FOSSEE Semester Internship Report

On

Design of scripting API and GUI of 3psLCCA

Submitted by

Sumagna Das

2nd Year B.Tech Student, Department of Computer Science and Engineering

Indian Institute of Technology, Bhilai

Bhilai

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Swastik

Parth Karia

June 6, 2026

Acknowledgments

I would like to express my sincere gratitude to FOSSEE for providing me with the oppurtunity to undertake this internship. This experience has helped in my professional growth as well as foster a greater understanding of open-source software.

I would like to express my thanks to Parth Karia, my mentor and a project staff member at the OSDAG team. His guidance and technical knowledge helped in overcoming the challenges I faced during my work as well as learn more about how to work collaboratively with r.

I would also like to acknowledge my colleagues, Harsh Chelani and Navtej, who worked with me during this internship. Their collaboration helped us able to tackle difficult problems and learn from each other simultaneously.

Heartfelt appreciation goes to Prof. Siddhartha Ghosh, Principal Investigator of the OSDAG project, Department of Civil Engineering at IIT Bombay, for his leadership and guidance throughout this fellowship. His vision for advancing educational resources through open-source initiatives has made this learning opportunity possible.

Finally, i would like to extend my gratitude to the entire OSDAG team and my college for making this journey easier, guiding me to the completion of my work.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

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

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

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

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

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

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

1.1.1 ICT Initiatives of MoE

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

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

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

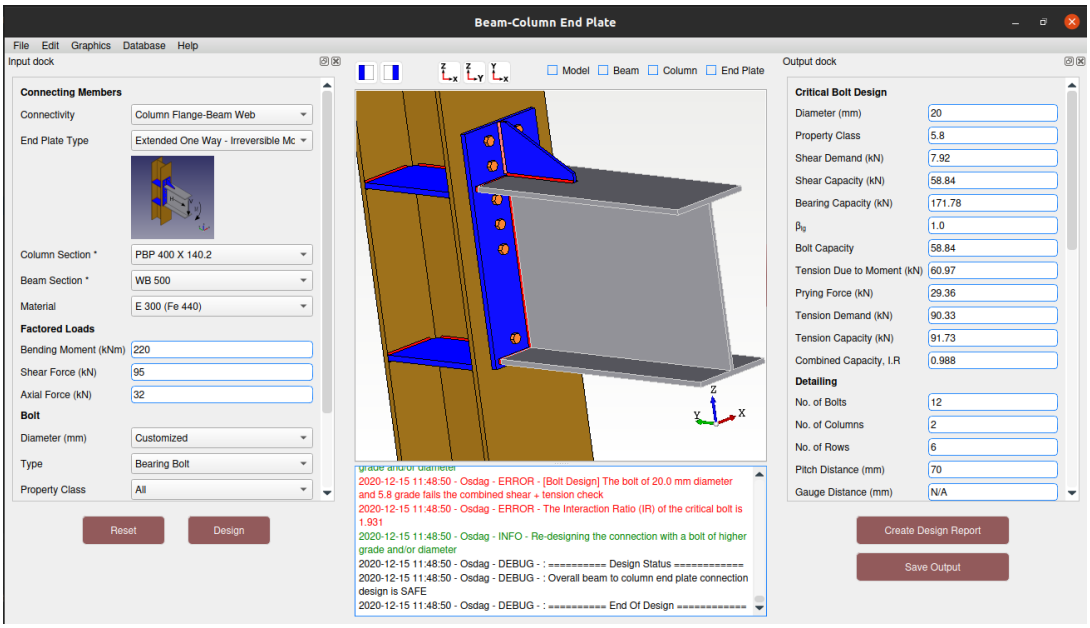

Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Redesign of GUI

2.1 Problem Statement

The GUI for the 3psLCCA software was very primitive, and the design was also outdated. Hence, A full redesign was done with theme system and modular component system, taking reference from Bootstrap. It is fully designed using PySide6, ensuring a modern UI and convenient use.

2.2 Tasks Done

2.2.1 Initial Redesign

This script represents the main starting point for the initial software redesign, adding more functionality and structure to a script designed by my mentor.

Description of the script

The script is structured as follows:

- `ProjectManager`: This is the starting GUI class of the app, where the user can either create a project or select one to open and edit.
- `ProjectWindow`: This is the class that represents the window to work on our project after creating or opening one.
- *GUI Components*: The GUI of the project window as a whole is divided into different components, one example of which is given.

The script is displayed below

Main script

```
import os
import sys
import uuid

from PySide6.QtCore import Qt, QFile, QTextStream
from PySide6.QtWidgets import (
    QApplication,
    QFileDialog,
    QFrame,
    QHBoxLayout,
    QLabel,
    QListWidget,
    QMainWindow,
    QMessageBox,
    QPushButton,
    QStackedWidget,
    QStatusBar,
    QTextEdit,
    QVBoxLayout,
    QWidget,
    QMenuBar,
    QMenu,
    QToolButton,
    QTreeWidget,
    QTreeWidgetItem
)
from PySide6.QtGui import QAction, QIcon

from gui.components.global_info.main import GeneralInfo
from gui.components.bridge_data.main import BridgeData
from gui.components.structure.main import StructureTabView
```

```
from gui.components.traffic_data.main import TrafficData
from gui.components.financial_data.main import FinancialData
from gui.components.carbon_emission.main import
   CarbonEmissionTabView
from gui.components.maintenance.main import Maintenance
from gui.components.recycling.main import Recycling
from gui.components.demolition.main import Demolition
from gui.components.logs import Logs

# Icon file references
# QIcon(":/vectors/new.svg")
# QIcon(":/vectors/open.svg")
# QIcon(":/vectors/save.svg")
# QIcon(":/vectors/save_as.svg")
# QIcon(":/vectors/create_copy.svg")
# QIcon(":/vectors/print.svg")
# QIcon(":/vectors/rename.svg")
# QIcon(":/vectors/export.svg")
# QIcon(":/vectors/version_history.svg")
# QIcon(":/vectors/info.svg")
# QIcon(":/vectors/contact.svg")
# QIcon(":/vectors/feedback.svg")
# QIcon(":/vectors/video_tutorial.svg")
# QIcon(":/vectors/join_community.svg")

class ProjectWindow(QMainWindow):
    def __init__(self, manager):
        super().__init__()
        self.manager = manager
        self.project_id = None
        self.project_path = None

        self.setWindowTitle("LCCA - Home")
        self.resize(1100, 750)
```

```
        # Main window container
        self.main_stack = QStackedWidget()
        self.setCentralWidget(self.main_stack)

        # Build UI pieces
        self.setup_home_ui()
        self.setup_project_ui()

        # Add a status bar to show file path
        self.status_bar = QStatusBar()
        self.setStatusBar(self.status_bar)

        self.show_home()

    def setup_home_ui(self):
        self.home_widget = QWidget()
        layout = QVBoxLayout(self.home_widget)
        layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

        layout.addWidget(QLabel("<h1>LCCA Project Manager</h1>"))

        btn_new = QPushButton("Create New Project")
        btn_new.setFixedSize(220, 45)
        btn_new.clicked.connect(
            lambda: self.manager.open_project(is_new=True, target
                =self)
        )

        btn_open = QPushButton("Open .lcca File")
        btn_open.setFixedSize(220, 45)
        # btn_open.clicked.connect(self._handle_file_picker)
```

```
        self.btn_return = QPushButton("    Return to Active
           Project")
        self.btn_return.setFixedSize(220, 45)
        self.btn_return.setStyleSheet(
            "background-color: #2ecc71; color: white; font-weight
               : bold;"
        )
        self.btn_return.clicked.connect(self.show_project_view)
        self.btn_return.hide()

        layout.addWidget(btn_new)
        layout.addWidget(btn_open)
        layout.addWidget(self.btn_return)
        self.main_stack.addWidget(self.home_widget)

    def setup_project_ui(self):
        self.project_widget = QWidget()
        master_layout = QVBoxLayout(self.project_widget)
        master_layout.setContentsMargins(0, 0, 0, 0)
        master_layout.setSpacing(0)

        #1. Menu bar of the whole project window
        bar = QWidget()
        bar_layout = QHBoxLayout(bar)
        bar_layout.setContentsMargins(4,2,4,2)

        self.menubar = QMenuBar()

        bar_layout.addWidget(self.menubar)
        bar_layout.addStretch()
        bar_layout.addWidget(QPushButton("Save"))
        bar_layout.addWidget(QPushButton("Calculate"))
        bar_layout.addWidget(QPushButton("Lock"))
```

```
        self.menuFile = QMenu("&File", self.menubar)
        self.menuHelp = QMenu("&Help", self.menubar)
        self.menuTut = QAction("&Tutorials", self.menubar)
        logAction = QAction("&Logs", self.menubar)

        home_action = QAction("Home", self)
        home_action.triggered.connect(self.show_home)
        log_window = Logs()
        logAction.triggered.connect(lambda: self.content_stack.
            setCurrentWidget(log_window))
        self.menubar.addAction(home_action)
        self.menubar.addMenu(self.menuFile)
        self.menubar.addMenu(self.menuHelp)
        self.menubar.addAction(self.menuTut)
        self.menubar.addAction(logAction)

        self.actionNew = QAction("New", self)
        self.actionOpen = QAction("Open", self)
        self.actionSave = QAction("Save", self)
        self.actionSaveAs = QAction("Save As...", self)
        self.actionCreateCopy = QAction("Create a Copy", self)
        self.actionPrint = QAction("Print", self)
        self.actionRename = QAction("Rename", self)
        self.actionExport = QAction("Export", self)
        self.actionVersionHistory = QAction("Version History",
            self)
        self.actionInfo = QAction("Info", self)

        self.menuFile.addAction(self.actionNew)
        self.menuFile.addAction(self.actionOpen)
        self.menuFile.addSeparator()
        self.menuFile.addAction(self.actionSave)
        self.menuFile.addAction(self.actionSaveAs)
        self.menuFile.addAction(self.actionCreateCopy)
```

```
        self.menuFile.addAction(self.actionPrint)
        self.menuFile.addSeparator()
        self.menuFile.addAction(self.actionRename)
        self.menuFile.addAction(self.actionExport)
        self.menuFile.addAction(self.actionVersionHistory)
        self.menuFile.addAction(self.actionInfo)

        self.actionDocumentation = QAction("Contact us", self)
        self.actionFeedback = QAction("Feedback", self)
        self.actionVideoTutorial = QAction("Video Tutorials",
           self)
        self.actionJoinCommunity = QAction("Join our Community",
           self)

        self.menuHelp.addAction(self.actionDocumentation)
        self.menuHelp.addAction(self.actionFeedback)
        self.menuHelp.addSeparator()
        self.menuHelp.addAction(self.actionVideoTutorial)
        self.menuHelp.addAction(self.actionJoinCommunity)

        # 2. Workspace Layout
        workspace = QWidget()
        workspace_layout = QHBoxLayout(workspace)
        workspace_layout.setContentsMargins(0, 0, 0, 0)
        workspace_layout.setSpacing(0)

        #3. Sidebar setup
        self.sidebar = QTreeWidget()
        self.sidebar.setHeaderHidden(True)
        self.metadata_page = QLabel()
        self.metadata_page.setAlignment(Qt.AlignmentFlag.
           AlignCenter)

        sidebar_info = {"General Information": {},
```

```
                    "Bridge Data": {},
                    "Input Parameters": {
                        "Construction Work Data": ["
                            Foundation", "Super-Structure", "
                            Substructure", "Miscellaneous"],
                        "Traffic Data": [],
                        "Financial Data": [],
                        "Carbon Emission Data": ["Material
                            Emissions", "Transportation
                            Emissions", "Machinery Emissions",
                             "Traffic Diversion Emissions", "
                            Social Cost of Carbon"],
                        "Maintenance and Repair": [],
                        "Recycling": [],
                        "Demolition": []
                    },
                    "Outputs": {}
                    }
        self.widget_map = { "General Information": GeneralInfo(),
                            "Bridge Data": BridgeData(),
                            "Construction Work Data":
                                StructureTabView(),
                            "Traffic Data": TrafficData(),
                            "Financial Data": FinancialData(),
                            "Carbon Emission Data":
                                CarbonEmissionTabView(),
                            "Maintenance and Repair": Maintenance
                                (),
                            "Recycling": Recycling(),
                            "Demolition": Demolition(),
                            "Outputs": self.metadata_page
                            }

        # Create the sidebar
```

```
        for (header, subheaders) in sidebar_info.items():
            widget = QTreeWidgetItem(self.sidebar)
            widget.setText(0, header)
            for subheader, subitems in subheaders.items():
                subwidget = QTreeWidgetItem(widget)
                subwidget.setText(0, subheader)
                for subitem in subitems:
                    subitem_widget = QTreeWidgetItem(subwidget)
                    subitem_widget.setText(0, subitem)

        # Content stack
        self.content_stack = QStackedWidget()

        # Add all the widgets to the stack from the widget map
        for widget in list(self.widget_map.values()):
            self.content_stack.addWidget(widget)
        self.content_stack.addWidget(log_window)

        # IMPORTANT: Connect AFTER content_stack is defined
        self.sidebar.itemPressed.connect(self.select_sidebar)
        # self.sidebar.currentRowChanged.connect(self.
            content_stack.setCurrentIndex)

        workspace_layout.addWidget(self.sidebar)
        workspace_layout.addWidget(self.content_stack)

        # Add all the widgets to the main layout
        master_layout.setMenuBar(bar)
        master_layout.addWidget(workspace)

        self.main_stack.addWidget(self.project_widget)

    def _handle_file_picker(self):
        file_path, _ = QFileDialog.getOpenFileName(
```

```
            self, "Open LCCA Project", "", "LCCA Files (*.lcca)"
        )
        if file_path:
            self.manager.open_project(filepath=file_path, target=
                self)

    def create_new_project(self):
        self.project_id = f"NEW-{str(uuid.uuid4())[:8]}"
        self.project_path = None
        # self.text_editor.clear()
        self._sync_ui()
        self.show_project_view()

    def _sync_ui(self):
        self.metadata_page.setText(
            f"<h2>Project Metadata</h2><p><b>ID:</b> {self.
                project_id}</p><p><b>File:</b> {self.project_path
                or 'Unsaved'}</p>"
        )
        # self.title_label.setText(f"Active: {self.project_id}")
        self.status_bar.showMessage(f"Path: {self.project_path or
            'Unsaved Project'}")
        self.sidebar.setCurrentItem(self.sidebar.topLevelItem(0))

    def show_home(self):
        self.setWindowTitle("LCCA - Home")
        if self.has_project_loaded():
            self.btn_return.show()
        self.main_stack.setCurrentWidget(self.home_widget)

    def show_project_view(self):
        if self.has_project_loaded():
            self.setWindowTitle(f"LCCA Editor - {self.project_id}
                ")
```

```
            self.main_stack.setCurrentWidget(self.project_widget)

    def select_sidebar(self, item: QTreeWidgetItem):
        header =  item.text(0)
        parent = item.parent()
        item.setExpanded(True)
        if header in self.widget_map.keys():
            self.content_stack.setCurrentWidget(self.widget_map[
               header])
        elif parent:
            parent_header = parent.text(0)
            if parent_header == "Construction Work Data":
                self.content_stack.setCurrentWidget(self.
                   widget_map["Construction Work Data"])
                self.widget_map["Construction Work Data"].
                   select_tab(header)
            elif parent_header == "Carbon Emission Data":
                self.content_stack.setCurrentWidget(self.
                   widget_map["Carbon Emission Data"])
                self.widget_map["Carbon Emission Data"].
                   select_tab(header)

    def has_project_loaded(self):
        return self.project_id is not None

    def closeEvent(self, event):
        self.manager.remove_window(self)
        event.accept()


# --- Project Manager ---


class ProjectManager:
```

```
    def __init__(self):
        self.windows = []

    def get_available_window(self):
        for win in self.windows:
            if not win.has_project_loaded():
                return win
        return None

    def open_project(self, filepath=None, is_new=False, target=
        None):
        # NEW: Check if this file is already open in ANY window
        if filepath:
            for win in self.windows:
                if win.project_path == filepath:
                    # File is already open! Just focus that
                        window.
                    win.show_project_view()  # Ensure it's not on
                         the Home screen
                    win.show()
                    win.raise_()
                    win.activateWindow()
                    return  # Exit without opening a duplicate

        # --- Original Logic Continues ---
        if target is None:
            target = self.get_available_window()

        if target is None or (target.has_project_loaded() and (
            filepath or is_new)):
            target = self.create_window()
            if filepath or is_new:
                self.open_project(filepath, is_new, target)
                return
```

```
        if is_new:
            target.create_new_project()
        else:
            target.show_home()

        target.show()
        target.activateWindow()

    def create_window(self):
        win = ProjectWindow(self)
        self.windows.append(win)
        return win

    def remove_window(self, win):
        if win in self.windows:
            self.windows.remove(win)
        if not self.windows:
            QApplication.quit()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    manager = ProjectManager()

    # file = QFile("gui/assets/themes/lightstyle.qss")
    # if file.open(QFile.ReadOnly | QFile.Text):
    #     stream = QTextStream(file)
    #     stylesheet = stream.readAll()
    #     file.close()
    #     app.setStyleSheet(stylesheet)

    manager.open_project()
    sys.exit(app.exec())
```

Example component (bridge$_{d}ata.py$)

```
from PySide6.QtCore import Qt, Signal
from PySide6.QtGui import QIntValidator, QDoubleValidator
from PySide6.QtWidgets import (QScrollArea, QPushButton, QWidget,
                               QLabel, QVBoxLayout, QGridLayout,
                                  QLineEdit, QComboBox)
import sys

class BridgeData(QWidget):
    next = Signal(str)
    def __init__(self):
        super().__init__()
        self.setObjectName("central_panel_widget")
        left_panel_vlayout = QVBoxLayout(self)
        left_panel_vlayout.setContentsMargins(0, 0, 0, 0)
        left_panel_vlayout.setSpacing(0)

        # --- Scroll Area Setup ---
        self.scroll_area = QScrollArea()
        self.scroll_area.setWidgetResizable(True)
        scroll_content_widget = QWidget()
        scroll_content_widget.setObjectName("
           scroll_content_widget")
        self.scroll_area.setWidget(scroll_content_widget)
        scroll_content_layout = QVBoxLayout(scroll_content_widget
           )
        scroll_content_layout.addStretch(1)

        # --- General Information Form Widget ---
        self.general_widget = QWidget()
        self.general_widget.setObjectName("
           general_info_form_widget")
        grid_layout = QGridLayout(self.general_widget)
        grid_layout.setContentsMargins(30, 20, 30, 20)
```

```
        grid_layout.setHorizontalSpacing(10)
        grid_layout.setVerticalSpacing(10)

        # Bridge type ComboBox
        label = QLabel("Bridge Type *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 0, 0, 1, 1)
        valuer_combo = QComboBox(self.general_widget)
        options = [ "Beam", "Arch", "Truss",
                    "Suspension", "Cable-Stayed",
                    "Box Girder", "Other"
        ]
        valuer_combo.addItems(options)
        grid_layout.addWidget(valuer_combo, 0, 1, 1, 2)

        # Primary Material dropdown
        label = QLabel("Primary Structural Material *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 1, 0, 1, 1)
        valuer_combo = QComboBox(self.general_widget)
        options = [ "RCC", "Prestressed Concrete"
                    "Steel", "Composite", "Timber"
                    "Other"
        ]
        valuer_combo.addItems(options)
        grid_layout.addWidget(valuer_combo, 1, 1, 1, 2)

        # Total Bridge Length
        label = QLabel("Total Bridge Length *")
        label.setObjectName("left_label")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 2, 0, 1, 1)
        input_widget = QLineEdit(self.general_widget)
        input_widget.setValidator(QDoubleValidator())
```

```
        grid_layout.addWidget(input_widget, 2, 1, 1, 2)

        # Span Length
        label = QLabel("Span Length *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 3, 0, 1, 1)
        input_widget = QLineEdit(self.general_widget)
        input_widget.setValidator(QDoubleValidator())
        grid_layout.addWidget(input_widget, 3, 1, 1, 2)

        # Deck Width
        label = QLabel("Deck Width")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 4, 0, 1, 1)
        input_widget = QLineEdit(self.general_widget)
        input_widget.setValidator(QDoubleValidator())
        grid_layout.addWidget(input_widget, 4, 1, 1, 2)

        # Number of lanes
        label = QLabel("Number of Lanes *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 5, 0, 1, 1)
        input_widget = QLineEdit(self.general_widget)
        input_widget.setValidator(QIntValidator(1,10))
        grid_layout.addWidget(input_widget, 5, 1, 1, 1)

        # Latitude
        label = QLabel("Coordinates (Latitude) *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 6, 0, 1, 1)
        input_widget = QLineEdit(self.general_widget)
        input_widget.setValidator(QDoubleValidator(bottom=-90.0,
            top=90.0))
        grid_layout.addWidget(input_widget, 6, 1, 1, 1)
```

```
        # Longitude
        label = QLabel("Coordinates (Longitude) *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 7, 0, 1, 1)
        input_widget = QLineEdit(self.general_widget)
        input_widget.setValidator(QDoubleValidator(bottom=-180.0,
            top=180.0))
        grid_layout.addWidget(input_widget, 7, 1, 1, 1)

        # Design Life of Bridge ComboBox
        label = QLabel("Design Life of Bridge (Years) *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 8, 0, 1, 1)
        valuer_combo = QComboBox(self.general_widget)

        options = [
            "50", "75", "100", "120", "Custom"
        ]
        valuer_combo.addItems(options)

        grid_layout.addWidget(valuer_combo, 8, 1, 1, 1)
        valuer_combo = QComboBox(self.general_widget)

        options = [
            "Months", "Years"
        ]
        valuer_combo.addItems(options)
        grid_layout.addWidget(valuer_combo, 8, 2, 1, 1)

        # Time for Construction
        label = QLabel("Time for Construction *")
        label.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        grid_layout.addWidget(label, 9, 0, 1, 1)
```

```
        input_widget = QLineEdit(self.general_widget)
        grid_layout.addWidget(input_widget, 9, 1, 1, 1)
        scroll_content_layout.insertWidget(0, self.general_widget
            )

        next_btn = QPushButton("Next")
        # next_btn.clicked.connect(self.create_project)
        scroll_content_layout.insertWidget(2, next_btn, alignment
            =Qt.AlignRight)

        left_panel_vlayout.addWidget(self.scroll_area)
        self.bottom_widget = QWidget()
        self.bottom_widget.setObjectName("bottom_widget")

    # def create_project(self):
    #     self.next.emit(KEY_BRIDGE)
```

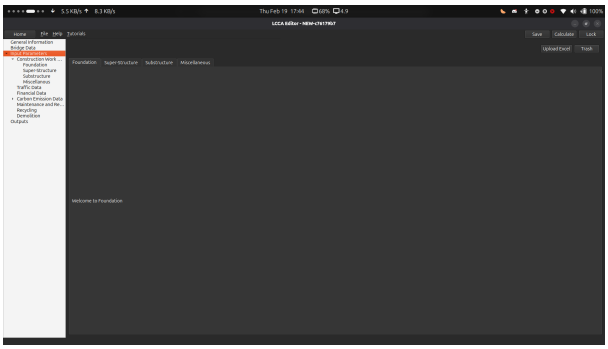

Figure 2.1: Initial Redesign of the GUI

2.2.2 Theme system

After a initial GUI design was made, the theme system was added, using the in-built `QColor` roles for setting a style, while utilizing the stylesheets for sizing and positioning the widgets and its content. Any small changes overriding the default style based on `QColor` roles was done on a per-widget basis.

Description of the system

The color system is designed as follows, based on roles:

- `Window`: Primary background color for the window
- `AlternateBase`: Alternate background color for sidebar and modals
- `Mid`: Border Color
- `Highlight`: Color for highlighting widgets on select
- `Light`: Color for highlighting widgets on hover
- `Button`: Button color
- `Base`: Background color for input fields like text boxes.
- `Text`: Primary text color.
- `HighlightText`: Text color when selected.
- `Text`: Text color inside button.

Code snippet

```
############## Palette Roles #############
# Window           -> Primary Background color
# AlternateBase    -> Secondary Background color (for sidebar,
   modals, etc)
# Mid              -> Borders
# Highlight        -> Highlight on select
# Light            -> Highlight on Hover
# Button           -> Button color
# Base             -> Input field background color
# Text             -> Text color
# ButtonText       -> Button text color
# HighlightedText  -> Text color for selected elements
# Light            -> Text color for elements, highlighted on
   hover



from PySide6.QtGui import QPalette, QColor
```

```
curr_theme = "auto"

accent_dark = QColor("#6B7D20")
accent_light = QColor("#91b014")

##################### DARK THEME PALETTE ####################
dark = QPalette()

dark.setColor(QPalette.Accent, accent_dark)
dark.setColor(QPalette.Window, QColor("#282828"))
dark.setColor(QPalette.AlternateBase, QColor("#333333"))
dark.setColor(QPalette.Mid, QColor("#d6d8d2"))
dark.setColor(QPalette.Highlight, accent_dark)
dark.setColor(QPalette.Light, QColor("#4d4d4d"))
dark.setColor(QPalette.Button, QColor("#282828"))
dark.setColor(QPalette.Base, QColor("#404040"))
dark.setColor(QPalette.Text, QColor("#ffffff"))
dark.setColor(QPalette.HighlightedText, QColor("#000000"))
dark.setColor(QPalette.ButtonText, QColor("#ffffff"))

##################### LIGHT THEME PALETTE ####################
light = QPalette()

light.setColor(QPalette.Accent, accent_light)
light.setColor(QPalette.Window, QColor("#f4f4f4"))
light.setColor(QPalette.AlternateBase, QColor("#fcfcfc"))
light.setColor(QPalette.Mid, accent_light)
light.setColor(QPalette.Highlight, accent_light)
light.setColor(QPalette.Light, QColor("#cccccc"))
light.setColor(QPalette.Button, QColor("#ffffff"))
light.setColor(QPalette.Base, QColor("#ffffff"))
light.setColor(QPalette.Text, QColor("#000000"))
light.setColor(QPalette.HighlightedText, QColor("#000000"))
light.setColor(QPalette.ButtonText, QColor("#000000"))
```

Commit reference: ea1302c0

2.2.3 Bug fixes and improvements

Following bugfixes and improvements were done (in chronological order with commit SHAs) :-

- Button width issues (e9e2dbcd)
- Disabled input selecting text when click, top bar buttons not showing properly and greeting on home page duplication (87eb742b
- Styling related bugs (04fd4ecf)
- Menubar not showing in MacOS (37d83c4d)
- Unresponsive app due to disk I/O for in-function module imports (d6ec54d2)

Chapter 3

Scripting API

3.1 Problem Statement

The 3psLCCA software is divided into three parts - core (for calculations and validation), `SafeChunk Engine`(for project filesystem and save management) and the frontend or GUI. This system is designed in such a way that one can do the calculations programmatically without opening the GUI using only the core module. This headless access requires a scripting API linked to the core module, capable of opening the files and calculating without much effort.

3.2 API design

The main scripting API consists of two classes :-

- `ThreePsAPI` - Main class for scripting
- `DataPreparer` - Class for handling the .3pslcca file data in memory

3.2.1 Input

The following functions can be used to construct an API object as well as set required data :-

- `ThreePsAPI(input_data, lcc_breakdown, wpi_data)` - Constructor which makes an API object for scripting and calculations

- `ThreePsAPI.set_input_data(input_data)` - Sets Input Data
- `ThreePsAPI.set_lcc_breakdown(lcc_breakdown)` - Sets LCC Breakdown data
- `ThreePsAPI.set_wpi_data(wpi_data)` - Sets WPI Data
- `ThreePsAPI.load_3pslcca(path)` - Loads a 3pslcca file in memory to read data from an existing project.

3.2.2 File loading and preparation

After the file is loaded in memory, the `DataPreparer` takes the chunkwise data and outputs 3 data objects :-

- `input_data`
- `wpi_data`
- `lcc_breakdown_data`

All of the data is prepared and formatted according to the proper format required by the core module.

3.2.3 Validation and calculation

After the full data is acquired in a proper format, it can be validated and/or calculated using these functions :-

- `ThreePsAPI.validate()` - Validates input data and returns a report for warnings and errors, if any
- `ThreePsAPI.calculate()` - Calculates and returns the results in JSON format, while also storing it in the object

3.2.4 Output

The calculation data for different stages can be accessed using the following functions :-

- `ThreePsAPI.initial_stage_results()` (Initial Stage)
- `ThreePsAPI.use_stage_results()` (Use Stage)

- `ThreePsAPI.reconstr_stage_results()` (Reconstruction Stage)
- `ThreePsAPI.eol_stage_results()` (End-of-Life Stage)

Chapter 4

Conclusions

4.1 Tasks Accomplished

4.1.1 Redesign of the 3psLCCA software

The GUI of 3psLCCA software was redesigned, with the help of my mentor, to conform to the modern standards as well as ease of use. A new theme system was also added for color customization UI.

4.1.2 Basic implementation for a scripting API

A basic scripting API to do calculations and validations for 3psLCCA was implemented, making sure to sync the format of files and data across the modules to have a seamless experience working with the API.

4.2 Skills Developed

4.2.1 Technical skills

- **Python**: Proficiency in backend refactoring, debugging crashes and undefined behaviour.
- **Scripting API development**: Better understanding of how to create a scripting API for an already existing module and its documentation

4.2.2 Professional skills

- **Problem-Solving**: Complex challenges, including code duplication, UI inconsistencies, and module incompatibilities, were addressed through scalable, innovative solutions.
- **Technical Documentation**: Understanding acquired regarding documentation for written code for readability for those who come after.
- **Collaboration and time management**: Planning with colleagues on various parts of the tasks to ensure code quality and achieve the set goals within deadlines.

Chapter A

Appendix

A.1 Last updated repository links

- 3psLCCA Scripting API
- 3psLCCA-gui repository

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.