FOSSEE Semester Long Internship Report

On

Development of Installer for Osdag

Submitted by

Aathithya Sharan A

3rd Year B.E Student, Department of EC(ACT)

Chennai Institute of Technology

Chennai

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

June 30, 2025

Acknowledgments

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

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

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

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

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

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

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

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

1.1.1 ICT Initiatives of MoE

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

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

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

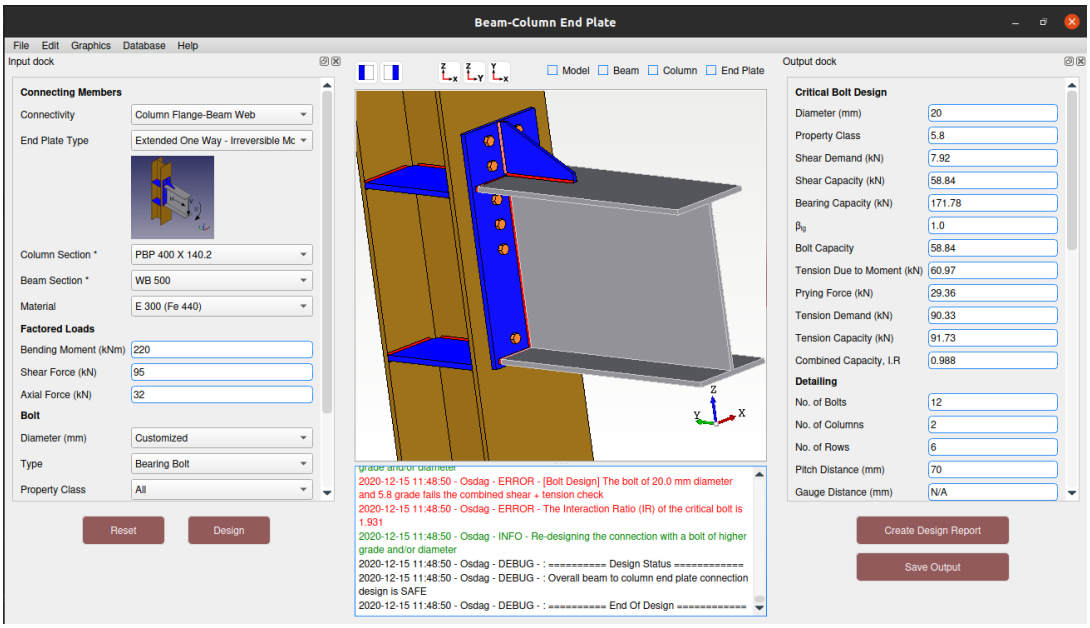

Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

Installing Osdag currently requires users to manually set up several tools like Miniconda, MiKTeX, and Osdag itself. This process can be confusing or difficult for users who are not familiar with command-line tools or software environments.

To make this easier, the task is to create a single executable file for Windows and Linux that can automatically install everything needed to run Osdag. The installer should check if required tools are already installed, install them if needed, and guide the user through any necessary steps—so that the full setup can be completed with very little user effort.

2.2 Tasks Done

2.2.1 Windows Installer (.exe)

- Developed a Windows-compatible `.exe` installer using PyInstaller to automate the installation of Osdag and its dependencies.
- Implemented checks to detect existing installations of Miniconda3 and MiKTeX to avoid redundant installations.
- Automated the download and installation of Miniconda3 and MiKTeX if not already present.

- Allowed user interaction only during the MiKTeX setup wizard, and ensured the process resumes automatically after the wizard closes.
- Used Python's `subprocess` module to execute the official Osdag installation commands non-interactively.
- Created the Conda environment and installed Osdag using commands from the official Osdag Downloads page.
- Ensured clear logging and step-by-step feedback to the user throughout the installation process.
- Created a Start Menu shortcut for Osdag, allowing the user to launch the application directly with a single click.

2.2.2 Output

The result is a single `.exe` file that, when executed, sets up all necessary components for running Osdag on Windows. After successful installation, a shortcut is created in the Start Menu to launch Osdag with a single click.

2.3 Linux Installer (.deb Package)

2.3.1 Tasks Done

- Developed a Debian-compatible `.deb` package to automate the installation of Osdag and its dependencies on Linux.
- Created a Python script `insta-osdag.py` that:
  - Removes old MiKTeX repositories if present.
  - Installs Curl and TeX Live if not already installed.
  - Installs Miniconda and sets up the Conda environment.
  - Installs Osdag using Mamba.
  - Creates a shell launcher script `Launch_Osdag.sh` on the Desktop for easy launch.

- Structured the `.deb` package folder as per Debian packaging standards.
- Wrote appropriate `control` and `rules` files under the `DEBIAN/` directory.
- Ensured the Python installer script is placed in `/usr/local/bin/`.
- Used `dpkg-deb` to build the `.deb` package.

2.3.2 Folder Structure

```
osdag-package
├── DEBIAN
│   ├── control
│   └── rules
├── etc
├── usr
│   └── local
│       └── bin
│           └── insta-osdag.py
└── var
```

2.3.3 Control File Content

Listing 2.1: DEBIAN/control file

```
Package: Osdag-package
Version: 1.0
Section: utils
Priority: optional
Architecture: all
Depends: python3
Maintainer: Aathithya Sharan
Description: A Python-based script for Osdag setup.
```

2.3.4 Rules File Content

Listing 2.2: DEBIAN/rules file

```
#!/usr/bin/make -f
```

```
%:dh $@
override_dh_auto_install:
        cp -r * /usr/local/
        chmod +x /usr/local/bin/insta-osdag.py
override_dh_installinit: true
```

2.3.5 Command Used to Build the Package

After organizing the folder structure correctly, run the following command from the directory one level above `osdag-package/`:

```
dpkg-deb --build osdag-package
```

2.3.6 Output

This command produces a `osdag-package.deb` file. Installing it with

```
sudo dpkg -i osdag-package.deb
```

will:

- Install all required dependencies.
- Create a Conda environment and install Osdag.
- Place a desktop shortcut that allows launching Osdag with a double-click.

2.3.7 Launcher Output

Once installed, a script named `Launch_Osdag.sh` appears on the Desktop. The user can double-click it to activate the Conda environment and launch Osdag easily.

Chapter 3

Internship Task 1: Pixi-based environment Management for Osdag

3.1 Task 1: Problem Statement

Miniconda has a large installation size (over 400 MB) and can be slow with managing complex dependencies.

3.2 Task 1: Tasks Done

Pixi is used as an alternative to Miniconda to manage complex dependencies in Osdag, since Pixi is much smaller (around 15-20 MB) and works faster in managing complex dependencies, it uses simple configuration files, supports quick setup, and helps maintain reliable and consistent environments.

3.3 pixi.toml Manifest file

This section presents a pixi.toml manifest file, which serves as a central configuration for the Osdag project, contains the metadata, dependencies, tasks, and environment definitions.

3.3.1 Description of the Script

The script is structured as follows:

- `[workspace]`: Defines project name, version, channels, authors, and platform targets.
- `[dependencies]`: Lists global dependencies.
- `[feature.<name>.dependencies]`: Lists dependencies specific to a feature (e.g., `editable`, `noneditable`).
- `[feature.<name>.tasks]`: Declare tasks that can be executed conditionally based on the active feature (environment).
- `[environments]`: Defines named environments and associates them with features.

3.3.2 Pixi.toml

Below is the `pixi.toml` file used in the Osdag project: The pixi.toml file, when executed with appropriate commands, will create the appropriate environments. and launch osdag from the respective environment based on the command run.

```
[workspace]
name = "Osdag"
version = "1.0.0"
description = "pixi instructions for osdag"
authors = ["Aathithya Sharan"]
channels = ["conda-forge", "osdag"]
platforms = ["win-64"]

[dependencies]
python = "3.10.*"
numpy  = "2.2.5"
openpyxl = "3.1.5"
pandas = "2.2.3"
pynput = "1.8.1"
pyqt = "5.15.10"
pythonocc-core = "7.8.1.1"
pylatex = "1.4.2"
pygithub = "2.6.1"
```

```
pyyaml = "6.0.2"
smesh = "9.9.0.0"
tbb = "2022.1.0"

[feature.editable.dependencies]
osdag = "2025.01.a.2"

[feature.noneditable.dependencies]
osdag = "2025.01.a.2"

[feature.editable.tasks]
install-editable = "pip install -e ."
osdag-editable = "osdag"

[feature.noneditable.tasks]
osdag = "osdag"

[environments]
dev = ["editable"]
default = ["noneditable"]
```

3.3.3 Explanation of the Code

- Line 1-7: The [project] table defines the metadata for the project, where:
  - name = "": Sets the name of the project
  - version = "": Specifies the version of osdag project
  - description = "": This helps with documentation and clarity
  - authors = [""]: Declares the project authors
  - channels = [""]: Lists the Conda channels from which Pixi should fetch packages.
  - platforms = [""]: Defines the target platform for environment creation.
    - platforms["win-64"]: ensures compatibility for 64-bit Windows systems

    - platforms["linux-64"]: ensures compatibility with 64-bit Linux systems
    - platforms["osx-64"]: ensures compatibility with 64-bit macOS systems

- Line 9-21: The table [dependencies] lists all the global dependencies required by osdag.
- Line 23-24: The table [feature.editable.dependencies] lists the dependencies specific to the editable environment (dev).
- Line 26-27: The table [feature.noneditable.dependencies] lists the dependencies specific to the non-editable environment (default).
- Line 29-31: The table [feature.editable.tasks] lists the tasks specific to the editable environment (dev).
- Line 33-34: The table [feature.noneditable.tasks] lists the tasks specific to the non-editable environment (default).
- Line 36-38: The table [environments] defines the separate environments (dev, default) and assigns feature names (editable, noneditable) to the environments.

3.4 Task 1: Documentation

Insert your contribution towards Osdag Developer/ User manual. A sample is given below.

3.4.1 Directory Structure

```
osdag-pixi
├── .pixi
│   └── envs
│       └── default
├── pixi.exe
├── pixi.lock
└── pixi.toml
```

Task execution Calls The following tasks will perform the following operations based on the pixi.toml file attached in section 4.3.2:

- pixi run install-editable: installs osdag with an editable environment (dev).

- pixi run osdag-editable: runs osdag from the editable environment (dev).
- pixi install: installs osdag with a non-editable environment (default).
- pixi run osdag: runs osdag from the non-editable environment (default).

3.5 pixi.lock file

locks the versions of dependencies used in the project, This ensures reproducible environments across different systems

Chapter 4

Internship Task 2: Plugins Integration for Osdag

4.1 Task 2: Problem Statement

Osdag required a mechanism to support the addition of new design modules without modifying the existing codebase. The goal was to develop a solution that allows seamless integration of such modules while maintaining Osdag's structure and performance.

4.2 Task 2:Tasks Done

To enable seamless integration of new design modules in Osdag, I implemented a plugin architecture that allows dynamic discovery and loading of external modules without modifying the core codebase.

4.2.1 Valid plugins

Plugins that are valid will only be loaded by the plugin manager, a valid plugin must contain a register class and a exposed main method that matches the method mentioned in pyproject.toml.

4.2.2 pyproject.toml file

The pyproject.toml file is a standardized configuration file, that defines build system requirements and project metadata in a unified format, In the context of Osdag, pyproject.toml plays a crucial role in registering and managing external plugins using Python entry points. Entry points allow Osdag to dynamically discover and load plugins without requiring hardcoded references. The entry point for the plugin to be registered with and the main class that gets called when loading the plugin should be mentioned is mentioned in the pyproject.toml

4.2.3 Plugin Discovery Mechanism

The plugin manager in Osdag, handles the discovery and loading of plugins in osdag, the plugin manager loads the plugins in 2 ways:

- from entry-point [osdag.plugins]: The plugins when created are registered with the entry point osdag.plugins, the plugin manager loads the plugins from the osdag.plugins entry point via importlib.metadata.
- from (plugins/) directory: The plugins are loaded from the plugins directory.

4.2.4 Activation and Deactivation of plugins

The Activation and Deactivation of plugins are carried out by the respective activate and deactivate methods defined in the plugin. The UI of the plugin contains a toggle switch that enables activation and deactivation of the plugin.

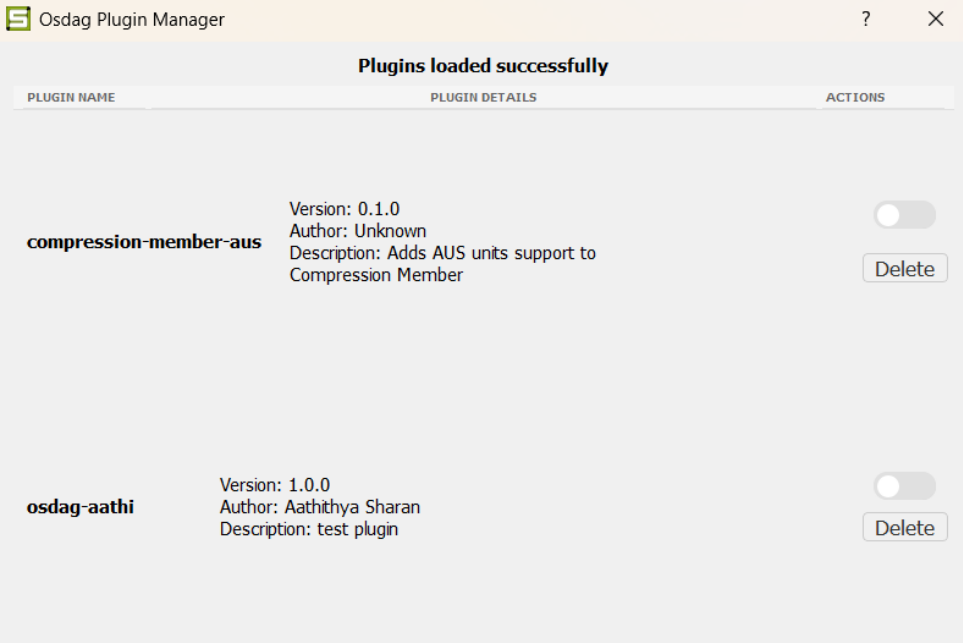

Figure 4.1: Plugin manager showing loaded plugins with toggle switch for activation and deactivation

4.3 Explanation of the Code

The python codes required for creating a plugin and the existing plugin system code to manage ui and handle functions with plugins in osdag are:

4.3.1 pyproject.toml

pyproject.toml plays a crucial role in registering and managing plugins using Python entry points. Entry points allow Osdag to dynamically discover and load plugins.

Description of the Script

The pyproject.toml script is structured as follows:

- The [build-system] section: specifies how the project should be built, it defines which build backend to use and what packages are required to build the project
- The [project] section: contains metadata about your Python project such as its name, version, description, authors, dependencies, and supported Python versions.
- The [project.entry-points.] section: is used to expose specific parts of your project as plugin entry points.

pyproject.toml code

Below is the pyproject.toml file used for the calculator plugin in the Osdag project

```
[build-system]
requires = ["setuptools>=6.10"]
build-backend = "setuptools.build_meta"

[project]
name = "simple-calculator-plugin"
version = "1.0.0"
description = "calculator plugin"
authors = [{name = "Aathithya Sharan"}]
requires-python = ">=3.7"
dependencies = [
    "PyQt5>=5.14.2"
]

[project.entry-points."osdag.plugins"]
simple_calculator = "calculator_plugin:plugin_class"
```

Explanation of Code

- **Line 1-3:** This section specifies that the plugin uses Setuptools as its build backend, with a minimum version of 6.10. `build_meta` is the backend responsible for building the package.

- **Line 5-13:** The `[project]` section defines metadata about the plugin. The `dependencies` list includes external libraries required, such as `PyQt5` for GUI components. This section ensures that the plugin can be correctly built, distributed, and installed using Python packaging tools like `pip`.

- **Line 15-16:** This declares a new entry point under the group `osdag.plugins`, which Osdag scans to load external modules. The entry `simple_calculator = "calculator_plugin:plugin_class"` tells Osdag to import the class `plugin_class` from the module `calculator_plugin` when loading this plugin.

4.3.2 `__init__.py`

The `__init__.py` file initializes the plugin by defining its metadata (name, version, author), lifecycle methods like `register()` and `deactivate()`, and exposes the main plugin class. It enables the plugin to be discovered, loaded, and managed by Osdag.

Description of the Script

The `__init__.py` is structured as follows:

- **Metadata Definition:** Information such as the plugin's name, version, description, and author that identifies the plugin within the host system.
- It can also contain the logic for the plugin to run
- **Plugin Lifecycle Methods:** Functions like `register()` and `deactivate()` that control how the plugin behaves when it is activated or removed. These ensure proper integration and cleanup.
- **Entry Point Exposure:** A designated class or object is exposed to make the plugin available through Python packaging systems, such as the `entry_points` in `pyproject.toml`.

`__init__.py` code

Below is the `__init__.py` file used for the calculator plugin in the Osdag project

```
from PyQt5 import QtWidgets, QtCore, QtGui
from math import sqrt, sin, cos, tan, radians


class CalculatorWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Scientific Calculator")
        self.setWindowIcon(self.style().standardIcon(QtWidgets.
           QStyle.SP_ComputerIcon))
        self.setFixedSize(420, 580)
        self.setup_ui()
```

```
        self.setup_buttons()
        self.current_input = ''
        self.stored_value = 0
        self.current_operation = None
        self.should_clear = False
        self.decimal_added = False

    def setup_ui(self):
        self.layout = QtWidgets.QVBoxLayout(self)
        self.layout.setSpacing(8)
        self.layout.setContentsMargins(10, 5, 10, 10)

        self.display = QtWidgets.QLineEdit('0')
        self.display.setReadOnly(True)
        self.display.setAlignment(QtCore.Qt.AlignRight)
        self.display.setMaxLength(20)
        font = self.display.font()
        font.setPointSize(26)
        font.setBold(True)
        self.display.setFont(font)
        self.display.setMinimumHeight(60)
        self.display.setStyleSheet('''
            QLineEdit {
                background-color: white;
                border: 2px solid #aaa;
                border-radius: 8px;
                padding: 10px 15px;
                color: #333;
                margin-bottom: 5px;
            }
        ''')

        self.memory_display = QtWidgets.QLabel('')
        self.memory_display.setAlignment(QtCore.Qt.AlignRight)
```

```
        memory_font = self.memory_display.font()
        memory_font.setPointSize(11)
        self.memory_display.setFont(memory_font)
        self.memory_display.setStyleSheet('''
            color: #555;
            font-weight: bold;
            margin-bottom: 2px;
            padding: 0 5px;
        ''')
        self.memory_display.setAlignment(QtCore.Qt.AlignRight |
           QtCore.Qt.AlignVCenter)

        self.buttons_layout = QtWidgets.QGridLayout()
        self.buttons_layout.setSpacing(6)  # Slightly reduced
           spacing

        self.layout.addWidget(self.memory_display)
        self.layout.addWidget(self.display)
        self.layout.addLayout(self.buttons_layout)

    def setup_buttons(self):
        buttons = [
            ('C', 1, 0, 1, 1, ''),
            ('  ', 1, 1, 1, 1, ''),
            ('%', 1, 2, 1, 1, ''),
            ('  ', 1, 3, 1, 1, ''),

            ('7', 2, 0, 1, 1, ''),
            ('8', 2, 1, 1, 1, ''),
            ('9', 2, 2, 1, 1, ''),
            ('  ', 2, 3, 1, 1, ''),

            ('4', 3, 0, 1, 1, ''),
            ('5', 3, 1, 1, 1, ''),
```

```
            ('6', 3, 2, 1, 1, ''),
            ('-', 3, 3, 1, 1, ''),

            ('1', 4, 0, 1, 1, ''),
            ('2', 4, 1, 1, 1, ''),
            ('3', 4, 2, 1, 1, ''),
            ('+', 4, 3, 1, 1, ''),

            ('0', 5, 0, 1, 2, ''),
            ('.', 5, 2, 1, 1, ''),
            ('=', 5, 3, 1, 1, ''),

            ('sin', 0, 0, 1, 1, ''),
            ('cos', 0, 1, 1, 1, ''),
            ('tan', 0, 2, 1, 1, ''),
            ('   ', 0, 3, 1, 1, ''),

            ('  ', 6, 0, 1, 1, ''),
            ('x  ', 6, 1, 1, 1, ''),
            ('1/x', 6, 2, 1, 1, ''),
            ('   ', 6, 3, 1, 1, ''),
        ]

        for btn_text, row, col, row_span, col_span, style in
            buttons:
            button = QtWidgets.QPushButton(btn_text)
            button.setMinimumSize(60, 60)
            button.setStyleSheet('''
                QPushButton {
                    background-color: #f8f9fa;
                    border: 1px solid #dee2e6;
                    border-radius: 6px;
                    padding: 10px;
                    font-size: 18px;
```

```
                    font-weight: normal;
                    margin: 2px;
                    color: #212529;
                }
                QPushButton:hover {
                    background-color: #e9ecef;
                }
                QPushButton:pressed {
                    background-color: #dee2e6;
                }
            ''')
            button.clicked.connect(self.on_button_click)
            self.buttons_layout.addWidget(button, row, col,
               row_span, col_span)

    def on_button_click(self):
        button = self.sender()
        text = button.text()

        if text.isdigit() or text == '.':
            self.handle_digit_or_decimal(text)
        elif text in ['+', '-', '  ', '  ']:
            self.handle_operation(text)
        elif text == '=':
            self.calculate_result()
        elif text == 'C':
            self.clear_all()
        elif text == '    ':
            self.backspace()
        elif text == '  ':
            self.toggle_sign()
        elif text == '%':
            self.percentage()
        elif text == '    ':
```

```
            self.square_root()
        elif text == 'x  ':
            self.square()
        elif text == '1/x':
            self.reciprocal()
        elif text == '  ':
            self.add_pi()
        elif text in ['sin', 'cos', 'tan']:
            self.trig_function(text)

    def handle_digit_or_decimal(self, text):
        if self.should_clear:
            self.current_input = ''
            self.should_clear = False

        if text == '.':
            if not self.decimal_added:
                if not self.current_input:
                    self.current_input = '0.'
                else:
                    self.current_input += '.'
                self.decimal_added = True
        else:
            if self.current_input == '0' and text == '0':
                return
            if self.current_input == '0':
                self.current_input = text
            else:
                self.current_input += text

        self.display.setText(self.current_input)

    def handle_operation(self, op):
        if self.current_input:
```

```
            if self.current_operation:
                self.calculate_result()
            self.stored_value = float(self.current_input)
            self.current_operation = op
            self.memory_display.setText(f"{self.stored_value} {op
               }")
            self.should_clear = True
            self.decimal_added = False

    def calculate_result(self):
        if not self.current_operation or not self.current_input:
            return

        current_value = float(self.current_input)
        result = 0

        if self.current_operation == '+':
            result = self.stored_value + current_value
        elif self.current_operation == '-':
            result = self.stored_value - current_value
        elif self.current_operation == '  ':
            result = self.stored_value * current_value
        elif self.current_operation == '  ':
            if current_value == 0:
                self.display.setText("Error")
                self.current_input = ''
                self.current_operation = None
                return
            result = self.stored_value / current_value

        if result.is_integer():
            result = int(result)

        self.display.setText(str(result))
```

```
        self.current_input = str(result)
        self.current_operation = None
        self.memory_display.clear()
        self.should_clear = True

    def clear_all(self):
        self.current_input = '0'
        self.stored_value = 0
        self.current_operation = None
        self.should_clear = False
        self.decimal_added = False
        self.display.setText('0')
        self.memory_display.clear()

    def backspace(self):
        if len(self.current_input) > 0:
            if self.current_input[-1] == '.':
                self.decimal_added = False
            self.current_input = self.current_input[:-1]
            if not self.current_input:
                self.current_input = '0'
            self.display.setText(self.current_input)

    def toggle_sign(self):
        if self.current_input and self.current_input != '0':
            if self.current_input[0] == '-':
                self.current_input = self.current_input[1:]
            else:
                self.current_input = '-' + self.current_input
            self.display.setText(self.current_input)

    def percentage(self):
        if self.current_input:
            value = float(self.current_input) / 100
```

```
            if value.is_integer():
                value = int(value)
            self.current_input = str(value)
            self.display.setText(self.current_input)

    def square_root(self):
        if self.current_input:
            try:
                value = float(self.current_input)
                if value < 0:
                    self.display.setText("Error")
                    return
                result = sqrt(value)
                if result.is_integer():
                    result = int(result)
                self.current_input = str(result)
                self.display.setText(self.current_input)
            except:
                self.display.setText("Error")

    def square(self):
        if self.current_input:
            value = float(self.current_input)
            result = value ** 2
            if result.is_integer():
                result = int(result)
            self.current_input = str(result)
            self.display.setText(self.current_input)

    def reciprocal(self):
        if self.current_input and self.current_input != '0':
            try:
                value = float(self.current_input)
                if value == 0:
```

```
                    self.display.setText("Error")
                    return
                result = 1 / value
                if result.is_integer():
                    result = int(result)
                self.current_input = str(result)
                self.display.setText(self.current_input)
            except:
                self.display.setText("Error")

    def add_pi(self):
        self.current_input = str(3.14159265359)
        self.display.setText(self.current_input)

    def trig_function(self, func):
        if self.current_input:
            try:
                value = radians(float(self.current_input))
                if func == 'sin':
                    result = sin(value)
                elif func == 'cos':
                    result = cos(value)
                elif func == 'tan':
                    if cos(value) == 0:
                        self.display.setText("Error")
                        return
                    result = tan(value)

                if abs(result) < 1e-10:
                    result = 0

                if result.is_integer():
                    result = int(result)
```

```
                self.current_input = str(result)
                self.display.setText(self.current_input)
            except:
                self.display.setText("Error")

class ScientificCalculatorPlugin:
    def __init__(self):
        self.name = 'Scientific Calculator'
        self.version = '2.0.0'
        self.description = 'A full-featured scientific calculator
            with basic and advanced functions'
        self.author = 'Aathithya Sharan'
        self.widget = None

    def register(self):
        self.widget = CalculatorWidget()
        self.widget.show()
        return {
            'name': self.name,
            'version': self.version,
            'description': self.description,
            'author': self.author
        }

    def deactivate(self):
        if self.widget:
            self.widget.close()
            self.widget = None

plugin_class = ScientificCalculatorPlugin
```

Explanation of Code

- **CalculatorWidget (Lines 4–258)**

  This class defines the full GUI and logic of the scientific calculator using PyQt5.

  - **Lines 5–22**: The `__init__()` method initializes the calculator widget, sets window properties, and calls setup methods.
  - **Lines 24–61**: The `setup_ui()` method creates the main layout, display area, and memory label.
  - **Lines 63–118**: The `setup_buttons()` method builds the grid of calculator buttons and connects them to the click handler.
  - **Lines 120–256**: These lines define event handlers for all supported operations, including digits, basic math, trigonometric functions, square root, and more.

- **ScientificCalculatorPlugin (Lines 260–278)**

  This class represents the plugin interface that Osdag interacts with.

  - **Lines 261–267**: The `__init__()` method stores metadata such as name, version, and author.
  - **Lines 269–274**: The `register()` method is called when Osdag activates the plugin. It initializes and shows the calculator widget.
  - **Lines 276–278**: The `deactivate()` method safely closes the widget if it's running.

- **plugin_class (Line 280)**

  This line creates an instance of the plugin class and exposes it as `plugin_class`. Osdag uses this instance to load the plugin.

Relation to pyproject.toml:

Osdag uses the entry point defined in `pyproject.toml` to discover and load the plugin:

```
[project.entry-points."osdag.plugins"]
simple_calculator = "calculator_plugin:plugin_class"
```

- `calculator_plugin` refers to the name of the Python module (i.e., the filename) or the main class containing the register method.

- `plugin_class` is the instance of `ScientificCalculatorPlugin` that Osdag uses.

When the plugin is activated, Osdag:

1. Loads the main class `ScientificCalculatorPlugin` which is the `plugin_class`.
2. Calls `plugin_class.register()`, which shows the calculator.
3. On deactivation, it calls `plugin_class.deactivate()` to close the UI.

4.3.3 Plugin Manager

Plugin manager is responsible for discovering and loading plugins to the Osdag application, it loads plugins in two ways, via entry points and from the file directory.

Description of the Script

- Provides a system to discover and load plugins dynamically from multiple sources, including entry points and a local directory.
- Maintains and manages metadata about each plugin, such as name, version, description, and author.
- Ensures loaded plugins conform to expected interfaces by checking for required methods before activation.
- Supports safe activation and deactivation of plugins within the host application.
- Includes functionality to prompt user confirmation before deleting a plugin.
- Handles plugin uninstallation and cleanup, including removing plugin files and directories.
- Supports error handling and informative logging throughout the plugin lifecycle.

Plugin Manager code

Below is the code for plugin manager used in the Osdag project

```
import os
import importlib.util
import sys
from typing import Dict, Any, Optional
from importlib.metadata import entry_points
from dataclasses import dataclass
import subprocess
import shutil
import stat
from pathlib import Path
from PyQt5.QtWidgets import QMessageBox, QPushButton, QVBoxLayout
    , QWidget
from PyQt5.QtCore import Qt

@dataclass
class PluginInfo:
    #storing plugin metadata
    name: str
    version: str
    description: str
    author: str
    module: Any
    location: Optional[str] = None

class PluginManager:
    def __init__(self):
        self.plugins: Dict[str, PluginInfo] = {}

    def load_plugins(self):
        """Load plugins from entry points and local directory."""
        print("Starting plugin loading process...")
        self.plugins.clear()  # Clear existing plugins before
            loading
        self._load_from_entry_points()
```

```
        print("Checking local directory for additional plugins...
            ")
        self._load_from_directory()

        if self.plugins:
            print("Loaded plugins:")
            for name, info in self.plugins.items():
                print(f"- {name} (v{info.version})")
        else:
            print("No plugins were loaded.")

    def _load_from_entry_points(self):
        try:
            print("Searching for entry points in group 'osdag.
                plugins'...")

            # Try using pkg_resources (more compatible with
                different Python versions)
            try:
                import pkg_resources
                entry_points = list(pkg_resources.
                    iter_entry_points(group='osdag.plugins'))
                print(f"Found {len(entry_points)} entry points
                    using pkg_resources")

                for entry_point in entry_points:
                    print(f"Attempting to load plugin from entry
                        point: {entry_point.name}")
                    try:
                        plugin_class = entry_point.load()
                        print(f"Loaded class {plugin_class.
                            __name__} from entry point")
                        plugin_instance = plugin_class()
```

```
                    if hasattr(plugin_instance, 'register'):
                        plugin_info = PluginInfo(
                            name=getattr(plugin_instance, '
                                name', entry_point.name),
                            version=getattr(plugin_instance,
                                'version', '0.1.0'),
                            description=getattr(
                                plugin_instance, 'description'
                                , ''),
                            author=getattr(plugin_instance, '
                                author', 'Unknown'),
                            module=plugin_instance
                        )
                        self.plugins[entry_point.name] =
                            plugin_info
                        print(f"Successfully loaded plugin
                            from entry point: {entry_point.
                            name} v{plugin_info.version}")
                    else:
                        print(f"Plugin {entry_point.name}
                            does not have register() function.
                            ")
                except Exception as e:
                    print(f"Error loading plugin {entry_point
                        .name}: {str(e)}")

            if self.plugins:
                return
        except ImportError:
            print("pkg_resources not available, falling back
                to importlib.metadata")
        except Exception as e:
            print(f"Error using pkg_resources: {str(e)}")
```

```
        # Fallback to importlib.metadata
        try:
            from importlib.metadata import entry_points as
                get_entry_points
            all_entry_points = get_entry_points()

            # Handle different return types from entry_points
                ()
            if isinstance(all_entry_points, dict):
                # Older Python versions return a dict
                osdag_plugins = all_entry_points.get('osdag.
                    plugins', [])
            else:
                # Newer Python versions return an object with
                    select method
                try:
                    osdag_plugins = all_entry_points.select(
                        group='osdag.plugins')
                except AttributeError:
                    # Handle case where entry_points()
                        returns a list of EntryPoint objects
                    osdag_plugins = []
                    for ep in all_entry_points:
                        try:
                            if hasattr(ep, 'group') and ep.
                                group == 'osdag.plugins':
                                osdag_plugins.append(ep)
                        except Exception:
                            pass  # Skip entry points that
                                can't be processed

            print(f"Found {len(osdag_plugins)} entry points
                using importlib.metadata")
```

```
                for plugin_entry in osdag_plugins:
                    print(f"Attempting to load plugin from entry
                        point: {plugin_entry.name}")
                    try:
                        self._load_plugin_from_entry(plugin_entry
                            )
                    except Exception as e:
                        print(f"Error loading plugin {
                            plugin_entry.name}: {str(e)}")
            except Exception as e:
                print(f"Error using importlib.metadata: {str(e)}"
                    )
        except Exception as e:
            print(f"Error discovering plugins via entry points: {
                str(e)}")

    def _load_from_directory(self):
        plugin_dir = os.path.join(os.path.dirname(__file__), '
            plugins')
        print(f"Searching for plugins in directory: {plugin_dir}"
            )

        if not os.path.exists(plugin_dir):
            print(f"Plugin directory not found: {plugin_dir}")
            return

        # Directories to ignore
        ignore_dirs = {'__pycache__', '.egg-info'}

        for plugin_name in os.listdir(plugin_dir):
            if plugin_name in ignore_dirs or plugin_name.endswith
                ('.egg-info'):
                continue
```

```
            plugin_path = os.path.join(plugin_dir, plugin_name)
            if not os.path.isdir(plugin_path):
                continue

            print(f"Found potential plugin directory: {
               plugin_name}")

            # Check for specific plugin subdirectories
            for subdir in os.listdir(plugin_path):
                # Skip non-directories and ignored directories
                if subdir in ignore_dirs or subdir.endswith('.egg
                   -info'):
                    continue

                subdir_path = os.path.join(plugin_path, subdir)
                if not os.path.isdir(subdir_path):
                    continue

                # Check if this subdirectory has __init__.py
                init_file = os.path.join(subdir_path, '__init__.
                   py')
                if os.path.exists(init_file):
                    print(f"Found plugin with __init__.py: {
                       plugin_name}/{subdir}")
                    try:
                        self._try_load_plugin_from_directory(
                           plugin_name, subdir_path)
                    except Exception as e:
                        print(f"Error loading plugin from {
                           plugin_name}/{subdir}: {str(e)}")

    def _try_load_plugin_from_directory(self, plugin_name: str,
       plugin_path: str) -> bool:
```

```
        """Try to load a plugin from a directory. Returns True if
            successful."""
        init_path = os.path.join(plugin_path, '__init__.py')
        if not os.path.exists(init_path):
            print(f"No __init__.py found in {plugin_path}")
            return False

        try:
            print(f"Attempting to load plugin from: {plugin_path}
                ")
            spec = importlib.util.spec_from_file_location(
                plugin_name, init_path)
            if spec and spec.loader:
                module = importlib.util.module_from_spec(spec)
                sys.modules[plugin_name] = module
                spec.loader.exec_module(module)

                plugin_class = None
                if hasattr(module, 'plugin_class'):
                    plugin_class = getattr(module, 'plugin_class'
                        )
                else:
                    for attr_name in dir(module):
                        attr = getattr(module, attr_name)
                        if isinstance(attr, type) and hasattr(
                            attr, 'register'):
                            plugin_class = attr
                            break

                if plugin_class:
                    try:
                        plugin_instance = plugin_class()
                        # Set initial state to inactive - do not
                            auto-activate
```

```
                    plugin_instance.is_active = False

                    # Additional validation to ensure this is
                        a proper plugin
                    if not hasattr(plugin_instance, 'register
                        ') or not callable(plugin_instance.
                        register):
                        print(f"Skipping invalid plugin {
                            plugin_name}: missing register()
                            method")
                        return False
                except Exception as e:
                    print(f"Error initializing plugin {
                        plugin_name}: {str(e)}")
                    return False

                if hasattr(plugin_instance, 'register'):
                    # Get the root plugin directory (parent
                        of the current path)
                    root_plugin_dir = os.path.dirname(
                        plugin_path)
                    plugin_info = PluginInfo(
                        name=getattr(plugin_instance, 'name',
                            plugin_name),
                        version=getattr(plugin_instance, '
                            version', '0.1.0'),
                        description=getattr(plugin_instance,
                            'description', ''),
                        author=getattr(plugin_instance, '
                            author', 'Unknown'),
                        module=plugin_instance,
                        location=root_plugin_dir  # Store the
                            root plugin directory
                    )
```

```
                        self.plugins[plugin_name] = plugin_info
                        print(f"Successfully loaded plugin: {
                            plugin_name} v{plugin_info.version}")
                        return True
                    else:
                        print(f"Plugin class in {plugin_name}
                            does not have register() method")
                else:
                    print(f"No valid plugin class found in {
                        plugin_name}")
            else:
                print(f"Could not load specification for plugin:
                    {plugin_name}")
        except Exception as e:
            print(f"Error loading plugin {plugin_name}: {str(e)}"
                )
            print(f"Full plugin path: {plugin_path}")
        return False

    def _load_plugin_from_entry(self, plugin_entry):
        try:
            print(f"Loading entry point: {plugin_entry.name} = {
                plugin_entry.value}")
            plugin_class = plugin_entry.load()
            print(f"Loaded class {plugin_class.__name__} from
                entry point")
            plugin_instance = plugin_class()

            if hasattr(plugin_instance, 'register'):
                plugin_info = PluginInfo(
                    name=getattr(plugin_instance, 'name',
                        plugin_entry.name),
                    version=getattr(plugin_instance, 'version', '
                        0.1.0'),
```

```
                    description=getattr(plugin_instance, '
                        description', ''),
                    author=getattr(plugin_instance, 'author', '
                        Unknown'),
                    module=plugin_instance
                )
                self.plugins[plugin_entry.name] = plugin_info
                print(f"Successfully loaded plugin from entry
                    point: {plugin_entry.name} v{plugin_info.
                    version}")
            else:
                print(f"Plugin {plugin_entry.name} does not have
                    register() function.")
        except Exception as e:
            print(f"Error loading entry point {plugin_entry.name
                }: {str(e)}")
            raise

    def get_plugin_info(self, plugin_name: str) -> PluginInfo:
        return self.plugins.get(plugin_name)

    def _show_delete_confirmation(self, plugin_name: str):
        """Show a confirmation dialog before deleting a plugin.
            """
        reply = QMessageBox.question(
            None,
            'Delete Plugin',
            f'Are you sure you want to delete the plugin "{
                plugin_name}"?',
            QMessageBox.Yes | QMessageBox.Cancel,
            QMessageBox.Cancel
        )

        if reply == QMessageBox.Yes:
```

```
            self._delete_plugin(plugin_name)

    def _delete_plugin(self, plugin_name: str):
        """Delete a plugin and its directory."""
        plugin_info = self.get_plugin_info(plugin_name)
        if not plugin_info:
            print(f"Plugin {plugin_name} not found")
            return False

        try:
            # Uninstall using pip
            result = subprocess.run(
                ['pip', 'uninstall', '-y', plugin_name],
                capture_output=True,
                text=True
            )

            if result.returncode == 0:
                print(f"Successfully uninstalled {plugin_name}")

                # Remove plugin directory if available
                if plugin_info.location and os.path.exists(
                   plugin_info.location):
                    try:
                        def handle_remove_readonly(func, path,
                           exc):
                            if not os.access(path, os.W_OK):
                                os.chmod(path, stat.S_IWRITE)
                                func(path)
                            else:
                                raise

                        plugin_dir = os.path.join(plugin_info.
                           location)
```

```
                    if os.path.exists(plugin_dir):
                        shutil.rmtree(plugin_dir,
                            ignore_errors=False, onerror=
                            handle_remove_readonly)
                        print(f"Successfully removed plugin
                            directory: {plugin_dir}")

                except Exception as e:
                    print(f"Error removing plugin directory:
                        {e}")
                    return False

            # Remove from plugin list
            if plugin_name in self.plugins:
                del self.plugins[plugin_name]

            return True
        else:
            print(f"Failed to uninstall plugin: {result.
                stderr}")
            return False
    except Exception as e:
        print(f"Error deleting plugin: {str(e)}")
        return False
```

Explanation of Code

- **Lines 1–21**: Import necessary modules and define the `PluginInfo` dataclass to hold plugin metadata.
- **Lines 23–71**: Define the `PluginManager` class constructor and the main `load_plugins` method, which clears previous plugins and initiates loading from entry points and local directories.
- **Lines 73–141**: Implement `_load_from_entry_points` method to discover plugins

registered via entry points using both `pkg_resources` and `importlib.metadata`, with fallback mechanisms and detailed logging.

- **Lines 143–182**: Implement `_load_from_directory` method that scans a specific local plugins directory for potential plugin folders and attempts to load them.
- **Lines 184–242**: Implement `_try_load_plugin_from_directory` method which attempts to import and validate plugin modules from given directory paths, checking for expected interfaces.
- **Lines 244–263**: Implement `_load_plugin_from_entry` helper method that loads a plugin from a single entry point, instantiating it and extracting metadata.
- **Lines 265–268**: Provide `get_plugin_info` method to retrieve metadata for a loaded plugin by name.
- **Lines 270–282**: Implement `_show_delete_confirmation` method to display a user prompt before deleting a plugin.
- **Lines 284–337**: Implement `_delete_plugin` method that handles plugin uninstallation through pip, deletes plugin files/directories safely, and updates the plugin registry accordingly.

4.3.4 main.py

The main.py code defines a PyQt5-based dialog window to manage plugins in an application. It initializes a plugin manager to load available plugins and displays them with toggle switches for activation and buttons for deletion. The user can activate or deactivate plugins, with status updates shown in the UI. Plugin deletion includes a confirmation prompt and removes the plugin both from UI and system.

Description of script

- Provides a graphical interface for managing software plugins in an application.
- Allows users to view, activate, deactivate, and delete plugins interactively.
- Communicates status updates to keep users informed about plugin operations.

- Ensures safe and user-confirmed removal of unwanted plugins.
- Serves as the main entry point to launch the plugin management interface.

main.py code

Below is the code for main.py used in the Osdag project

```
import sys
import os
from PyQt5 import QtWidgets
from osdag.gui.ui_plugins import Ui_PluginsDialog
from osdag.data.osdag_plugins.plugin_manager import PluginManager

class MainWindow(QtWidgets.QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_PluginsDialog()
        self.ui.setupUi(self)

        print("Initializing plugin manager...")
        self.plugin_manager = PluginManager()
        self.plugin_manager.load_plugins()

        self.load_plugins()

    def load_plugins(self):
        try:
            self.ui.clearPlugins()

            plugins = self.plugin_manager.plugins

            for plugin_name, plugin_info in plugins.items():
                toggle_switch, delete_btn, _ = self.ui.addPlugin(
                    plugin_name,
                    f"Version: {plugin_info.version}\n"
```

```
                    f"Author: {plugin_info.author}\n"
                    f"Description: {plugin_info.description}"
                )

                # Connect toggle switch to handle both activate
                    and deactivate
                toggle_switch.toggled.connect(
                    lambda checked, name=plugin_name: (
                        self.activate_plugin(name) if checked
                        else self.deactivate_plugin(name)
                    )
                )
                delete_btn.clicked.connect(lambda _, name=
                    plugin_name: self.delete_plugin(name))

            self.ui.status_label.setText("Plugins loaded
                successfully")

        except Exception as e:
            self.ui.status_label.setText(f"Error loading plugins:
                {str(e)}")

    def activate_plugin(self, plugin_name):
        try:
            plugin_info = self.plugin_manager.get_plugin_info(
                plugin_name)
            if plugin_info:
                if not hasattr(plugin_info.module, 'is_active')
                    or not plugin_info.module.is_active:
                    plugin_info.module.register()
                    plugin_info.module.is_active = True
                    self.ui.status_label.setText(f"Plugin '{
                        plugin_info.name}' v{plugin_info.version}
                        activated successfully!")
```

```
            else:
                self.ui.status_label.setText(f"Plugin '{
                    plugin_name}' not found")
        except Exception as e:
            self.ui.status_label.setText(f"Error activating {
                plugin_name}: {str(e)}")

    def deactivate_plugin(self, plugin_name):
        try:
            plugin_info = self.plugin_manager.get_plugin_info(
                plugin_name)
            if plugin_info:
                if hasattr(plugin_info.module, 'deactivate'):
                    if hasattr(plugin_info.module, 'is_active')
                        and plugin_info.module.is_active:
                        plugin_info.module.deactivate()
                        plugin_info.module.is_active = False
                        self.ui.status_label.setText(f"Plugin '{
                            plugin_info.name}' v{plugin_info.
                            version} deactivated successfully!")
                else:
                    self.ui.status_label.setText(f"Plugin '{
                        plugin_info.name}' does not support
                        deactivation")
            else:
                self.ui.status_label.setText(f"Plugin '{
                    plugin_name}' not found")
        except Exception as e:
            self.ui.status_label.setText(f"Error deactivating {
                plugin_name}: {str(e)}")

    def delete_plugin(self, plugin_name):
        try:
            reply = QtWidgets.QMessageBox.question(
```

```
                self,
                'Delete Plugin',
                f'Are you sure you want to delete the plugin "{
                    plugin_name}"?',
                QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox
                    .Cancel,
                QtWidgets.QMessageBox.Cancel
            )

            if reply == QtWidgets.QMessageBox.Yes:
                self.ui.clearPlugins()

                success = self.plugin_manager._delete_plugin(
                    plugin_name)

                if success:
                    self.load_plugins()
                    self.ui.status_label.setText(f"Plugin '{
                        plugin_name}' has been successfully
                        deleted.")
                else:import sys
import os
from PyQt5 import QtWidgets
from osdag.gui.ui_plugins import Ui_PluginsDialog
from osdag.data.osdag_plugins.plugin_manager import PluginManager

class MainWindow(QtWidgets.QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_PluginsDialog()
        self.ui.setupUi(self)

        print("Initializing plugin manager...")
        self.plugin_manager = PluginManager()
```

```
        self.plugin_manager.load_plugins()

        self.load_plugins()

    def load_plugins(self):
        try:
            self.ui.clearPlugins()

            plugins = self.plugin_manager.plugins

            for plugin_name, plugin_info in plugins.items():
                toggle_switch, delete_btn, _ = self.ui.addPlugin(
                    plugin_name,
                    f"Version: {plugin_info.version}\n"
                    f"Author: {plugin_info.author}\n"
                    f"Description: {plugin_info.description}"
                )

                # Connect toggle switch to handle both activate
                   and deactivate
                toggle_switch.toggled.connect(
                    lambda checked, name=plugin_name: (
                        self.activate_plugin(name) if checked
                        else self.deactivate_plugin(name)
                    )
                )
                delete_btn.clicked.connect(lambda _, name=
                   plugin_name: self.delete_plugin(name))

            self.ui.status_label.setText("Plugins loaded
               successfully")

        except Exception as e:
            self.ui.status_label.setText(f"Error loading plugins:
```

```
                {str(e)}")

    def activate_plugin(self, plugin_name):
        try:
            plugin_info = self.plugin_manager.get_plugin_info(
                plugin_name)
            if plugin_info:
                if not hasattr(plugin_info.module, 'is_active')
                    or not plugin_info.module.is_active:
                    plugin_info.module.register()
                    plugin_info.module.is_active = True
                    self.ui.status_label.setText(f"Plugin '{
                        plugin_info.name}' v{plugin_info.version}
                        activated successfully!")
            else:
                self.ui.status_label.setText(f"Plugin '{
                    plugin_name}' not found")
        except Exception as e:
            self.ui.status_label.setText(f"Error activating {
                plugin_name}: {str(e)}")

    def deactivate_plugin(self, plugin_name):
        try:
            plugin_info = self.plugin_manager.get_plugin_info(
                plugin_name)
            if plugin_info:
                if hasattr(plugin_info.module, 'deactivate'):
                    if hasattr(plugin_info.module, 'is_active')
                        and plugin_info.module.is_active:
                        plugin_info.module.deactivate()
                        plugin_info.module.is_active = False
                        self.ui.status_label.setText(f"Plugin '{
                            plugin_info.name}' v{plugin_info.
                            version} deactivated successfully!")
```

```
                else:
                    self.ui.status_label.setText(f"Plugin '{
                        plugin_info.name}' does not support
                        deactivation")
            else:
                self.ui.status_label.setText(f"Plugin '{
                    plugin_name}' not found")
        except Exception as e:
            self.ui.status_label.setText(f"Error deactivating {
                plugin_name}: {str(e)}")

    def delete_plugin(self, plugin_name):
        try:
            reply = QtWidgets.QMessageBox.question(
                self,
                'Delete Plugin',
                f'Are you sure you want to delete the plugin "{
                    plugin_name}"?',
                QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox
                    .Cancel,
                QtWidgets.QMessageBox.Cancel
            )

            if reply == QtWidgets.QMessageBox.Yes:
                self.ui.clearPlugins()

                success = self.plugin_manager._delete_plugin(
                    plugin_name)

                if success:
                    self.load_plugins()
                    self.ui.status_label.setText(f"Plugin '{
                        plugin_name}' has been successfully
                        deleted.")
```

```
                else:
                    self.ui.status_label.setText(f"Failed to
                        delete plugin '{plugin_name}'.")
        except Exception as e:
            self.ui.status_label.setText(f"Error deleting plugin:
                {str(e)}")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
                    self.ui.status_label.setText(f"Failed to
                        delete plugin '{plugin_name}'.")
        except Exception as e:
            self.ui.status_label.setText(f"Error deleting plugin:
                {str(e)}")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Explanation of code

- Line [1-3] Import essential Python modules and PyQt5 components required for GUI functionality.
- Line [4] Import the UI layout class generated from the Qt Designer file for the plugins dialog.
- Line [5] Import the PluginManager class responsible for managing plugin operations.
- Line [7-18] Define the MainWindow class inheriting from QDialog to create the

main plugin management window.

- Line [9-13] Initialize the UI, set up the interface, and instantiate the PluginManager to load available plugins.
- Line [15-31] Define the method to populate the UI with plugins, showing their details and connecting toggle and delete controls.
- Line [17] Clear any existing plugin entries from the UI before loading new ones.
- Line [19-29] Iterate through all loaded plugins, add them to the UI, and set up event handlers for activation, deactivation, and deletion.
- Line [30] Update the status label to indicate successful plugin loading.
- Line [32-34] Handle and display any exceptions that occur during the loading process.
- Line [36-48] Define the method to activate a selected plugin.
- Line [38-45] Check if the plugin exists and is inactive, then call its register function to activate and update the UI status.
- Line [46-48] Handle cases where the plugin is not found or activation fails, updating the status accordingly.
- Line [50-64] Define the method to deactivate a selected plugin.
- Line [52-61] Check if the plugin exists and supports deactivation, call its deactivate method, and update the status.
- Line [62-64] Handle missing deactivation support or plugin not found cases with appropriate status messages.
- Line [66-85] Define the method to delete a plugin after user confirmation.
- Line [68-75] Show a confirmation dialog and, if confirmed, attempt to delete the plugin via the plugin manager.
- Line [77-81] Refresh the plugin list and update the status based on the success or failure of the deletion.

- Line [82-85] Handle exceptions during deletion and display error messages.
- Line [87-92] Standard PyQt5 application setup code: initialize the application, create and show the main window, and start the event loop.

4.3.5 Plugin UI

This plugin ui code creates a graphical user interface for managing plugins. It features a toggle switch for activating or deactivating plugins, alongside options to delete them. It also contains a scrollable area to handle numerous plugins efficiently.

Description of script

- Implements a user interface for managing plugins.
- Provides a toggle switch to activate or deactivate individual plugins.
- Displays plugin names and detailed descriptions in a clean, organized, and scrollable layout.
- Allows users to delete plugins with buttons.

Plugin ui code

Below is the code for plugin ui used in the osdag project

```
    from PyQt5 import QtWidgets, QtCore, QtGui
from PyQt5.QtCore import Qt, QPropertyAnimation, QRect, QSize
from PyQt5.QtGui import QPainter, QColor, QCursor, QFont
from PyQt5.QtWidgets import QApplication, QDialog, QFrame,
   QGridLayout, QHBoxLayout, QLabel, QMessageBox, QPushButton,
   QScrollArea, QVBoxLayout, QWidget
from importlib.resources import files

class AnimatedToggle(QPushButton):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setCheckable(True)
        self.setFixedSize(60, 24)
```

```
        self.setCursor(Qt.PointingHandCursor)

        # Colors
        self._bg_color = QColor('#e0e0e0')  # Light gray
            background when off
        self._circle_color = QColor('#ffffff')  # White circle
        self._active_color = QColor('#4CAF50')  # Green when on
        self._inactive_color = QColor('#e0e0e0')  # Light gray
            when off
        self._circle_padding = 3

        # Animation
        self._circle_position = self._circle_padding
        self.animation = QPropertyAnimation(self, b'
            circle_position', self)
        self.animation.setEasingCurve(QtCore.QEasingCurve.
            InOutCubic)
        self.animation.setDuration(200)  # ms

        # Connect signal
        self.toggled.connect(self.start_animation)

    @QtCore.pyqtProperty(int)
    def circle_position(self):
        return self._circle_position

    @circle_position.setter
    def circle_position(self, pos):
        self._circle_position = pos
        self.update()

    def start_animation(self, checked):
        self.animation.stop()
        self.animation.setStartValue(self._circle_position)
```

```
        if checked:
            self.animation.setEndValue(self.width() - (self.
                height() - 2 * self._circle_padding) - self.
                _circle_padding)
            self._bg_color = self._active_color
        else:
            self.animation.setEndValue(self._circle_padding)
            self._bg_color = self._inactive_color

        self.animation.start()

    def hitButton(self, pos):
        return True

    def paintEvent(self, e):
        # Set up the painter
        p = QPainter(self)
        p.setRenderHint(QPainter.Antialiasing)
        p.setPen(Qt.PenStyle.NoPen)

        # Draw the background with shadow
        shadow_rect = QRect(1, 1, self.width()-2, self.height()
            -2)
        p.setBrush(QColor(0, 0, 0, 30))
        p.drawRoundedRect(shadow_rect, 11, 11)

        # Draw the main background
        rect = QRect(0, 0, self.width()-1, self.height()-1)
        p.setBrush(self._bg_color)
        p.drawRoundedRect(rect, 11, 11)

        # Draw the circle
        circle_rect = QRect(
```

```
            self._circle_position,
            self._circle_padding,
            self.height() - 2 * self._circle_padding,
            self.height() - 2 * self._circle_padding
        )
        p.setBrush(self._circle_color)
        p.drawEllipse(circle_rect)

        p.end()

class Ui_PluginsDialog:
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(800, 500)
        Dialog.setMinimumSize(QtCore.QSize(800, 400))

        # Create main layout
        self.verticalLayout = QtWidgets.QVBoxLayout(Dialog)
        self.verticalLayout.setObjectName("verticalLayout")
        self.verticalLayout.setContentsMargins(10, 10, 10, 10)

        # Add status label at the top
        self.status_label = QtWidgets.QLabel("Select plugins to
           manage")
        self.status_label.setObjectName("status_label")
        self.status_label.setAlignment(QtCore.Qt.AlignCenter)
        font = QtGui.QFont()
        font.setBold(True)
        font.setPointSize(12)
        self.status_label.setFont(font)
        self.verticalLayout.addWidget(self.status_label)

        # Create scroll area for plugins
        self.scrollArea = QtWidgets.QScrollArea(Dialog)
```

```
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setObjectName("scrollArea")
        self.scrollArea.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.scrollAreaWidgetContents = QtWidgets.QWidget()
        self.scrollAreaWidgetContents.setObjectName("
           scrollAreaWidgetContents")
        self.scrollArea.setStyleSheet("""
            QScrollArea {
                border: none;
                background: transparent;
            }
            QScrollBar:vertical {
                border: none;
                background: #f0f0f0;
                width: 10px;
                margin: 0px;
            }
            QScrollBar::handle:vertical {
                background: #c0c0c0;
                min-height: 20px;
                border-radius: 5px;
            }
            QScrollBar::add-line:vertical, QScrollBar::sub-line:
               vertical {
                height: 0px;
            }
        """)

        # Create grid layout for plugins with no spacing
        self.gridLayout = QtWidgets.QGridLayout(self.
           scrollAreaWidgetContents)
        self.gridLayout.setObjectName("gridLayout")
        self.gridLayout.setSpacing(0)  # No spacing between rows
           or columns
```

```
        self.gridLayout.setContentsMargins(5, 1, 5, 1)  # Minimal
            margins (left, top, right, bottom)
        self.gridLayout.setVerticalSpacing(0)  # No vertical
           spacing between rows

        # Set column stretch factors
        self.gridLayout.setColumnStretch(0, 1)  # Name column
        self.gridLayout.setColumnStretch(1, 3)  # Details column
           (more space)
        self.gridLayout.setColumnStretch(2, 1)  # Buttons column

        # Set column minimum widths
        self.gridLayout.setColumnMinimumWidth(0, 120)  # Name
           column
        self.gridLayout.setColumnMinimumWidth(1, 300)  # Details
           column
        self.gridLayout.setColumnMinimumWidth(2, 100)  # Buttons
           column

        # Add header labels with proper styling
        # Ultra-thin header row
        header_row = QtWidgets.QFrame()
        header_row.setFrameShape(QFrame.StyledPanel)
        header_row.setFixedHeight(20)  # Fixed small height
        header_row.setStyleSheet("""
            QFrame {
                background: #f5f5f5;
                border: none;
                border-bottom: 1px solid #e0e0e0;
                margin: 0;
                padding: 0;
                font-size: 9px;
                font-weight: bold;
                text-transform: uppercase;
```

```
            color: #666;
        }
    """)
    header_layout = QtWidgets.QHBoxLayout(header_row)
    header_layout.setContentsMargins(8, 0, 8, 0)  # Minimal
        vertical padding
    header_layout.setSpacing(5)  # Minimal spacing between
        items

    # Plugin Name header
    plugin_header = QtWidgets.QLabel("Plugin Name")
    plugin_header.setFixedWidth(100)  # Smaller width

    # Details header - centered
    details_header = QtWidgets.QLabel("Plugin Details")
    details_header.setAlignment(QtCore.Qt.AlignCenter)

    # Actions header - right aligned
    actions_header = QtWidgets.QLabel("Actions")
    actions_header.setFixedWidth(100)  # Fixed width for
        actions

    # Add widgets to header
    header_layout.addWidget(plugin_header)
    header_layout.addWidget(details_header, 1)  # Allow
        details to expand
    header_layout.addWidget(actions_header, 0, Qt.AlignRight)
         # Align actions to right

    # Add header to grid with no extra spacing
    self.gridLayout.addWidget(header_row, 0, 0, 1, 3)
    self.gridLayout.setRowMinimumHeight(0, 20)  # Match fixed
        height
```

```
        # Add scroll area to layout
        self.scrollArea.setWidget(self.scrollAreaWidgetContents)
        self.verticalLayout.addWidget(self.scrollArea)

        # Set window icon and title
        try:
            Dialog.setWindowIcon(QtGui.QIcon(str(files("osdag.
                data.ResourceFiles.images").joinpath("Osdag.png"))
                ))
        except Exception:
            # Fallback if resource file can't be found
            pass
        Dialog.setWindowTitle("Osdag Plugin Manager")

        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def clearPlugins(self):
        """Clear all plugins from the UI."""
        # Skip the header row (first 3 items)
        while self.gridLayout.count() > 3:
            item = self.gridLayout.takeAt(3)
            if item.widget():
                item.widget().deleteLater()

    def toggle_metadata(self, label, button):
        """Toggle between showing full and truncated metadata."""
        if label.is_expanded:
            label.setText(label.full_text[:150] + ("..." if len(
                label.full_text) > 150 else ""))
            button.setText("Read More")
        else:
            label.setText(label.full_text)
            button.setText("Show Less")
```

```
        label.is_expanded = not label.is_expanded
        # Adjust height based on content
        if label.is_expanded:
            label.setFixedHeight(QtWidgets.QWIDGETSIZE_MAX)
        else:
            label.setFixedHeight(50)

    def addPlugin(self, plugin_name, plugin_metadata):
        row = self.gridLayout.rowCount()

        # Create plugin name label with minimal spacing
        name_widget = QtWidgets.QWidget(self.
            scrollAreaWidgetContents)
        name_layout = QtWidgets.QHBoxLayout(name_widget)
        name_layout.setContentsMargins(2, 1, 2, 1)  # Minimal
            margins

        name_label = QtWidgets.QLabel(plugin_name)
        name_label.setObjectName(f"name_{plugin_name}")
        font = QtGui.QFont()
        font.setBold(True)
        name_label.setFont(font)
        name_label.setAlignment(QtCore.Qt.AlignLeft | QtCore.Qt.
            AlignVCenter)
        name_layout.addWidget(name_label)
        name_layout.addStretch()

        # Metadata widget with minimal spacing
        metadata_widget = QtWidgets.QWidget(self.
            scrollAreaWidgetContents)
        metadata_layout = QtWidgets.QVBoxLayout(metadata_widget)
        metadata_layout.setContentsMargins(2, 1, 2, 1)  # Minimal
            margins
        metadata_layout.setSpacing(0)  # No spacing between
```

```
        metadata items

    metadata_label = QtWidgets.QLabel(plugin_metadata)
    metadata_label.setObjectName(f"metadata_{plugin_name}")
    metadata_label.setWordWrap(True)
    metadata_label.setAlignment(QtCore.Qt.AlignLeft | QtCore.
        Qt.AlignTop)
    metadata_layout.addWidget(metadata_label)
    metadata_layout.addStretch()

    # Create animated toggle button
    toggle_switch = AnimatedToggle()
    toggle_switch.setObjectName(f"toggle_{plugin_name}")
    toggle_switch.setCheckable(True)

    # Create delete button
    delete_button = QtWidgets.QPushButton("Delete")
    delete_button.setObjectName(f"delete_{plugin_name}")
    delete_button.setFixedSize(80, 30)

    # Create a container widget for the plugin row with
        minimal styling
    row_widget = QtWidgets.QFrame()
    row_widget.setFrameShape(QFrame.StyledPanel)
    row_widget.setStyleSheet("""
        QFrame {
            border: none;
            border-bottom: 1px solid #f0f0f0;
            margin: 0;
            padding: 0;
        }
        QFrame:hover {
            background: #f8f8f8;
        }
```

```
        """)
        row_layout = QtWidgets.QHBoxLayout(row_widget)
        row_layout.setContentsMargins(5, 2, 5, 2)  # Minimal
            padding
        row_layout.setSpacing(10)  # Spacing between elements
        row_layout.setAlignment(Qt.AlignVCenter)  # Vertically
            center contents

        # Add name and metadata to the row
        row_layout.addWidget(name_widget, 1, QtCore.Qt.AlignLeft
            | QtCore.Qt.AlignVCenter)
        row_layout.addWidget(metadata_widget, 3, QtCore.Qt.
            AlignLeft | QtCore.Qt.AlignVCenter)

        # Create a widget for buttons with proper spacing
        buttons_widget = QtWidgets.QWidget()
        buttons_layout = QtWidgets.QHBoxLayout(buttons_widget)
        buttons_layout.setContentsMargins(0, 0, 0, 0)
        buttons_layout.setSpacing(8)
        buttons_layout.addStretch()

        # Style and add buttons
        toggle_switch.setFixedSize(52, 24)
        delete_button.setFixedSize(70, 24)
        delete_button.setStyleSheet("""
            QPushButton {
                background: transparent;
                color: #333;
                border: 1px solid #ccc;
                border-radius: 4px;
                padding: 2px 8px;
            }
            QPushButton:hover {
                background: #f0f0f0;
```

```
            }
            QPushButton:pressed {
                background: #e0e0e0;
            }
        """)

        # Create a vertical layout for the buttons with minimal
            spacing
        button_container = QWidget()
        button_layout = QVBoxLayout(button_container)
        button_layout.setContentsMargins(0, 0, 0, 0)
        button_layout.setSpacing(2)  # Very small space between
            buttons
        button_layout.addWidget(toggle_switch, 0, Qt.AlignCenter)
        button_layout.addWidget(delete_button, 0, Qt.AlignCenter)

        buttons_layout.addWidget(button_container, alignment=Qt.
            AlignRight)

        # Add buttons to row and row to grid
        row_layout.addWidget(buttons_widget, 1, QtCore.Qt.
            AlignRight)
        self.gridLayout.addWidget(row_widget, row, 0, 1, 3)

        # Set column stretch factors and alignment
        self.gridLayout.setColumnStretch(0, 1)
        self.gridLayout.setColumnStretch(1, 2)
        self.gridLayout.setColumnStretch(2, 0)  # Don't stretch
            buttons column

        # No row stretch to prevent extra space
        self.gridLayout.setRowStretch(row, 0)

        # Return only the widgets that need connections
```

```
        return toggle_switch, delete_button, None


if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    Dialog = QDialog()
    ui = Ui_PluginsDialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

- Line [1–8]: Imports necessary PyQt5 modules for GUI components, layout control, painting, animation, and resource management.
- Line [10–80]: Defines the `AnimatedToggle` class, a custom toggle switch with smooth animations and dynamic color changes based on its state.
- Line [82–259]: Implements the `Ui_PluginsDialog` class which builds the main window UI layout for managing plugins.
  - Line [83–142]: Constructs the dialog layout, sets a title label, and creates a scrollable area to list plugins.
  - Line [143–145]: Adds a header row with styled labels for plugin name, metadata, and action buttons.
  - Line [146–163]: Styles the scroll area and defines stretch and minimum widths for responsive layout behavior.
  - Line [164–168]: Attempts to load a custom icon for the window from internal resources, with error handling.
- Line [171–177]: Defines `clearPlugins`, a utility function to remove all dynamically added plugin widgets from the layout.
- Line [179–191]: Implements `toggle_metadata` for toggling between collapsed and expanded views of plugin descriptions.

- Line [193–257]: Defines `addPlugin` which inserts a new row for a plugin, including name, metadata, toggle, and delete functionality with consistent layout styling.
- Line [259–267]: Contains the main execution block: launches the Qt application, displays the plugin manager window, and starts the event loop.

4.4 Task 2: Documentation

Insert your contribution towards Osdag Developer/ User manual. A sample is given below.

4.4.1 Directory Structure

```
osdag
├── data
│   └── osdag_plugins
│       ├── plugins
│       └── plugin_manager.py
│           ├── Calculator-Plugin
│           │   ├── calculator_plugin
│           │   └── __init__.py
│           └── pyproject.toml
└── gui
    ├── main.py
    └── ui_plugins.py
```

Task execution Calls

- User Interaction and UI Trigger: When the user clicks on "Plugins" in the Help menu, the `on_comboBox_help_changed` method in `ui_OsdagMainPage.py` triggers the instantiation of the `MainWindow` class (`gui/main.py`), initializing the plugin manager.
- Plugin Discovery and Loading: The `PluginManager` (`plugin_manager.py`) loads plugins from defined entry points (`pyproject.toml`) and local directories (`data/osdag_plugins/p` by scanning for valid plugin structures (i.e., folders with `__init__.py`), then loads and registers each plugin.
- UI Display and Plugin Control: Loaded plugins are displayed in the UI with their name and description. Users can toggle plugins, triggering activation/deactivation

via `toggle_plugin()`, and calling each plugin's `register()` method to integrate functionality into the main application.

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the initial phase of my internship, I focused on understanding the Osdag codebase, exploring how its various components were interconnected, and gaining insights into the application's architecture and evolution. As I progressed, I worked on migrating Osdag's dependency management from Miniconda to Pixi, providing users with the option to install two separate environments—one for development and another for general use. Following that, I contributed to the design and implementation of the plugin system in Osdag, handling the full workflow of plugin installation, activation, deactivation, and deletion. Throughout the internship, I ensured that every task I completed was thoroughly documented to support clarity, reproducibility, and ease of future maintenance.

5.2 Skills Developed

During the fellowship, I gained valuable technical and professional skills, including:

- **Pixi Packaging and Environment Management:** I learned how to effectively use Pixi for managing environments and packaging, enabling clean isolation of dependencies for development and general-use environments.
- **Python Scripting and Automation:** I enhanced my Python scripting skills by automating various installation and setup tasks, streamlining the deployment process and reducing manual effort.

- **Scripting:** I gained hands-on experience with shell scripting by creating .bat files to execute commands and launch applications, improving usability for end users.
- **Plugin Development and Integration:** I learned how to create, register, and integrate plugins into a modular Python application like Osdag, gaining a deeper understanding of extensibility and dynamic loading mechanisms.
- **PyQt for UI Development:** I developed basic UI components using PyQt, allowing me to contribute to the visual and interactive aspects of the application.

Chapter A

Appendix

A.1 Work Reports

Osdag task datasheet

Date	day	Task	Work Hours
11 February 2025	Tuesday	Installed Osdag / Testing /Raising errors	3 hrs
12 February 2025	Wednesday	Gmeet	1 hr
13 February 2025	Thursday	Understanding the code of Cover Plate bolted connection module	3.5 hrs
14 February	Friday	understanding the code of Cover Plate bolted connection module	2 hrs
15 February 2025	Saturday	preparing Documentation	4 hrs
17 February 2025	Monday	preparing documentation	2 hrs
18 February 2025	Tuesday	-	-
19 February 2025	Wednesday	Gmeet on version control	30 mins
20 February 2025	Thursday	version control tutorials	2.5 hrs
21 February 2025	Friday	Gmeet, latex packages identification	2 hrs
22 February 2025	Saturday	latex packages in one folder	3.5 hrs
24 February 2025	Monday	Change in command line in (reportGenerator_latex.py)	3 hrs
25 February 2025	Tuesday	Environment setup	3 hrs
26 February 2025	Wednesday	Change in command line in (SectionModeller_Latex.py)	3 hrs
27 February 2025	Thursday	identifying more latex packages neede by osdag	3 hrs
28 February 2025	Friday	correcting command line changes in python files	4 hrs
3 March 2025	Monday	command line changes to add pkg location to env var	3.5 hrs
4 March 2025	Tuesday	leave	-
5 March 2025	Wednesday	leave	-
6 March 2025	Thursday	leave	-
7 March 2025	Friday	leave	-
10 March 2025	Monday	leave	-
11 March 2025	Tuesday	leave	-
12 March 2025	Wednesday	testing latex packaging and referencing	3.5 hrs
13 March 2025	Thursday	Gmeet	1 hr
14 March 2025	Friday	Documentation on latex package referencing	2.5 hrs
17 March 2025	Monday	Documentation on latex package referencing	2 hrs
18 March 2025	Tuesday	Gmeet	1 hr
19 March 2025	Wednesday	understanding pixi manifest file	4 hrs
20 March 2025	Thursday	creation of pixi manifest file for osdag	2.5 hrs
21 March 2025	Friday	gmeet + referring to pixi documentation	4 hrs
24 March 2025	Monday	Gmeet	1 hr
25 March 2025	Tuesday	checking pixi's match to osdag	4.5 hrs
26 March 2025	Wednesday	referring to more documentation on pixi	4 hrs
27 March 2025	Thursday	understanding pixi commands	4 hrs
28 March 2025	Friday	understanding package dependecies of osdag	4 hrs
31 March 2025	Monday	creation of pixi manifest file	3.5 hrs
1 April 2025	Tuesday	testing	3 hrs
2 April 2025	Wednesday	Gmeet	1 hr
3 April 2025	Thursday	creation of bat files for launching osdag in pixi env	2 hrs
4 April 2025	Friday	creation of bat files for updating osadg in pix env	2 hrs
7 April 2025	Monday	testing + github repo initialization	4 hrs
8 April 2025	Tuesday	Documentation on pixi	4 hrs
9 April 2025	Wednesday	gmeet	1 hr
10 April 2025	Thursday	understanding python plugins	5.5 hrs
11 April 2025	Friday	creation of files to support plugin	4 hrs
14 April 2025	Monday	sample plugin creation	4.5 hrs
15 April 2025	Tuesday	gmeet	1 hr

16 April 2025	Wednesday	ui changes in plugin manager	5 hrs
17 April 2025	Thursday	entry point overriding check	3 hrs
18 April 2025	Friday	repo initialization	2 hrs
21 April 2025	Monday	leave	-
22 April 2025	Tuesday	leave	-
23 April 2025	Wednesday	leave	-
24 April 2025	Thursday	leave	-
25 April 2025	Friday	leave	-
28 April 2025	Monday	leave	-
29 April 2025	Tuesday	leave	-
30 April 2025	Wednesday	leave	-
1 May 2025	Thursday	gmeet	1 hr
2 May 2025	Friday	understanding python manifest file creation for plugins	4.5 hrs
5 May 2025	Monday	creation of manifest file for osdag plugins + testing	5 hrs
6 May 2025	Tuesday	gmeet	1 hr
7 May 2025	Wednesday	creation of pixi based installation of osdag release version	3 hrs
8 May 2025	Thursday	gmeet + squashing commits + documentation	6 hrs
9 May 2025	Friday	gmeet	1 hr
12 May 2025	Monday	creating calculator plugin	4 hrs
13 May 2025	Tuesday	implementing functionality for calc plugin	4 hrs
14 May 2025	Wednesday	gmeet	1 hr
15 May 2025	Thursday	implemeting ui for calc plugin	4.5 hrs
16 May 2025	Friday	testing + fixing errors	2.5 hrs
19 May 2025	Monday	understading handling of 2 envs in pixi	3.5 hrs
20 May 2025	Tuesday	creation of pixi manifest file to handle 2 env	4 hrs
21 May 2025	Wednesday	testing + correcting errors and simplifying approach	4 hrs
22 May 2025	Thursday	gmeet	1 hr
23 May	Fridat	pixi dual env documentation	3.5 hrs
26 May 2025	Monday	understading methods in submodules for creation of aus plugin	4 hrs
27 May 2025	Tuesday	attempting to override submodule method by aus plugin methods	6 hrs
28 May 2025	Wednesday	gmeet	1 hr
29 May 2025	Thursday	plugins installation implementation	3 hrs
30 May 2025	Friday	github api identification	4 hrs
2 June 2025	Monday	experimenting github api	5 hrs
3 June 2025	Tuesday	github api implementation to plugins installation	3.5 hrs
4 June 2025	Wednesday	install plugins ui integration	4.5 hrs
5 June 2025	Thursday	testing install plugins + fixing errors	4 hrs
6 June 2025	Friday	fixing errors in install plugins functionality	3.5 hrs
9 June 2025	Monday	testing install plugins	3 hrs
10 June	Tuesday	gmeet	1 hr
11 June 2025	Wednesday	understanding module and submodule functionality in osdag	6 hrs
12 June 2025	Thursday	understading submodule ui creation	4.5 hrs
13 June 2025	Friday	creating column submodule plugin	5 hrs
16 June 2025	Monday	changing core osdag file code to support submodule plugin	5 hrs
17 June 2025	Tuesday	testing + fixing errors	6 hrs
18 June 2025	Wednesday	gmeet	1 hr
19 June 2025	Thursday	testing + fixing errors	3.5 hrs
20 June 2025	Friday	documentation to create submodule plugins	3 hrs
23 June 2025	Monday	documentation to create submodule plugins	3 hrs
24 June 2025	Tuesday	github repo initialization	1.5 hrs

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.