numeric.
- Unit is missing or not in the allowed unit list.
- Flexible fields (e.g., `carbon_emission`, `conversion_factor`) contain invalid values.
- Recyclability percentage is not numeric or outside the range 0–100.
- Scrap rate is not numeric.

- Duplicate material entries with conflicting values.

2. Warnings

Warnings indicate non-critical issues. Rows with warnings are still processed and included in the output.

Examples of warning conditions:

- Missing rate source (`rate_src`).
- Missing carbon emission source (`carbon_emission_src`).
- Missing recyclability percentage.
- Missing scrap rate.
- Duplicate material entries with identical values (automatically merged).

Report Structure

The final validation report is generated in JSON format:

```
{
  "validation_report": {
    "errors": [ ... ],
    "warnings": [ ... ]
  },
  "parsed_data": [ ... ]
}
```

Errors and warnings include:

- Sheet name
- Component type
- Excel row number
- Item code (if available)

This structured validation ensures data integrity, consistency, and traceability before further processing in the application.

4.5 Validation Report Documentation

The system performs structured validation on all Excel data before processing. It checks numerical fields (quantity, rate, recyclability percentage, scrap rate), verifies allowed measurement units, and ensures flexible fields contain either valid numbers or predefined acceptable values.

Errors are treated as critical issues and prevent the row from being included in the final parsed output. Warnings indicate non-critical issues such as missing source fields or duplicate entries with identical values.

The final output is generated in JSON format containing:

- A validation report (errors and warnings)
- Cleaned and structured parsed data

This ensures data accuracy, consistency, and traceability before integration into the main application.

Chapter 5

Internship Task 3

5.1 Task 3 Title

Integrating parser Code into the LCCA app Code

5.2 Tasks Done

5.2.1 Creation of Excel Data Parser for LCCA Application

The Excel Data Parser module was successfully integrated into the LCCA application within the `maintemplate.py` file, ensuring tight coupling with the core workflow of the system. The parser was designed to read structured Excel datasets, extract relevant life-cycle cost parameters, and convert them into a normalized internal data format compatible with the application's computational engine.

To maintain modularity and scalability, the parsing logic was encapsulated within a dedicated service layer. This layer is responsible not only for transforming raw Excel input into structured objects but also for enforcing validation rules and applying necessary business logic constraints specific to Life Cycle Cost Analysis (LCCA). By separating concerns, the architecture enhances maintainability and reduces code redundancy.

The service layer was then connected to the UI and backend controller components to establish a seamless data pipeline. Upon file upload, the Excel data is processed through the parser service, validated, and then forwarded to the analysis module for cost computation and reporting. This integration ensures real-time data reflection within the

application interface.

Structured error handling mechanisms were implemented across all layers—parser, service, and UI. The parser captures file format inconsistencies and missing columns, the service layer manages logical validation errors, and the UI provides user-friendly feedback messages to prevent system crashes and guide corrective action.

Comprehensive input validation checks were incorporated to ensure data integrity. These checks verify mandatory fields, numerical consistency, and acceptable value ranges to prevent malformed or incomplete datasets from entering the computational workflow.

Finally, rigorous testing was conducted using both valid and intentionally corrupted Excel datasets to evaluate robustness. The integration was confirmed to be stable, accurate, and resilient under edge-case scenarios, ensuring reliable performance within the LCCA application environment.

5.3 Task 3: Documentation

5.3.1 Overview of Excel Parser Integration in LCCA

The Excel parsing functionality was implemented directly within `maintemplate.py` as part of the LCCA desktop application. The parser is responsible for reading structured Excel files, validating construction-related data, transforming it into a normalized internal format, and distributing it across the appropriate UI widgets. The workflow is triggered through the `Upload Excel` button, which invokes the `open_excel_dialog()` method.

5.3.2 Excel Reading and Multi-Sheet Handling

The core parsing logic is implemented in the function `parse_and_validate_excel(file_path)`. The system uses the `pandas.read_excel()` function with `sheet_name=None` to load all sheets dynamically. Each sheet is processed independently, enabling support for multiple construction categories such as Foundation, Sub-Structure, Super-Structure, and Miscellaneous works.

The parser iterates row-by-row and identifies:

- Header rows (prefixed with `CID#`)
- Section identifiers (component-level grouping)

- Data rows containing material-level information

This design allows flexible Excel structures while maintaining structured output.

5.3.3 Header Normalization and Field Mapping

Header fields are cleaned using the `clean_header()` function, which removes prefixes like `CID#` and converts them to lowercase standardized keys. This ensures consistent internal key naming regardless of formatting variations in Excel.

Special handling is implemented for edge cases such as:

- Excavation sheets where index alignment differs
- Carbon emission unit normalization (e.g., handling `KgCO2e/kg` vs `kgCO2e/kg`)

This normalization step guarantees compatibility with downstream cost and emission calculations.

5.3.4 Validation Mechanism

A multi-layer validation framework is embedded within the parser. Validation is performed before any data is accepted into the system.

Critical Validations (Errors):

- Quantity must be numeric
- Rate must be present and numeric
- Unit must belong to predefined `ALLOWED_UNITS`
- Recyclability percentage must be numeric and within 0–100
- Flexible fields (carbon emission, conversion factor) must be numeric or explicitly marked `not_available`

If any critical validation fails, the import process is stopped using `show_error_message()`.

Non-Critical Validations (Warnings):

- Missing source references (`rate_src`, `carbon_emission_src`)
- Missing scrap rate

- Missing recyclability percentage

Warnings allow data import but notify the user through a structured dialog.

5.3.5 Duplicate Detection Logic

The parser implements section-wise duplicate detection using a dictionary-based tracking mechanism. If duplicate material names are found:

- Identical rows are merged with a warning.
- Conflicting rows generate a validation error requiring user intervention.

This prevents inconsistent cost computation and maintains data integrity.

5.3.6 Data Persistence and JSON Storage

After successful validation:

1. Data is saved into the active project using `ProjectDataManager.bulk_import_excel_data()`.
2. A temporary JSON file (`temporary_construction_data.json`) is generated for cost calculation and logging.

The method `save_temp_data()` ensures persistent storage in the local `temp_file_db` directory. Startup cleanup logic removes stale temporary files to maintain consistency.

5.3.7 Cost Calculation from Parsed Data

The method `calculate_cost_from_json()` reads the generated JSON file and computes construction costs category-wise. The computation follows:

$$\text{Total Cost} = \sum (\text{Quantity} \times \text{Rate})$$

Costs are grouped into:

- FOUNDATION
- SUB-STRUCTURE

- SUPER-STRUCTURE
- MISCELLANEOUS

A structured dictionary containing the breakdown and grand total (in INR) is printed to the terminal and used for locking the construction stage.

5.3.8 UI Distribution and Widget Mapping

Parsed sections are distributed using `distribute_imported_data()`. A sheet-to-widget mapping system dynamically assigns parsed data to the correct tab widget (Foundation, SuperStructure, etc.).

The system supports:

- Dynamic tab creation
- Widget caching for data persistence
- Automatic project ID synchronization

This ensures that Excel-imported data populates the corresponding construction widgets seamlessly.

5.3.9 Construction Locking Mechanism

After calculation, the method `calculate_and_lock_construction_data()` locks construction data to prevent further modification. This ensures analytical integrity before proceeding to financial and carbon analysis stages.

5.3.10 Conclusion

The Excel parser implementation provides a robust, validated, and structured data ingestion pipeline within the LCCA application. It integrates file parsing, validation, duplicate control, persistence, UI mapping, and cost computation into a cohesive workflow. The architecture ensures data integrity, computational reliability, and scalability for future enhancements.

5.3.11 Python Code

The Python script is shown below. Each section is commented for clarity.

Listing 5.1: Parser Code which is integrated into the UI

```
1  %-----begin code-----
2
3  import pandas as pd
4  import json
5  import numpy as np
6  import os
7
8  # Database Constants Setup
9  BASE_DIR = os.path.dirname(os.path.abspath(__file__))
10 DB_FOLDER = os.path.join(BASE_DIR, "temp_file_db")
11 JSON_DB_PATH = os.path.join(DB_FOLDER, "temporary_construction_data.
    json")
12
13 # ===== RITIK : EXCEL PARSING LOGIC =====
14
15 ALLOWED_UNITS = {'cum', 'rmt', 'm2', 'mt', 'nos', 'kg'}
16 FLEXIBLE_FIELD_CONFIG = ['carbon_emission', 'conversion_factor']
17 ALLOWED_FLEXIBLE_VALUES = {'not_available'}
18 ALLOWED_RECYCLE_VALUES = {'recycleable', 'recyclable', 'non-recyclable'
    }
19
20 def clean_header(header_val):
21     if not isinstance(header_val, str):
22         return str(header_val)
23     # Converts 'CID#Item' -> 'item'
24     return header_val.replace('CID#', '').lower().strip()
25
26 def is_valid_number(val):
27     if val is None or val == "":
28         return False
29     try:
30         float(val)
31         return True
32     except (ValueError, TypeError):
```

```

33         return False
34
35 def get_row_identifier(sheet, type_name, row_idx, item_code=None):
36     # Pandas index 0 = Excel Row 1
37     base_id = f"Sheet: '{sheet}', Component: '{type_name}', Row: {
38         row_idx + 1}"
39     if item_code:
40         return f"{base_id} [Item: {item_code}]"
41     return base_id
42
43 def normalize_unit_string(val):
44     """
45     RITIK: Helper to normalize known unit strings, especially handling
46     typos like KgCO2e (zero) vs kgCO2e.
47     """
48     if not isinstance(val, str):
49         return val
50
51     val_clean = val.strip()
52
53     # Specific replacements
54     # Case: "KgCO2e/kg" (Capital K, C, Zero instead of O) -> "k g C O e /
55     kg"
56     # Case: "KgCO2e/kg" (Capital K, C, Letter O) -> "k g C O e /kg"
57     # Case: "kgCO2e/kg" (lowercase k, Letter O) -> "k g C O e /kg"
58     if val_clean in ["KgCO2e/kg", "KgCO2e/kg", "kgCO2e/kg", "kgCO2e/kg"
59         ]:
60         return "kgCO\u2082e/kg"
61
62     return val
63
64 def parse_and_validate_excel(file_path):
65     try:
66         all_sheets = pd.read_excel(file_path, sheet_name=None, header=
67             None)
68     except Exception as e:
69         return {"fatal_error": f"Failed to read file: {str(e)}"}
70
71     parsed_sections = []

```

```

67     global_errors = []
68     global_warnings = []
69
70     for sheet_name, df in all_sheets.items():
71
72         current_section = None
73         current_keys = None
74         section_seen_data = {}
75
76         for idx, row in df.iterrows():
77             if row.isnull().all():
78                 continue
79
80             # Safely get first two string values for detection
81             def get_val(r, i):
82                 if i < len(r) and pd.notna(r[i]):
83                     return str(r[i]).strip()
84                 return ""
85
86             first_val = get_val(row, 0)
87             second_val = get_val(row, 1)
88
89             # --- 1. Identify Header (CID#) ---
90             # We check col 0 OR col 1 to support the 'Excavation' sheet
91             structure
92             is_header_row = first_val.startswith('CID#') or second_val.
93                 startswith('CID#')
94
95             if is_header_row:
96                 current_keys = []
97                 for i, val in enumerate(row):
98                     v_str = get_val(row, i)
99                     if v_str != "":
100                         current_keys.append((i, clean_header(v_str)))
101
102             # SPECIAL HANDLING: Excavation Sheet
103             # If header starts at index 1 (Name) but index 0 is
104             empty,
105             # we explicitly map index 0 to 'item' so we capture the

```

```

        item code.
103         has_idx_0 = any(k[0] == 0 for k in current_keys)
104         has_name_at_1 = any(k[0] == 1 and k[1] == 'name' for k
            in current_keys)
105
106         if not has_idx_0 and has_name_at_1:
107             current_keys.insert(0, (0, 'item'))
108
109         continue
110
111     # --- Skip Visual Headers ---
112     if first_val in ['Material', 'SOR Item No.']:
113         continue
114
115     # --- 2. Identify Section ---
116     col2 = row[2] if len(row) > 2 else np.nan
117     col3 = row[3] if len(row) > 3 else np.nan
118
119     if pd.notna(row[0]) and pd.isna(col2) and pd.isna(col3):
120         if current_section:
121             parsed_sections.append(current_section)
122
123             current_section = {
124                 "sheetName": sheet_name,
125                 "type": first_val,
126                 "data": []
127             }
128             section_seen_data = {}
129             continue
130
131     # --- 3. Process Data Row ---
132     if current_section and current_keys:
133         raw_row_data = {}
134         for col_idx, key in current_keys:
135             val = row[col_idx] if col_idx < len(row) else None
136             if pd.isna(val) or str(val).strip() == "":
137                 raw_row_data[key] = None
138             else:
139                 if key == 'carbon_emission_units':

```

```

140         val = normalize_unit_string(val)
141         raw_row_data[key] = val
142
143         # === VALIDATION ===
144         qty_val = raw_row_data.get('quantity')
145         if qty_val is None:
146             continue
147
148         final_row_data = raw_row_data.copy()
149         row_has_critical_error = False
150         is_duplicate = False
151
152         # Context for logging
153         item_code = final_row_data.get('item')
154         row_context = get_row_identifer(sheet_name,
155                                         current_section['type'], idx, item_code)
156
157         # Validate Quantity
158         if not is_valid_number(qty_val):
159             global_errors.append(f"{row_context} - Error:
160                                 Quantity '{qty_val}' is not a valid number.")
161             row_has_critical_error = True
162
163         # Validate Rate
164         rate_val = final_row_data.get('rate')
165         if rate_val is None:
166             global_errors.append(f"{row_context} - Error:
167                                 Quantity is present but 'rate' is missing.")
168             row_has_critical_error = True
169         elif not is_valid_number(rate_val):
170             global_errors.append(f"{row_context} - Error: Rate
171                                 '{rate_val}' is not a valid number.")
172             row_has_critical_error = True
173
174         # Validate Sources
175         if final_row_data.get('rate_src') is None:
176             global_warnings.append(f"{row_context} - Warning: '
177                                    rate_src' is missing.")
178         if final_row_data.get('carbon_emission_src') is None:

```

```

174         global_warnings.append(f"{row_context} - Warning: '
            carbon_emission_src' is missing.")
175
176     # Validate Flexible Numbers
177     for field in FLEXIBLE_FIELD_CONFIG:
178         val = final_row_data.get(field)
179         if val is not None:
180             val_str = str(val).strip().lower()
181             if not is_valid_number(val) and val_str not in
                ALLOWED_FLEXIBLE_VALUES:
182                 global_errors.append(f"{row_context} -
                    Error: Field '{field}' value '{val}' is
                    invalid. Must be number or {
                    ALLOWED_FLEXIBLE_VALUES}.")
183                 row_has_critical_error = True
184
185     # Validate Units
186     unit_val = final_row_data.get('unit')
187     if unit_val:
188         if str(unit_val).strip().lower() not in
            ALLOWED_UNITS:
189             global_errors.append(f"{row_context} - Error:
                Unit '{unit_val}' is not valid. Allowed: {
                ALLOWED_UNITS}")
190             row_has_critical_error = True
191     else:
192         global_errors.append(f"{row_context} - Error: Unit
            is missing.")
193         row_has_critical_error = True
194
195     # Validate Recyclability
196     perc_val = final_row_data.get('recyclability_percentage')
197
198     if perc_val is None or str(perc_val).strip() == "":
199         global_warnings.append(f"{row_context} - Warning:
            'recyclability_percentage' is missing.")
200     else:
201         if not is_valid_number(perc_val):
            global_errors.append(f"{row_context} - Error:

```

```

201         'recyclability_percentage' '{perc_val}' is
202         not a valid number.")
203     row_has_critical_error = True
204     else:
205         if float(perc_val) > 100:
206             global_errors.append(f"{row_context} -
207                 Error: 'recyclability_percentage' ({
208                     perc_val}) cannot be greater than 100.")
209             row_has_critical_error = True
210         elif float(perc_val) < 0:
211             global_errors.append(f"{row_context} -
212                 Error: 'recyclability_percentage' ({
213                     perc_val}) cannot be negative.")
214             row_has_critical_error = True
215
216     # Validate Scrap Rate
217     scrap_val = final_row_data.get('scrap_rate')
218     if scrap_val is None or str(scrap_val).strip() == "":
219         global_warnings.append(f"{row_context} - Warning: '
220             scrap_rate' is missing.")
221     else:
222         if not is_valid_number(scrap_val):
223             global_errors.append(f"{row_context} - Error:
224                 'scrap_rate' '{scrap_val}' is not a valid
225                 number.")
226             row_has_critical_error = True
227
228     # === DUPLICATE CHECK ===
229     name_val = final_row_data.get('name')
230     if name_val:
231         if name_val in section_seen_data:
232             original_entry = section_seen_data[name_val]
233             original_data = original_entry['data']
234             original_loc = original_entry['row_id']
235
236             if final_row_data == original_data:
237                 global_warnings.append(f"{row_context} -
238                     Warning: Duplicate material '{name_val}'
239                     found with identical values. Rows have

```

```

                been merged.")
230         is_duplicate = True
231     else:
232         global_errors.append(f"{row_context} -
                                Error: Duplicate material '{name_val}'
                                found with conflicting values compared
                                to {original_loc}. User selection
                                required.")
233         is_duplicate = True
234     else:
235         section_seen_data[name_val] = {
236             'data': final_row_data,
237             'row_id': row_context
238         }
239
240         if not row_has_critical_error and not is_duplicate:
241             current_section['data'].append(final_row_data)
242
243     if current_section:
244         parsed_sections.append(current_section)
245
246     return {
247         "validation_report": {
248             "errors": global_errors,
249             "warnings": global_warnings
250         },
251         "parsed_data": parsed_sections
252     }
253
254 def show_error_message(error_list, parent=None):
255     error_text = "      ERRORS FOUND - Import Stopped\n\n" + "      " * 60 +
256                 "\n"
257     for idx, error in enumerate(error_list, 1):
258         error_text += f"{idx}. {error}\n"
259     error_text += "      " * 60
260     QMessageBox.critical(parent, "Excel Import - Errors", error_text)
261     return False
262
263 def show_warning_message(warning_list, parent=None):

```

```

263     warning_text = "            WARNINGS\n\n" + "    " * 60 + "\n"
264     for idx, warning in enumerate(warning_list, 1):
265         warning_text += f"{idx}. {warning}\n"
266     warning_text += "    " * 60 + "\n\nData will be imported with\n\n"
267     QMessageBox.warning(parent, "Excel Import - Warnings", warning_text
268                          )
269     return True
270
271 # RITIK: End of Excel Parsing Logic
272 def map_parsed_data_to_widgets(parsed_data, ui_instance):
273     # This logic is integrated into distribute_imported_data
274     pass
275
276 def map_section_to_widget(widget, component_type, data_items):
277     # This logic is integrated into distribute_imported_data
278     pass
279
280 class UiMainWindow(object):
281     # --- Ritik: Excel Functionality ---
282     def open_excel_dialog(self):
283         # 1. CHECK FOR ACTIVE PROJECT (REQUIRED FOR AUTOSAVE)
284         if not self.current_project_id:
285             QMessageBox.warning(self.central_widget, "No Project", "
286                 Please create or open a project before uploading Excel
287                 data.")
288             return
289
290         print("[UI] Upload Excel clicked")
291         file_path, _ = QFileDialog.getOpenFileName(None, "Select Excel
292             File", "", "Excel Files (*.xlsx *.xls)")
293         if not file_path:
294             return
295
296         print(f"[UI] Excel selected: {file_path}")
297         result = parse_and_validate_excel(file_path)
298         print("\n===== PARSER OUTPUT =====\n")
299         print(json.dumps(result, indent=2, ensure_ascii=False))

```

```

297     validation_report = result.get('validation_report', {})
298     errors = validation_report.get('errors', [])
299     warnings = validation_report.get('warnings', [])
300     parsed_data = result.get('parsed_data', [])
301
302     if errors:
303         show_error_message(errors, parent=self.windows)
304         return
305
306     # -----
307     #  SAVE TO AUTOSAVE.JSON (Project Persistence)
308     # -----
309     # This saves the data directly to the project structure (
310         autosave.json)
311     try:
312         self.database_manager.bulk_import_excel_data(self.
313             current_project_id, parsed_data)
314     except AttributeError:
315         print("      Error: DatabaseManager missing '
316             bulk_import_excel_data'. Data not saved to project file
317             .")
318
319     # -----
320     #  SAVE TO TEMP JSON (Preserved for Calculation & Logs)
321     # -----
322     # Keeps existing functionality for temporary file generation
323     and logs
324     self.save_temp_data(parsed_data)
325
326     # Reset lock flag on new upload to allow re-calculation
327     self.construction_locked = False
328     print("[LOCK] Construction lock reset due to new data upload.")
329
330     if warnings:
331         show_warning_message(warnings, parent=self.windows)
332
333     print(f"\n[MAPPING] Starting mapping of {len(parsed_data)}
334         section(s)...")

```

```

330 # -----
331 # DISTRIBUTE TO UI (Preserved for Logs & Widget Pop)
332 # -----
333 # This function generates the specific terminal output you
    requested
334 self.distribute_imported_data(parsed_data)
335
336 # New method to save parsed data to JSON file
337 def save_temp_data(self, parsed_data):
338     """Saves the parsed Excel data to a local JSON file."""
339     try:
340         # Create db directory if it doesn't exist
341         if not os.path.exists(DB_FOLDER):
342             os.makedirs(DB_FOLDER)
343             print(f"[DB] Created database folder at: {DB_FOLDER}")
344
345         # Write data to JSON
346         with open(JSON_DB_PATH, 'w', encoding='utf-8') as f:
347             json.dump(parsed_data, f, indent=4, ensure_ascii=False)
348
349         print(f"[DB]      Data successfully saved to: {JSON_DB_PATH}")
350         return True
351     except Exception as e:
352         print(f"[DB]      Error saving data: {str(e)}")
353         return False
354
355 # New method to calculate cost directly from JSON DB
356 def calculate_cost_from_json(self):
357     """Reads the JSON DB, calculates costs, and prints a neat
        dictionary to terminal."""
358     if not os.path.exists(JSON_DB_PATH):
359         print("[CALC]      No data file found. Please upload Excel
            first.")
360         return None
361
362     print("\n" + "="*60)
363     print("                  FINAL CONSTRUCTION COST CALCULATION (FROM
        JSON DB)")

```

```

364     print("="*60)
365
366     try:
367         with open(JSON_DB_PATH, 'r', encoding='utf-8') as f:
368             data = json.load(f)
369     except Exception as e:
370         print(f"[CALC] Error reading JSON: {e}")
371         return None
372
373     # Initialize Cost Structure
374     cost_summary = {
375         "FOUNDATION": 0.0,
376         "SUB-STRUCTURE": 0.0,
377         "SUPER-STRUCTURE": 0.0,
378         "MISCELLANEOUS": 0.0
379     }
380
381     # Mapping variations in Excel sheet names to our standard Keys
382     key_map = {
383         'foundation': 'FOUNDATION',
384         'sub-structure': 'SUB-STRUCTURE', 'sub structure': 'SUB-
385             STRUCTURE',
386         'super-structure': 'SUPER-STRUCTURE', 'super structure': '
387             SUPER-STRUCTURE',
388         'miscellaneous': 'MISCELLANEOUS', 'auxiliary works': '
389             MISCELLANEOUS'
390     }
391
392     # Calculation Loop
393     for section in data:
394         raw_sheet_name = str(section.get('sheetName', '')).lower().
395             strip()
396         category = key_map.get(raw_sheet_name, 'MISCELLANEOUS')
397
398         # Print Section Header
399         print(f"\n--- Processing: {section.get('type', 'Unknown
400             Component')}] ({category}) ---")
401         print(f"{'Material':<50} | {'Qty':<10} | {'Rate':<10} | {'
402             Total':<15}")

```

```

397         print("-" * 90)
398
399         for material in section.get('data', []):
400             try:
401                 name = material.get('name', 'Unknown')
402                 qty = float(material.get('quantity', 0))
403                 rate = float(material.get('rate', 0))
404                 total = qty * rate
405
406                 # Add to category total
407                 cost_summary[category] += total
408
409                 # Print Row
410                 print(f"{name:<50} | {qty:<10.2f} | {rate:<10.2f} |  

411                        {total:<15.2f}")
412             except (ValueError, TypeError):
413                 continue
414
415         # Calculate Grand Total
416         grand_total = sum(cost_summary.values())
417
418         # Final Dictionary Output
419         print("\n" + "="*60)
420         print("          FINAL COST DICTIONARY")
421         print("="*60)
422
423         final_output = {
424             "breakdown": cost_summary,
425             "grand_total": grand_total,
426             "currency": "INR"
427         }
428
429         # Pretty print the dictionary
430         print(json.dumps(final_output, indent=4, ensure_ascii=False))
431         print("="*60 + "\n")
432
433         return grand_total
434
435 def distribute_imported_data(self, parsed_data):

```

```

435     print(f"\n[DISTRIBUTE] Started distribution of {len(parsed_data
436           )} section(s)")
437
438     sheet_map = {
439         "foundation": KEY_FOUNDATION, "sub structure":
440             KEY_SUBSTRUCTURE, "sub-structure": KEY_SUBSTRUCTURE,
441         "substructure": KEY_SUBSTRUCTURE, "super structure":
442             KEY_SUPERSTRUCTURE, "super-structure":
443             KEY_SUPERSTRUCTURE,
444         "superstructure": KEY_SUPERSTRUCTURE, "miscellaneous":
445             KEY_AUXILIARY, "auxiliary works": KEY_AUXILIARY,
446         "auxiliary": KEY_AUXILIARY
447     }
448
449     grouped_data = {}
450     for section in parsed_data:
451         sheet_name_clean = str(section.get('sheetName', '')).lower
452             ().strip()
453         target_key = None
454         for key_alias, key_const in sheet_map.items():
455             if key_alias in sheet_name_clean:
456                 target_key = key_const
457                 break
458         if target_key:
459             if target_key not in grouped_data: grouped_data[
460                 target_key] = []
461             grouped_data[target_key].append(section)
462
463     for widget_key, sections in grouped_data.items():
464         if self.tabs_active and widget_key in self.
465             active_tab_widgets:
466             tab_idx = self.active_tab_widgets[widget_key]
467             if hasattr(self.current_right_widget, 'widget'):
468                 # JAWWAD: If widgets are cached, accessing via
469                 active tab might get the new UI instance
470                 # but logic below ensures we load data into it.
471                 widget_instance = self.current_right_widget.widget(
472                     tab_idx)
473                 if hasattr(widget_instance, '

```

```

464         load_from_excel_sections'):
465             widget_instance.load_from_excel_sections(
466                 sections)
467     else:
468         if not self.tabs_active:
469             self.tabs_active = True
470             self.active_tab_widgets = {}
471             self.current_right_widget = CustomTabWidget(parent=
472                 self)
473             self.right_panel_placeholder.layout().addWidget(
474                 self.current_right_widget)
475
476             # Simplified this block to use the standard
477             add_new_tab_widget method
478             # which now handles caching (retrieving existing widget
479             with data).
480             if widget_key not in self.active_tab_widgets:
481                 self.add_new_tab_widget(widget_key)
482
483             if widget_key in self.active_tab_widgets:
484                 tab_idx = self.active_tab_widgets[widget_key]
485                 widget_instance = self.current_right_widget.widget(
486                     tab_idx)
487                 if hasattr(widget_instance, '
488                     load_from_excel_sections'):
489                     widget_instance.load_from_excel_sections(
490                         sections)
491
492 # New cleanup method added here
493 def cleanup_temp_db(self):
494     """Removes the temporary JSON database file at application
495     startup."""
496     try:
497         if os.path.exists(JSON_DB_PATH):
498             try:
499                 os.remove(JSON_DB_PATH)
500                 print(f"[CLEANUP] Existing temporary database
501                     deleted: {JSON_DB_PATH}")
502             except PermissionError:

```

```

492         print(f"[CLEANUP] WARNING: Could not delete {
           JSON_DB_PATH} (locked). Skipping cleanup.")
493     except Exception as e:
494         print(f"[CLEANUP] WARNING: Error deleting {
           JSON_DB_PATH}: {e}")
495     else:
496         print("[CLEANUP] No existing temporary database found.
           Starting fresh.")
497
498         # Ensure the directory itself still exists for future
           uploads
499         if not os.path.exists(DB_FOLDER):
500             os.makedirs(DB_FOLDER)
501     except Exception as e:
502
503
504 def setupUi(self, MainWindow):
505     # Call the cleanup method immediately on startup
506     self.cleanup_temp_db()
507
508     # Initialize ProjectDataManager (Core) instead of
           DatabaseManager (Utils)
509     self.database_manager = ProjectDataManager()
510
511     self.tabs_active = False
512     self.active_tab_widgets = {}
513
514     # Cache to store construction widgets to persist data when
           navigating away
515     self.cached_construction_widgets = {}
516
517     # --- NEW: Flag to track if construction data is locked ---
518     self.construction_locked = False
519     # -----
520
521     # Variable to store the active project UUID globally
522     self.current_project_id = None
523
524     self.widget_map = {

```

```

525         KEY_STRUCTURE_WORKS_DATA: Foundation, KEY_FOUNDATION:
           Foundation,
526         KEY_SUPERSTRUCTURE: SuperStructure, KEY_SUBSTRUCTURE:
           SubStructure,
527         KEY_AUXILIARY: AuxiliaryWorks, KEY_FINANCIAL: FinancialData
           ,
528         KEY CARBON_EMISSION: CarbonEmissionData,
529         KEY_MACHINERY_DATA: CarbonMachineryWidget,
530         KEY_BRIDGE_TRAFFIC: BridgeAndTrafficData,
           KEY_MAINTAINANCE_REPAIR: MaintenanceRepairData,
531         KEY_RECYCLABLE: RecyclableWidget,
532         KEY_DEMOLITION_RECYCLE: DemolitionAndRecyclingData,
533         KEY_TRANSPORTATION_DATA: TransportationMainWidget
534     }
535
536     if not MainWindow.setObjectName():
537         MainWindow.setObjectName(u"MainWindow")
538
539     font_id = QFontDatabase.addApplicationFont(":/font/AlataRegular
           .ttf")
540     if font_id != -1:
541         font_family = QFontDatabase.applicationFontFamilies(font_id
           )[0]
542         QApplication.setFont(QFont(font_family, 10))
543
544     MainWindow.setWindowTitle("3psLCCA")
545     screen = QApplication.primaryScreen().geometry()
546     MainWindow.setGeometry(screen.width()//8, screen.height()//8,
           screen.width()*3//4, screen.height()*3//4)
547
548     MainWindow.setStyleSheet("""
549         QMainWindow { border: none; }
550         QMenuBar { background-color: #E8F5E9; border-bottom: 1px
           solid #d0d0d0; border-left: 1px solid #285A23; border-
           right: 1px solid #285A23; }
551         QMenuBar::item { padding: 4px 10px; background-color:
           transparent; border-bottom: 2px solid #FAFAFA; margin: 2
           px; }
552         QMenuBar::item:selected { border-bottom: 2px solid #806C6C;

```

```

    }
553     QMenu { background-color: #f0f0f0; border: 1px solid #
           d0d0d0; }
554     QMenu::item { padding: 4px 4px 4px 12px; text-align: left;
           color: #514E4E; }
555     QMenu::item:selected { background-color: #e0e0e0; }
556     QMenu::separator { height: 1px; background-color: #d0d0d0;
           margin: 4px 0px; }
557     QMenu::icon { padding-left: 5px; width: 16px; height: 16px;
           }
558     QMenu::indicator { width: 16px; height: 16px; }
559     QMenu::right-arrow { margin-right: 5px; }
560 """)
561
562     self.central_widget = QWidget()
563     self.central_widget.setStyleSheet("border: none;")
564     self.central_widget.setObjectName("central_widget")
565     MainWindow.setCentralWidget(self.central_widget)
566     main_layout = QVBoxLayout(self.central_widget)
567     main_layout.setContentsMargins(0, 0, 0, 0)
568     main_layout.setSpacing(0)
569
570     self.menubar = QMenuBar()
571     self.menubar.setObjectName(u"menubar")
572     main_layout.addWidget(self.menubar)
573
574     self.menuFile = QMenu("&File", self.menubar)
575     self.menuHome = QMenu("&Home", self.menubar)
576     self.menuReport = QMenu("&Report", self.menubar)
577     self.menuHelp = QMenu("&Help", self.menubar)
578     self.menubar.addMenu(self.menuFile)
579     self.menubar.addMenu(self.menuHome)
580     self.menubar.addMenu(self.menuReport)
581     self.menubar.addMenu(self.menuHelp)
582
583     self.actionNew = QAction(QIcon(":/vectors/new.svg"), "New",
           MainWindow)
584     self.actionOpen = QAction(QIcon(":/vectors/open.svg"), "Open",
           MainWindow)

```

```

585     self.actionSave = QAction(QIcon(":/vectors/save.svg"), "Save",
                                MainWindow)
586     self.actionSaveAs = QAction(QIcon(":/vectors/save_as.svg"), "
        Save As...", MainWindow)
587     self.actionCreateCopy = QAction(QIcon(":/vectors/create_copy.
        svg"), "Create a Copy", MainWindow)
588     self.actionPrint = QAction(QIcon(":/vectors/print.svg"), "Print
        ", MainWindow)
589     self.actionRename = QAction(QIcon(":/vectors/rename.svg"), "
        Rename", MainWindow)
590     self.actionExport = QAction(QIcon(":/vectors/export.svg"), "
        Export", MainWindow)
591     self.actionVersionHistory = QAction(QIcon(":/vectors/
        version_history.svg"), "Version History", MainWindow)
592     self.actionInfo = QAction(QIcon(":/vectors/info.svg"), "Info",
                                MainWindow)
593
594     self.menuFile.addAction(self.actionNew)
595     self.menuFile.addAction(self.actionOpen)
596     self.menuFile.addSeparator()
597     self.menuFile.addAction(self.actionSave)
598     self.menuFile.addAction(self.actionSaveAs)
599     self.menuFile.addAction(self.actionCreateCopy)
600     self.menuFile.addAction(self.actionPrint)
601     self.menuFile.addSeparator()
602     self.menuFile.addAction(self.actionRename)
603     self.menuFile.addAction(self.actionExport)
604     self.menuFile.addAction(self.actionVersionHistory)
605     self.menuFile.addAction(self.actionInfo)
606
607     self.actionDocumentation = QAction(QIcon(":/vectors/contact.svg
        "), "Contact us", MainWindow)
608     self.actionFeedback = QAction(QIcon(":/vectors/feedback.svg"),
        "Feedback", MainWindow)
609     self.actionVideoTutorial = QAction(QIcon(":/vectors/
        video_tutorial.svg"), "Video Tutorials", MainWindow)
610     self.actionJoinCommunity = QAction(QIcon(":/vectors/
        join_community.svg"), "Join our Community", MainWindow)
611

```

```

612     self.menuHelp.addAction(self.actionDocumentation)
613     self.menuHelp.addAction(self.actionFeedback)
614     self.menuHelp.addSeparator()
615     self.menuHelp.addAction(self.actionVideoTutorial)
616     self.menuHelp.addAction(self.actionJoinCommunity)
617
618     self.main_content_area = QWidget()
619     main_layout.addWidget(self.main_content_area, 1)
620     self.main_content_area.setObjectName("main_content_area")
621     self.main_content_area.setStyleSheet("""
622         #main_content_area { background-color: #FAFAFA; border: 1px
623             solid #285A23; border-top: 1px solid #BBBBBB; }
624         QLabel { color: #9F8888; font-size: 14px; border: none;
625             padding: 5px; }
626         QPushButton { background-color: #EDED; border: 1px solid
627             #d0d0d0; padding: 6px 16px; color: #514E4E; }
628         QPushButton: hover { background: qlineargradient(x1: 0, y1:
629             0, x2: 0, y2: 1, stop: 0 #BBBBBB, stop: 0.26 #E8E8E8,
630             stop: 1 #EDED); border-color: #806C6C; }
631         QPushButton: pressed { background-color: #d0d0d0; }
632     """)
633
634     content_layout = QVBoxLayout(self.main_content_area)
635     content_layout.setContentsMargins(0,0,0,0)
636     content_layout.setSpacing(0)
637
638     button_layout = QHBoxLayout()
639     button_layout.setSpacing(0)
640     button_layout.setContentsMargins(5, 5, 5, 5)
641
642     self.edit_button = QPushButton()
643     self.edit_button.setObjectName(u"edit_button")
644     self.edit_button.setIcon(QIcon(":/images/edit_button.png"))
645     self.edit_button.setFixedSize(60, 30)
646     self.edit_button.setIconSize(QSize(25, 25))
647     self.edit_button.setStyleSheet("""
648         QPushButton { border-radius: 5px; background-color: #EDED
649             ; border: 1px solid #BBBBBB; margin-right: 20px; margin-
650             left: 10px; }

```

```

644         QPushButton:hover { background-color: #FAFAFA; border: 1px
        solid #888888; }
645     """)
646     button_layout.addWidget(self.edit_button)
647
648     self.file_button= QPushButton()
649     self.file_button.setObjectName(u"file_button")
650     self.file_button.setFixedSize(50, 30)
651     self.file_button.setIcon(QIcon(":/images/file_button.png"))
652     self.file_button.setIconSize(QSize(30, 30))
653     self.file_button.setStyleSheet("""
654         QPushButton { border-radius: 5px; background-color: #EDEDED
        ; border: 1px solid #BBBBBB; margin-right: 20px; }
655         QPushButton:hover { border: 1px solid #888888; background-
        color: #FAFAFA; }
656     """)
657     button_layout.addWidget(self.file_button)
658
659     self.save_button = QPushButton()
660     self.save_button.setObjectName(u"save_button")
661     self.save_button.setFixedSize(40, 30)
662     self.save_button.setIcon(QIcon(":/images/save_button.png"))
663     self.save_button.setIconSize(QSize(25, 25))
664     self.save_button.setStyleSheet("""
665         QPushButton { border-radius: 5px; background-color: #EDEDED
        ; border: 1px solid #BBBBBB; }
666         QPushButton:hover { background-color: #FAFAFA; border: 1px
        solid #888888; }
667         QPushButton::menu-indicator { image: url(/images/
        arrow_down.png); width: 8px; height: 30px; subcontrol-
        position: right center; subcontrol-origin: padding;
        margin-right: 2px; margin-left: 2px; border-left: 1px
        solid #BBBBBB; }
668     """)
669
670     self.save_menu = QMenu(self.save_button)
671     self.save_menu.setStyleSheet("""
672         QMenu { background-color: #EDEDED; border: 1px solid #
        BBBBBB; padding: 2px; color: #9F8888; }

```

```

673         QMenu::item { padding: 2px 2px; margin: 0px; border: none;
        min-height: 18px; font-size: 11px; }
674         QMenu::item:selected { background-color: #FAFAFA; }
675         QMenu::icon { width: 0px; }
676     """)
677     self.save_action = QAction("Save", self.save_menu)
678     self.save_as_action = QAction("Save As", self.save_menu)
679     self.save_menu.addAction(self.save_action)
680     self.save_menu.addAction(self.save_as_action)
681     self.save_button.setMenu(self.save_menu)
682     button_layout.addWidget(self.save_button)
683
684     self.upload_excel_button = QPushButton("Upload excel")
685     self.upload_excel_button.setObjectName("upload_excel_button")
686     self.upload_excel_button.clicked.connect(self.open_excel_dialog
        )
687     self.upload_excel_button.setFixedSize(100, 30)
688     self.upload_excel_button.setStyleSheet("""
689         QPushButton { border-radius: 5px; background-color:
        #217346; border: 1px solid #1a5c38; color: white; font-
        weight: bold; font-size: 10px; padding-left: 5px;
        padding-right: 5px; margin-left: 20px; }
690         QPushButton:hover { background-color: #2b945a; border: 1px
        solid #217346; }
691         QPushButton:pressed { background-color: #1a5c38; }
692     """)
693     self.upload_excel_button.setToolTip("Upload Excel")
694     button_layout.addWidget(self.upload_excel_button)
695
696     button_layout.addStretch()
697     self.windows = QLabel("Windows:")
698     button_layout.addWidget(self.windows)
699
700     self.tutorial_tab = QPushButton("Tutorials")
701     self.project_details_tab = QPushButton("Project Details")
702     self.results_tab = QPushButton("Results")
703     self.compare = QPushButton("Compare")
704
705     button_layout.addWidget(self.project_details_tab)

```

```

706         button_layout.addWidget(self.results_tab)
707         button_layout.addWidget(self.compare)
708         button_layout.addWidget(self.tutorial_tab)
709         button_layout.addStretch()
710
711         content_layout.addLayout(button_layout)
712
713         body_widget = QWidget()
714         body_widget.setObjectName("body_widget")
715         body_widget.setStyleSheet("#body_widget { border-top: 1px solid
716                                   #d0d0d0; }")
717         body_layout = QHBoxLayout(body_widget)
718         body_layout.setSpacing(20)
719
720         self.left_panel_placeholder = QWidget()
721         self.left_panel_placeholder.setLayout(QVBoxLayout())
722         self.right_panel_placeholder = QWidget()
723         self.right_panel_placeholder.setLayout(QVBoxLayout())
724         body_layout.addWidget(self.left_panel_placeholder, 1)
725         body_layout.addWidget(self.right_panel_placeholder, 4)
726         content_layout.addWidget(body_widget)
727
728         self.current_left_widget = None
729         self.current_right_widget = None
730
731         # --- Updated Signal Connections to use Class Methods ---
732         self.results_tab.clicked.connect(self.show_results_widget)
733         self.compare.clicked.connect(self.show_comparison_widget)
734         self.tutorial_tab.clicked.connect(self.show_tutorial_widget)
735         self.project_details_tab.clicked.connect(lambda: self.
736                                                  show_project_detail_widgets())
737
738         # --- NEW: Calculation & Locking Logic ---
739         def calculate_and_lock_construction_data(self):
740             """Calculates total costs, prints to terminal, shows alert, and
741                 locks data."""
742
743             # Use the new JSON calculation method instead of UI scraping
744             grand_total = self.calculate_cost_from_json()

```

```

742
743     # Fallback if calculation returns None (e.g., file error)
744     if grand_total is None:
745         grand_total = 0.0
746
747     # Show the Alert Message
748     msg = QMessageBox(self.central_widget)
749     msg.setWindowTitle("Stage Locked")
750     msg.setIcon(QMessageBox.Information)
751     msg.setText("Construction work data has been locked and saved."
752               )
753     msg.setInformativeText(f"Total Construction Cost calculated
754                           from Database:\nRs. {grand_total:,.2f}\n\nClick 'OK' to
755                           continue.")
756     msg.setStandardButtons(QMessageBox.Ok)
757     msg.exec()
758
759     self.construction_locked = True
760
761     # --- New method to handle project ID signal from
762     ProjectDetailsWidget ---
763
764     def handle_project_creation(self, project_id):
765         """Captures the project ID created by the details widget and
766         stores it globally."""
767         print(f"[DEBUG] [MainTemplate] Main Window received Project ID:
768               {project_id}")
769         self.current_project_id = project_id
770
771     # --- Refactored Navigation Logic (Fixes Recursion & Implements
772     Locking) ---
773
774     def show_project_detail_widgets(self, widget_name=None):
775         """Handles widget switching and the 'Gatekeeper' logic for
776         construction data."""
777         print(f"[DEBUG] show_project_detail_widgets called with: {
778               widget_name}")
779
780         # 1. Normalize the widget name
781         if widget_name == KEY_STRUCTURE_WORKS_DATA or widget_name == "
782           Construction work data":

```

```

771         widget_name = KEY_FOUNDATION
772
773         # 2. Gatekeeper Logic: Check if leaving Construction ->
774         Analysis
775
776         construction_keys = [KEY_FOUNDATION, KEY_SUBSTRUCTURE,
777                               KEY_SUPERSTRUCTURE, KEY_AUXILIARY]
778
779         is_moving_away = widget_name and (widget_name not in
780                                           construction_keys)
781
782         # Check if we are currently looking at construction tabs
783         current_is_construction = self.tabs_active and any(k in self.
784                                                            active_tab_widgets for k in construction_keys)
785
786         # Trigger lock if: Moving Away + Currently in Construction +
787         Not yet locked
788
789         if is_moving_away and current_is_construction and not self.
790            construction_locked:
791             self.calculate_and_lock_construction_data()
792
793         # 3. Widget Switching Logic
794         if widget_name and widget_name in self.widget_map:
795
796             # Case A: The target is a Construction Tab (Foundation,
797             Super, etc.)
798             if widget_name in construction_keys:
799                 if self.tabs_active:
800                     # Tabs are already open, just switch or add
801                     if self.active_tab_widgets.get(widget_name) is not
802                        None:
803                         index = self.active_tab_widgets[widget_name]
804                         self.current_right_widget.activate_tab(index)
805                     else:
806                         self.add_new_tab_widget(widget_name)
807                 else:
808                     # Initialize Tab View (First time opening
809                     construction)
810                     self.setup_tabbed_view(widget_name)
811
812             # Case B: The target is a Single Page (Financial, Carbon,

```

```

        etc.)
801     # Handle Navigation away from tabs
802     else:
803         # If we are currently in tab mode, we need to remove
            the tab widget
804         self.remove_right_widget()
805         self.tabs_active = False
806         self.active_tab_widgets = {}
807
808         # Create and show the new single widget (Financial, etc
            .)
809         target_class = self.widget_map.get(widget_name)
810         if target_class:
811             print(f"[DEBUG] Instantiating widget: {target_class
                }")
812             self.current_right_widget = target_class(database=
                self.database_manager, parent=self)
813
814             # Pass the project ID if available
815             if self.current_project_id and hasattr(self.
                current_right_widget, 'set_project_id'):
816                 print(f"[DEBUG] [MainTemplate] Setting Project
                    ID on {widget_name}: {self.
                        current_project_id}")
817                 self.current_right_widget.set_project_id(self.
                    current_project_id)
818
819             # Connect signals safely
820             if hasattr(self.current_right_widget, 'next'):
821                 self.current_right_widget.next.connect(self.
                    next_widget)
822             if hasattr(self.current_right_widget, 'back'):
823                 self.current_right_widget.back.connect(self.
                    prev_widget)
824
825             self.right_panel_placeholder.layout().addWidget(
                self.current_right_widget)
826             print(f"[DEBUG] Widget added to layout.")
827

```

```

828     else:
829         # -- Default / Project Details Landing Page --
830         self.remove_right_widget()
831         self.tabs_active = False
832         self.active_tab_widgets = {}
833         self.current_right_widget = ProjectDetailsWidget()
834
835         # CONNECT SIGNAL HERE - This fixes the "Project ID is None
836         # error
837         self.current_right_widget.projectCreated.connect(self.
838             handle_project_creation)
839
840         self.right_panel_placeholder.layout().addWidget(self.
841             current_right_widget)
842         self.current_right_widget.closed.connect(self.
843             remove_right_widget)
844
845         # Prevent recursion: Connect signals cleanly
846         if hasattr(self.current_right_widget, 'param_buttons'):
847             for btn in self.current_right_widget.param_buttons:
848                 try: btn.clicked.disconnect()
849                 except: pass
850                 btn.clicked.connect(lambda checked, b=btn: self.
851                     show_project_detail_widgets(b.text().strip()))
852
853     def setup_tabbed_view(self, start_widget_name):
854         """Helper to initialize the tabbed construction view"""
855         self.remove_right_widget()
856         self.active_tab_widgets = {}
857         self.tabs_active = True
858         self.current_right_widget = CustomTabWidget(parent=self)
859         self.right_panel_placeholder.layout().addWidget(self.
860             current_right_widget)
861
862         self.remove_left_widget()
863         self.current_left_widget = ProjectDetailsLeft(self.widget_map,
864             parent=self)
865         self.left_panel_placeholder.layout().addWidget(self.
866             current_left_widget)

```

```

859         self.current_left_widget.closed.connect(self.remove_left_widget
860             )
861
862         self.current_left_widget.handle_button_selection(button_name=
863             start_widget_name)
864
865     def add_new_tab_widget(self, widget_name):
866         """Helper to create or retrieve a widget tab."""
867         # CACHING LOGIC
868         # 1. Check if the widget already exists in our cache
869         if widget_name in self.cached_construction_widgets:
870             print(f"[CACHE] Retrieving existing widget for: {
871                 widget_name}")
872             widget = self.cached_construction_widgets[widget_name]
873         else:
874             # 2. If not, create a new one and cache it
875             print(f"[CACHE] Creating new widget for: {widget_name}")
876             widget = self.widget_map[widget_name](database=self.
877                 database_manager, parent=self)
878
879             widget.next.connect(self.next_widget)
880             widget.back.connect(self.prev_widget)
881             self.cached_construction_widgets[widget_name] = widget
882
883             # ALWAYS Update Project ID (Case: User creates project AFTER
884             # opening widget)
885             if self.current_project_id and hasattr(widget, 'set_project_id'
886                 ):
887                 # Only update if different to avoid redundant reloads
888                 current_widget_id = getattr(widget, 'project_id', None)
889                 if current_widget_id != self.current_project_id:
890                     print(f"[CACHE] Updating Project ID for {widget_name}
891                         to {self.current_project_id}")
892                     widget.set_project_id(self.current_project_id)
893
894             # 3. Add the widget (either retrieved or new) to the tab view
895             index = self.current_right_widget.add_new_tab(widget,
896                 widget_name)
897             self.active_tab_widgets[widget_name] = index

```

```

890
891 # --- Standard Widget Toggle Functions ---
892 def show_tutorial_widget(self):
893     if self.current_left_widget:
894         self.left_panel_placeholder.layout().removeWidget(self.
            current_left_widget)
895         self.current_left_widget.setParent(None)
896     self.current_left_widget = TutorialWidget()
897     self.left_panel_placeholder.layout().addWidget(self.
            current_left_widget)
898     self.current_left_widget.closed.connect(self.remove_left_widget
        )
899
900 def show_results_widget(self):
901     if self.current_right_widget:
902         self.right_panel_placeholder.layout().removeWidget(self.
            current_right_widget)
903         self.current_right_widget.setParent(None)
904     self.current_right_widget = ResultsWidget()
905     self.right_panel_placeholder.layout().addWidget(self.
            current_right_widget)
906     self.current_right_widget.closed.connect(self.
            remove_right_widget)
907
908 def show_comparison_widget(self):
909     if self.current_right_widget:
910         self.right_panel_placeholder.layout().removeWidget(self.
            current_right_widget)
911         self.current_right_widget.setParent(None)
912     self.current_right_widget = ComparisonWidget()
913     self.right_panel_placeholder.layout().addWidget(self.
            current_right_widget)
914     self.current_right_widget.closed.connect(self.
            remove_right_widget)
915
916 def remove_right_widget(self):
917     # Before destroying the current widget, if it's the Tab
        container,
918     # rescue the cached children so they persist!

```

```

919     if self.tabs_active and self.cached_construction_widgets:
920         print("[CACHE] Detaching widgets to prevent deletion...")
921         for w in self.cached_construction_widgets.values():
922             try:
923                 w.setParent(None)
924             except:
925                 pass
926
927     if self.current_right_widget:
928         self.right_panel_placeholder.layout().removeWidget(self.
929             current_right_widget)
930         self.current_right_widget.setParent(None)
931         self.current_right_widget = None
932
933     def remove_left_widget(self):
934         if self.current_left_widget:
935             self.left_panel_placeholder.layout().removeWidget(self.
936                 current_left_widget)
937             self.current_left_widget.setParent(None)
938             self.current_left_widget = None
939
940     def next_widget(self, widget_name):
941         keys = list(self.widget_map.keys())
942         current_idx = keys.index(widget_name)
943         if current_idx + 1 < len(keys):
944             next_name = keys[current_idx + 1]
945             if self.current_left_widget and hasattr(self.
946                 current_left_widget, 'all_param_buttons'):
947                 if next_name in self.current_left_widget.
948                     all_param_buttons:
949                     self.current_left_widget.all_param_buttons[
950                         next_name].click()
951             self.show_project_detail_widgets(next_name)
952
953     def prev_widget(self, widget_name):
954         keys = list(self.widget_map.keys())
955         current_idx = keys.index(widget_name)
956         if current_idx - 1 >= 0:
957             prev_name = keys[current_idx - 1]

```

```

953         if self.current_left_widget and hasattr(self,
954           current_left_widget, 'all_param_buttons'):
955             if prev_name in self.current_left_widget.
956               all_param_buttons:
957                 self.current_left_widget.all_param_buttons[
958                   prev_name].click()
959                 self.show_project_detail_widgets(prev_name)
960 %----- end code -----

```

5.4 Code Explanation

5.4.1 Entry Point – `open_excel_dialog()`

The Excel import workflow begins when the user clicks the Upload Excel button in the LCCA interface. This action triggers the `open_excel_dialog()` method, which serves as the entry point for the parsing process.

This method performs the following operations:

- Opens a file selection dialog to allow the user to choose an Excel file.
- Verifies that a valid file path is selected.
- Passes the selected file path to the function `parse_and_validate_excel(file_path)`.
- Displays success, warning, or error messages depending on validation results.

This function acts as the interface bridge between the UI layer and the backend parsing logic.

5.4.2 Core Parsing Function – `parse_and_validate_excel(file_path)`

This function contains the primary logic for reading and validating Excel data.

Reading Excel File

The system reads the Excel file using:

```
pd.read_excel(file_path, sheet_name=None)
```

Using `sheet_name=None` ensures that:

- All sheets are loaded dynamically.
- Each sheet is stored as a dictionary entry.
- Multiple construction categories can be processed independently.

Row Classification

The parser iterates row-by-row and categorizes each row into:

- Header rows (identified using the `CID#` prefix)
- Section rows (used for grouping components)
- Data rows (material-level entries)

This classification enables structured extraction even when the Excel format is semi-flexible.

5.4.3 Header Normalization – `clean_header()`

The function `clean_header()` standardizes column names to prevent mismatches caused by formatting variations.

The normalization process:

- Removes prefixes such as `CID#`
- Converts text to lowercase
- Removes extra spaces and special characters
- Converts headers to consistent internal keys

For example:

- `CID# Quantity` becomes `quantity`
- `Rate (INR)` becomes `rate`

This ensures compatibility with internal data models and validation logic.

5.4.4 Validation Mechanism

A structured validation framework is embedded within the parser.

Critical Validations (Import Stops)

The following conditions trigger an error and stop the import process:

- Quantity must be numeric
- Rate must be numeric and non-empty
- Unit must belong to `ALLOWED_UNITS`
- Recyclability percentage must be within 0–100
- Carbon emission and conversion factor must be numeric or marked `not_available`

If any critical rule fails, the system calls `show_error_message()` and terminates processing.

Non-Critical Validations (Warnings)

The following generate warnings but do not stop import:

- Missing rate source (`rate_src`)
- Missing carbon emission source
- Missing scrap rate
- Missing recyclability percentage

Warnings are displayed in a structured dialog for user awareness.

5.4.5 Duplicate Detection Logic

The parser maintains a dictionary-based tracking system for each section.

When duplicate material names are detected:

- If rows are identical, they are merged with a warning.
- If rows conflict (different quantity, rate, etc.), a validation error is generated.

This prevents inconsistent cost calculations and preserves data integrity.

5.4.6 Data Storage and JSON Generation

After successful validation:

1. Data is stored in the project database using `ProjectDataManager.bulk_import_excel_data()`.
2. A temporary JSON file (`temporary_construction_data.json`) is created.

The method `save_temp_data()` ensures persistent storage in the local `temp_file_db` directory.

Temporary file cleanup is handled during application startup to prevent stale data usage.

5.4.7 Cost Computation – `calculate_cost_from_json()`

The cost calculation module reads the generated JSON file and computes:

$$\text{Total Cost} = \sum (\text{Quantity} \times \text{Rate})$$

Costs are grouped into:

- FOUNDATION
- SUB-STRUCTURE
- SUPER-STRUCTURE
- MISCELLANEOUS

The system generates a structured cost breakdown dictionary and calculates the grand total in INR.

5.4.8 UI Mapping – `distribute_imported_data()`

Parsed data is distributed to respective UI tabs using a sheet-to-widget mapping mechanism.

This module:

- Dynamically creates tabs if required

- Maps sheet names to corresponding widgets
- Synchronizes project ID across modules
- Caches widget instances to preserve state

This ensures seamless population of construction data into the application interface.

5.4.9 Construction Locking – `calculate_and_lock_construction_data()`

After cost calculation, this function locks the construction dataset to prevent further modification. This guarantees analytical consistency before proceeding to financial and carbon lifecycle analysis modules.

5.4.10 Summary

The Excel parser implementation integrates file ingestion, structured validation, duplicate detection, persistent storage, cost computation, and UI synchronization into a cohesive and modular workflow. The architecture ensures robustness, maintainability, and computational reliability within the LCCA application.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:		Ritik Kumar	
Project:		Osdag	
Internship:		FOSSEE Winter Fellowship 2024	
DATE	DAY	TASK	Hours Worked
1-Dec-2025	Monday	Requirement analysis for Excel (.xlsx) import functionality in LCCA	4
2-Dec-2025	Tuesday	Designed parser architecture and data flow structure	4
3-Dec-2025	Wednesday	Implemented basic Excel file reading using pandas	5
4-Dec-2025	Thursday	Developed multi-sheet handling mechanism	4
5-Dec-2025	Friday	Implemented row classification (headers, sections, data rows)	5
6-Dec-2025	Saturday	Created header normalization logic	4
7-Dec-2025	Sunday	Developed internal data mapping structure	5
8-Dec-2025	Monday	Implemented quantity and rate validation checks	0
9Dec 2025	Tuesday	Added unit validation and allowed unit control logic	0
10 Dec 2025	Wednesday	Added carbon emission and conversion factor validation	6
11 Dec-2025	Thursday	Developed structured error handling mechanism	5
12 Dec 2025	Friday	Implemented warning system for non-critical validation issues	6
13 Dec 2025	Saturday	Added duplicate material detection logic	5
14 Dec 2025	Sunday	Implemented duplicate merge and conflict resolution logic	6
15 Dec 2025	Monday	Developed JSON export functionality for parsed data	5
16 Dec 2025	Tuesday	Implemented temporary file storage system	0
17 Dec 2025	Wedneday	Connected parser to cost computation module	0
18 Dec 2025	Thursday	Implemented cost calculation logic from parsed JSON	7
19 Dec 2025	Friday	Conducted stage-1 functional testing	6
20 Dec 2025	Saturday	Fixed parsing inconsistencies and edge-case bugs	7
21 Dec 2025	Sunday	Improved validation robustness and input sanitization	6
22 Dec 2025	Monday	Refactored parser for modularity and scalability	7
23 Dec 2025	Tuesday	Conducted full dataset stress testing	6
24 Dec 2025	Wednesday	Performance optimization for large Excel files	6
25 Dec 2025	Thursday	Finalized parser core module	5
26 Dec 2025	Friday	Integrated parser with LCCA backend controller	6

Internship Work Report			
Name:		Ritik Kumar	
Project:		Osdag	
Internship:		FOSSEE Winter Fellowship 2024	
27 Dec 2025	Saturday	Debugged integration-level errors	7
28 Dec 2025	Sunday	Conducted full-system validation testing	6
29 Dec 2025	Monday	Final December review and documentation update	7
30 Dec 2025	Tuesday	Completed base-to-final parser development phase	8
31 Dec 2025	Wednesday	Began compatibility testing across different Excel formats	8
1 Jan 2026	Thursday	Tested multi-sheet variations and structural differences	6
2 Jan 2026	Friday	Verified parser stability under malformed datasets	7
3 Jan 2026	Saturday	Fixed compatibility issues related to header mismatches	6
4 Jan 2026	Sunday	Improved dynamic sheet detection logic	6
5 Jan 2026	Monday	Tested numeric precision and rounding behaviors	7
6 Jan 2026	Tuesday	Conducted edge-case testing for missing parameters	6
7 Jan 2026	Wednesday	Validated duplicate detection under real-world datasets	7
8 Jan 2026	Thursday	Performed regression testing	5
9 Jan 2026	Friday	Verified JSON export compatibility	6
10 Jan 2026	Saturday	Began UI integration with upload button trigger	7
11 Jan 2026	Sunday	Connected parser output to UI widgets	6
12 Jan 2026	Monday	Implemented sheet-to-widget mapping logic	7
13 Jan 2026	Tuesday	Debugged UI population errors	7
14 Jan 2026	Wednesday	Completed parser-to-UI integration testing	6
15 Jan 2026	Thursday	Designed filter logic for section-wise data population	7
16 Jan 2026	Friday	Implemented filtering mechanism for category grouping	6
17 Jan 2026	Saturday	Applied dynamic tab population logic	5
18 Jan 2026	Sunday	Tested filter logic across all construction categories	4
19 Jan 2026	Monday	Refined section-wise filtering logic and optimized category-based data segregation	3
20 Jan 2026	Tuesday	Implemented dynamic UI refresh after filtered data population	4
21 Jan 2026	Wednesday	Debugged widget-level data binding issues during Exc	3
22 Jan 2026	Thursday	Improved synchronization between parser output and project data	3

23 Jan 2026	Friday	Performed integration testing between parser, UI layer, and cost calculation module	5
24 Jan 2026	Saturday	Optimized performance for large Excel datasets during UI rendering	4
25 Jan 2026	Sunday	Conducted end-to-end workflow testing (Upload → Validate → Populate → Calculate)	6
26 Jan 2026	Monday	Resolved minor logical inconsistencies in validation and filtering conditions	2
27 Jan 2026	Tuesday	Refactored UI mapping code for better maintainability and scalability	3
28 Jan 2026	Wednesday	Updated technical documentation for parser-to-UI integration	4
29 Jan 2026	Thursday	Performed regression testing to ensure no impact on existing LCCA modules	5
30 Jan 2026	Friday	Final system stability testing under varied dataset scenarios	3
31 Jan 2026	Saturday	Completed final review, code cleanup, and internship phase closure	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.