FOSSEE Semester Internship Report

On

Refactoring of Osdag and Osdag Web codebase

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:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

February 19, 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.

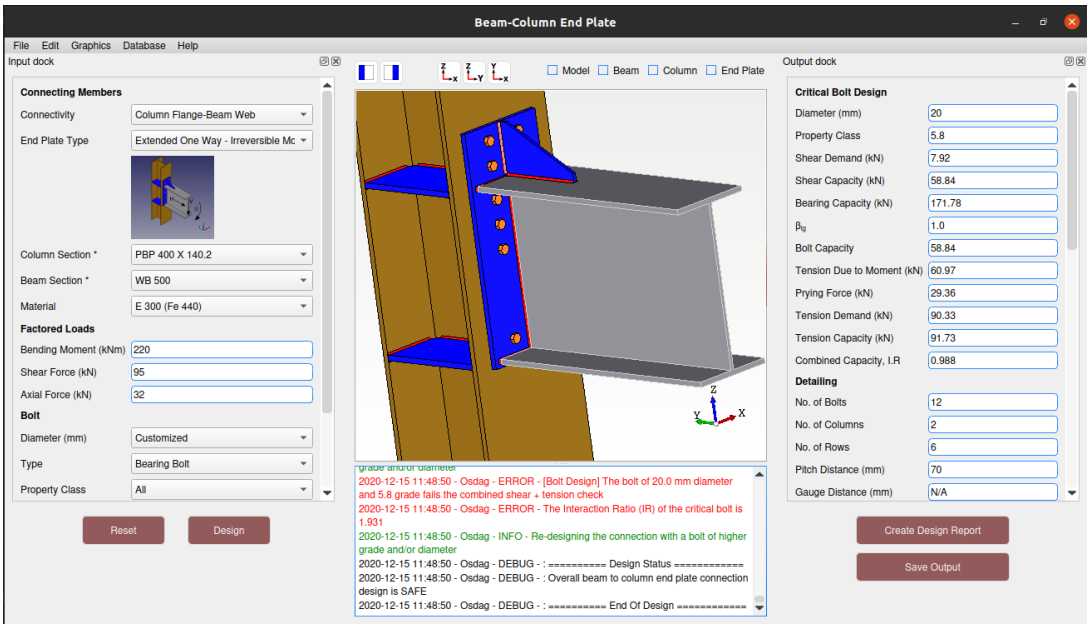

Figure 1.2: Osdag GUI

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

Screening Task

2.1 Problem Statement

State the problem defined during the screening task.

2.2 Tasks Done

List and describe the tasks performed as part of the screening task.

Chapter 3

Addition of web UI for various modules

3.1 Problem Statement

The UI for the "Bolted to the End", "Welded to the end", "Lap Joint Bolted", "Lap Joint Welded", "Butt Joint Welded", "Butt Joint Bolted" modules were not ready in the OSDAG Web app. The desktop already had the UI and functionality for most of the modules. This only meant creating a connection between the core functionality and the app based on the desktop app.

3.2 Tasks Done

The code containing the core functionality was already present for the most modules. Only a bridge between the code and the app was required for a working UI for the web app. Some modules also required creating new modals based on the desktop version of OSDAG.

3.2.1 Bolted to the End

Backend Code

This section presents a Python script for creating a bridge between the core codebase and the backend of the web app, which will serve the required data to the frontend.

Description of the script

The script is structured as follows:

- *Input Parameters*: The user specifies the different input parameters like section profile, material, bolt grade, diameter, etc which is then validated.
- *Design Calculations*: The program computes the number of bolts required, their arrangement, and verifies the adequacy of the end plate and bolt group.
- *Output*: Results include the number of bolts, their layout, and adequacy checks for the components as well as the 3D parts of the model.

Code snippet

```
import traceback
from osdag_api.modules.mesh_export import write_stl
old_stdout = sys.stdout  # Backup log
sys.stdout = open(os.devnull, "w")  # redirect stdout
sys.stdout = old_stdout  # Reset log

def get_required_keys() -> List[str]:
    # Using the same KEY constants as backend for consistency
    abhisogal_keys = [
        "Member.Profile",            # KEY_SEC_PROFILE
        "Member.Designation",        # KEY_SECSIZE
        "Material",                  # KEY_MATERIAL
        "Member.Material",           # KEY_SEC_MATERIAL
        "Connector.Plate.Thickness_List",  # KEY_PLATETHK
        "Bolt.Diameter",             # KEY_D
        "Bolt.Grade",               # KEY_GRD
        "Bolt.Type",                # KEY_TYP
        "Bolt.Bolt_Hole_Type",      # KEY_DP_BOLT_HOLE_TYPE
        "Bolt.Slip_Factor",         # KEY_DP_BOLT_SLIP_FACTOR
        "Connector.Material",       # KEY_CONNECTOR_MATERIAL
        "Design.Design_Method",     # KEY_DP_DESIGN_METHOD
```

Reference to full code: bolted_tension_member.py

Frontend Code

This section presents the configuration files for creating a UI between the backend of the web app and the frontend which will show the computated results and render the pre-built 3D components.

Reference to the code:

Input UI config: boltedToEndConfig.js

Output UI config: boltedToEndOutputConfig.js

3.2.2 Welded to the End

Backend Code

This section presents a Python script for creating a bridge between the core codebase and the backend of the web app, which will serve the required data to the frontend.

Description of the script

The script is structured as follows:

- *Input Parameters*: The user specifies the different input parameters like section profile, material, etc which is then validated.
- *Design Calculations*: The program computes the type of weld and verifies the adequacy of the end plate and bolt group.
- *Output*: Results include the computated results, section sizes, weld thickness, and adequacy checks for the components as well as the 3D parts of the model.

Code snippet

```
import traceback
from osdag_api.modules.mesh_export import write_stl
old_stdout = sys.stdout  # Backup logs
sys.stdout = open(os.devnull, "w")  # redirect stdout
sys.stdout = old_stdout  # Reset log

def get_required_keys() -> List[str]:
    # Using the same KEY constants as backend for consistency
    req_keys = [
```

```
        "Member.Profile",              # KEY_SEC_PROFILE
        "Member.Designation",          # KEY_SECSIZE
        "Material",                    # KEY_MATERIAL
        "Member.Material",             # KEY_SEC_MATERIAL
        "Connector.Plate.Thickness_List",  # KEY_PLATETHK
        "Connector.Material",          # KEY_CONNECTOR_MATERIAL
        "Design.Design_Method",        # KEY_DP_DESIGN_METHOD
        "Detailing.Corrosive_Influences",  #
           KEY_DP_DETAILING_CORROSIVE_INFLUENCES
        "Detailing.Edge_type",         # KEY_DP_DETAILING_EDGE_TYPE
        "Detailing.Gap",               # KEY_DP_DETAILING_GAP
        "Load.Axial",                  # KEY_AXIAL
        "Member.Length",               # KEY_LENGTH
```

Reference to full code: welded_tension_member.py

Frontend Code

This section presents the configuration files for creating a UI between the backend of the web app and the frontend which will show the computated results and render the pre-built 3D components.

Reference to the code:

Input UI config: weldedToEndConfig.js

Output UI config: weltdedToEndOutputConfig.js

3.2.3 Additional Changes done for Tension members

Modal with dynamic data source

Both the tension member modules required a specific type of modal whose data source changes with respect to the selected option for certain parameters.

Code snippet

```
      case 'dynamicSelect': {
        let options = field.getOptions(inputs);
        const value = options.find(opt => opt.value === inputs[
           field.key]);
```

```
        return (
          <Select
            options={options}
            value={value}
            onChange={(selected) => setInputs({ ...inputs, [field
               .key]: selected.value })}
            menuPortalTarget={document.body}
            styles={customSelectStyles}
            classNamePrefix="react-select"
            className="w-[60%]"
          />
        );
      }
```

New data fields for input data population

Some of the input parameters required new data sources from the backend. Newly added fields include :-

- *sectionProfileList* - Field containing the current list of section profiles.
- *channelList* - Field containing the list of current list of channel sizes.
- *sectionDesignation* - Field containing the list of current list of section sizes.

3.2.4 Lap Joint

Backend Code

This section presents a Python script for creating a bridge between the core codebase and the backend of the web app, which will serve the required data to the frontend.

Bolted This section outlines the code snippet for the Bolted Lap Joint module.

Code snippet

```
import traceback
from osdag_api.modules.mesh_export import write_stl
```

```
old_stdout = sys.stdout  # Backup log
sys.stdout = open(os.devnull, "w")  # redirect stdout
sys.stdout = old_stdout  # Reset log

def get_required_keys() -> List[str]:
    # Using the same KEY constants as backend for consistency
    req_keys = [
        KEY_SHEAR,
        KEY_AXIAL,
        KEY_MOMENT,
        KEY_MODULE,
        KEY_MATERIAL,
        KEY_PLATE_WIDTH,
        KEY_PLATE1_THICKNESS,
        KEY_PLATE2_THICKNESS,
        KEY_PLATE_WIDTH,
        KEY_GRD,
        KEY_D,
        KEY_TYP,
        KEY_DP_BOLT_HOLE_TYPE,
        KEY_DP_DETAILING_EDGE_TYPE,
        KEY_COVER_PLATE
    ]
```

Reference to full code: lap_joint_bolted.py

Welded This section outlines the code snippet for the Welded Lap Joint module.

Code snippet

```
import traceback
from osdag_api.modules.mesh_export import write_stl
old_stdout = sys.stdout  # Backup logs
sys.stdout = open(os.devnull, "w")  # redirect stdout
sys.stdout = old_stdout  # Reset log

def get_required_keys() -> List[str]:
```

```
    # Using the same KEY constants as backend for consistency
    req_keys = [
        KEY_SHEAR,
        KEY_AXIAL,
        KEY_MOMENT,
        KEY_MODULE,
        KEY_MATERIAL,
        KEY_PLATE_WIDTH,
        KEY_PLATE1_THICKNESS,
        KEY_PLATE2_THICKNESS,
        KEY_PLATE_WIDTH,
        KEY_DP_WELD_MATERIAL_G_O,
        KEY_DP_WELD_TYPE,
        KEY_WELD_SIZE,
        KEY_COVER_PLATE
    ]
    print("in lap_joint_welded.py: get_required_keys called,
       returning:", req_keys)
    return req_keys
```

Reference to full code: lap_joint_welded.py

Description of the script

The script is structured as follows:

- *Input Parameters*: The user specifies the different input parameters like material, weld thickness, weld grade, etc which is then validated.
- *Design Calculations*: The program computes the number of bolts required, their arrangement, and verifies the adequacy of the end plate and bolt group.
- *Output*: Results include the number of bolts, their layout, and adequacy checks for the components as well as the 3D parts of the model.

Frontend Code

This section presents the configuration files for creating a UI between the backend of the web app and the frontend which will show the computated results and render the

pre-built 3D components.

Reference to the code:

Bolted

Input UI config: lapJointBoltedConfig.js

Output UI config: lapJointBoltedOutputConfig.js

Welded

Input UI config: lapJointBoltedConfig.js

Output UI config: lapJointBoltedOutputConfig.js

3.2.5 Butt Joint

Backend Code

This section presents a Python script for creating a bridge between the core codebase and the backend of the web app, which will serve the required data to the frontend.

Bolted This section outlines the code snippet for the Bolted Lap Joint module.

Code snippet

```
import traceback
from osdag_api.modules.mesh_export import write_stl
old_stdout = sys.stdout  # Backup log
sys.stdout = open(os.devnull, "w")  # redirect stdout
sys.stdout = old_stdout  # Reset log

def get_required_keys() -> List[str]:
    # Using the same KEY constants as backend for consistency
    req_keys = [
        KEY_SHEAR,
        KEY_AXIAL,
        KEY_MOMENT,
        KEY_MODULE,
        KEY_MATERIAL,
        KEY_PLATE_WIDTH,
        KEY_PLATE1_THICKNESS,
        KEY_PLATE2_THICKNESS,
```

```
        KEY_PLATE_WIDTH,
        KEY_GRD,
        KEY_D,
        KEY_TYP,
        KEY_DP_BOLT_HOLE_TYPE,
        KEY_DP_DETAILING_EDGE_TYPE,
        KEY_COVER_PLATE
    ]
```

Reference to full code: butt_joint_bolted.py

Welded This section outlines the code snippet for the Welded Lap Joint module.

Code snippet

```
import traceback
from osdag_api.modules.mesh_export import write_stl
old_stdout = sys.stdout  # Backup logs
sys.stdout = open(os.devnull, "w")  # redirect stdout
sys.stdout = old_stdout  # Reset log

def get_required_keys() -> List[str]:
    # Using the same KEY constants as backend for consistency
    req_keys = [
        KEY_SHEAR,
        KEY_AXIAL,
        KEY_MOMENT,
        KEY_MODULE,
        KEY_MATERIAL,
        KEY_PLATE_WIDTH,
        KEY_PLATE1_THICKNESS,
        KEY_PLATE2_THICKNESS,
        KEY_PLATE_WIDTH,
        KEY_DP_WELD_MATERIAL_G_O,
        KEY_DP_WELD_TYPE,
        KEY_WELD_SIZE,
        KEY_COVER_PLATE
```

```
    ]
    print("in butt_joint_welded.py: get_required_keys called,
       returning:", req_keys)
    return req_keys
```

Reference to full code: butt_joint_welded.py

Description of the script

The script is structured as follows:

- *Input Parameters*: The user specifies the different input parameters like material, weld thickness, weld grade, etc which is then validated.
- *Design Calculations*: The program computes the number of bolts required, their arrangement, and verifies the adequacy of the end plate and bolt group.
- *Output*: Results include the number of bolts, their layout, and adequacy checks for the components as well as the 3D parts of the model.

Frontend Code

This section presents the configuration files for creating a UI between the backend of the web app and the frontend which will show the computated results and render the pre-built 3D components.

Reference to the code:

Bolted

Input UI config: buttJointBoltedConfig.js

Output UI config: buttJointBoltedOutputConfig.js

Welded

Input UI config: buttJointBoltedConfig.js

Output UI config: buttJointBoltedOutputConfig.js

3.2.6 Additional Changes done for Simple members

New data fields for input data population

Some of the input parameters required new data sources from the backend. Newly added fields include :-

- *coverPlateList* - Field containing the current list of cover plate types.
- *weldSizeList* - Field containing the current list of weld sizes.

Changes to fields to work properly with the modules

Some fields were not coded in the way required to make it work properly with the Lap joint and Butt Joint modules. Fixed fields are :-

- Plate thickness Fields are supposed to be able to select multiple options but it wasn't working that way. Hence, an exception had to be added.
- *toggleAllSelected* was not getting called when "All" option was being selected in input modals for thickness and similar inputs.
- Added a new "Design For" option in Design Preferences.

3.2.7 Plate Girder

This section outlines the changes made to the codebase for adding the "Optimized" option to Plate Girder which uses PSO for calculating the best option in the backend. The backend was already implemented and the mentioned code is for connecting the frontend to the backend.

Custom modal for specifying upper and lower bounds

```
<Modal
    width={window.innerWidth < 768 ? '90%' : 420}
    open={optimizedModal.state}
    onCancel={() => {
      setOptimizedModal({ state: false, key: '', });
      setModalValues({ lb: 0, ub: 0, inc: 0 });
    }
    }
    onOk={() => {
      setInputs({
        ...inputs,
        [`${optimizedModal.key}_lb`]: parseFloat(modalValues.lb),
```

```
        [`${optimizedModal.key}_ub`]: parseFloat(modalValues.ub),
        [`${optimizedModal.key}_inc`]: parseFloat(modalValues.inc
            ),
      })
      setOptimizedModal({ state: false, key: '', })
      setModalValues({ lb: 0, ub: 0, inc: 0 });
    }
    }
    centered
    styles={{
      body: {
        textAlign: "center",
        padding: "0px 0px !important",
        boxSizing: "content-box",
        display: "flex",
        flexDirection: "column",
        gap: "10px",
      },
    }}
  >
    <div style={{ background: "transparent", fontWeight: "bold"
       }}>Optimized Input</div>
    <div
      className="gap-4"
      style={{
        backgroundColor: "white",
        padding: "20px",
        borderRadius: "8px",
        width: "300px",
        boxShadow: "0 0 10px rgba(0,0,0,0.25)",
        display: 'grid',
        width: '100%',
      }}
    >
```

```
<div className="grid grid-cols-3 w-full">
  <div className="grid text-left content-center">Lower
     Bounds</div>
  <input
    type="number"
    value={String(modalValues.lb)}
    onChange={(e) => { setModalValues((prev) => ({ ...prev,
        lb: e.target.value })); }}
    className="grid col-start-2 col-end-4 h-9 border border
      -gray-400 rounded-md px-3 text-sm focus:border-osdag
      -green focus:ring-2 focus:ring-osdag-green/20
      outline-none"
  />
</div>
<div className="grid  justify-between grid-cols-3">
  <div className="grid text-left content-center">Upper
     Bounds</div>
  <input
    type="number"
    value={String(modalValues.ub)}
    onChange={(e) => { setModalValues((prev) => ({ ...prev,
        ub: e.target.value })); }}
    className="grid col-start-2 col-end-4 h-9 border border
      -gray-400 rounded-md px-3 text-sm focus:border-osdag
      -green focus:ring-2 focus:ring-osdag-green/20
      outline-none"

  />
</div>
<div className="grid  justify-between grid-cols-3">
  <div className="grid text-left content-center">Increment
     </div>
  <input
```

```
          type="number"
          value={String(modalValues.inc)}
          onChange={(e) => { setModalValues((prev) => ({ ...prev,
              inc: e.target.value })); }}
          className="grid col-start-2 col-end-4 h-9 border border
             -gray-400 rounded-md px-3 text-sm focus:border-osdag
             -green focus:ring-2 focus:ring-osdag-green/20
             outline-none"
        />
      </div>
    </div>
  </Modal>
```

New Optimization Graph component

This component was later modified and integrated into the codebase to provide the current Optimization Graph.

```
import { useRef, useEffect, useState } from 'react';
import Plot from 'react-plotly.js';
import Plotly from 'plotly.js-dist-min'

function OptimizationGraph({ data, onClose }) {
    const graphRef = useRef(null);
    const savePlotRef = useRef(null);
    const [dataState, setDataState] = useState(data);
    useEffect(() => {
        if (graphRef.current)
            savePlotRef.current.onclick = () => { Plotly.
               downloadImage(graphRef.current.el, { format: 'png'
               , width: 800, height: 600, filename: '
               pso_visualization' }) }
        console.log('Graph ref is set:', graphRef);
        let count = 0
        function updateGraph() {
```

```
            // Simulate data update
            setDataState((prev) => {
                let non_fease = {
                    x: [...prev.non_fease.x, Math.random() *
                       1000],
                    y: [...prev.non_fease.y, Math.random() *
                       1000],
                    z: [...prev.non_fease.z, Math.random() *
                       1000],
                };
                let fease = {
                    x: [...prev.fease.x, Math.random() * 800],
                    y: [...prev.fease.y, Math.random() * 800],
                    z: [...prev.fease.z, Math.random() * 800],
                };
                const newData = {
                    non_fease: non_fease,
                    fease: fease,
                    best: count++ % 5 === 0 ? {
                        x: [Math.random()],
                        y: [Math.random()],
                        z: [Math.random() * 600],
                    } : prev.best,
                };
                console.log('Updating graph data:', newData);
                return newData;
            });
        }
        // const intervalId = setTimeout(updateGraph, 1000);
    }, [graphRef]
    );

    return (<div className='flex flex-col w-full flex-1'>
        <div className='flex flex-col w-full h-[95%]'>
```

```
            <div className='flex flex-row h-fit p-4 font text-
               white' style={{ backgroundColor: "#6b7d20" }}>
                <div className='flex-1 content-center font-black'
                   >PSO OPTIMIZATION SPACE</div><div className='
                   flex w-auto gap-5'>
                    <div className='w-auto m-auto font-extrabold'
                       >ITERATION: 100</div>
                    <div className='w-auto m-auto font-black text
                       -[#ffd708] '>BEST: 777kg</div>
                    <div className='w-auto m-auto '>Particle: 25
                       @ Iter 100</div>
                    <div className='w-auto flex items-center
                       justify-center'>
                        <button className="flex h-fit box-content
                            justify-center px-4 py-2 items-center
                            bg-osdag-green text-white font-
                           semibold rounded-lg shadow-md hover:bg
                           -opacity-90 transition-opacity"
                           onClick={() => { onClose(); console.
                           log("should close") }}>CLOSE</button>
                    </div>
                </div>
            </div>
            <div className='flex flex-1 z-0 flex-row'>
                <div className="flex items-center w-[65%] justify
                   -center">
                    <Plot
                        ref={graphRef}
                        data={[
                            {
                                name: 'Utilization > 1',
                                x: dataState.non_fease.x,
                                y: dataState.non_fease.y,
                                z: dataState.non_fease.z,
```

```
                type: 'scatter3d',
                mode: 'markers',
                marker: {
                    size: 2,
                    color: "rgba(189, 0, 0, 0.83)
                       ",
                    opacity: 1
                },
            },
            {
                name: 'Utilization <= 1',
                x: dataState.fease.x,
                y: dataState.fease.y,
                z: dataState.fease.z,
                type: 'scatter3d',
                mode: 'markers',
                marker: {
                    size: 2,
                    color: 'rgba(34, 255, 0,1)',
                    opacity: 1
                },
            },
            {
                name: 'Gloabl Best',
                x: dataState.best.x,
                y: dataState.best.y,
                z: dataState.best.z,
                type: 'scatter3d',
                mode: 'markers',
                marker: {
                    size: 2,
                    color: 'rgba(255, 215, 0,1)',
                    opacity: 1
                },
```

```
        }
    ]}
    config={{ displayModeBar: false,
       responsive: true }}
    className="w-[80%] flex h-full pt-10 box-
       border"
    useResizeHandler={true}
    layout={{
        legend: { orientation: 'v',
           itemsizing: "constant", itemwidth:
            30, y: 0.9, x: 0.6, font: { size:
            10 }, bordercolor: "rgba(0,0,0,1)
           ", borderwidth: 1 },
        margin: { l: 10, r: 10, t: 30, b: 60
           },
        title: { text: "3D Scatter Plot:
           Utilization Ratio vs Depth vs
           Width", font: { weight: 800 } },
        autosize: true,
        scene: {
            zaxis: { showspikes: false, title
               : { text: "Weight (kg)" },
               tickangle: 0 },
            yaxis: {
                showspikes: false,
                title: {
                    text: "Depth (mm)"
                },
                tickangle: 0
            },
            xaxis: {
                showspikes: false,
                title: {
```

```
                                    text: "Utilization Ratio"
                                        ,
                                },
                                tickangle: 0
                            },
                            aspectmode: 'cube', dragmode:
                               false, hovermode: false,
                            camera: {
                                eye: { x: -2, y: -2, z: 1 },
                                up: { x: 0, y: 0, z: 1 }
                            }
                        },
                        annotations: [{ align: "left",
                           bgcolor: "rgba(255,255,227,1)",
                           bordercolor: "rgba(0,0,0,1)", y:
                           0.8, x: 0.3, text: `<b>Global Best
                           :</b><br>Iter: 10 | Particle: 13<
                           br>Depth: ${dataState.best.y}<br>
                           Weight: ${dataState.best.z}<br>UR:
                            ${dataState.best.x}` }]
                    }}
                />
            </div>
            <div className="w-[35%] flex flex-col justify-
               center text-5xl">
                <div className='h-fit font-black py-5 flex
                   justify-center text-xl'>Best Cross-Section
                    (I-Beam) </div>
                <div className='min-h-[50%] text-gray-500
                   items-center text-center flex justify-
                   center font-medium'>No Feasible Solution
                   Yet<br />(Searching...)</div>
            </div>
        </div>
```

```
        </div>
        <div className='flex justify-between box-border flex-row
           h-[5%] z-10 px-4 font w-full border-t-[1px] border-t-
           gray-700'>
            <div className='content-center items-center flex'>{"
               Optimizing"}</div>
            <div className='w-auto flex items-center justify-
               center'>
                <button ref={savePlotRef} className="flex h-fit
                   box-content justify-center px-4 py-1 items-
                   center  bg-gray-200 text-black font-semibold
                   rounded-lg shadow-md hover:bg-opacity-90
                   transition-opacity">Save Plot</button>
            </div>
        </div>
    </div>
    );
}

export default OptimizationGraph;
```

Chapter 4

Refactoring of codebase for various modules

4.1 Problem Statement

The old desktop app codebase was made without any plans for a web app using the same code. The core codebase, responsible for calculations, had to be replicated in the web app, which meant extra effort and work to sync code when the core codebase updated in the old desktop app. To avoid that issue, a unified git repository, containing the core codebase, is to be made which will be added as a git submodule to the repositories for desktop and web app. This requires the refactoring of the old codebase to a modularized new codebase differentiating core logic from GUI. Also, the refactoring is done to improve the app with a modern UI.

4.2 Tasks Done

4.2.1 Tension Member Bolted

Addition of functionality to the GUI for opening module

```
def open_bolted_end_tension(self):
    title = "Bolted to End Gusset"
    self.clear_layout(self.main_widget_layout)
```

```
    tension_bolted = CustomWindow(title, Tension_bolted, parent=
        self)

    # Load the last Design Inputs-start
        ------------------------------------
    last_design_folder = os.path.join('ResourceFiles', '
        last_designs')
    last_design_file = str(tension_bolted.backend.module_name()).
        replace(' ', '') + ".osi"
    last_design_file = os.path.join(last_design_folder,
        last_design_file)
    last_design_dictionary = {}

    # Create folder if it doesn't exist
    if not os.path.isdir(last_design_folder):
        os.makedirs(last_design_folder)

    # Load previous design if file exists
    if os.path.isfile(last_design_file):
        with open(str(last_design_file), 'r') as last_design:
            last_design_dictionary = yaml.safe_load(last_design)
            tension_bolted.setDictToUserInputs(
                last_design_dictionary)
    # Load the last Design Inputs-end
        ------------------------------------

    self.main_widget_instance = tension_bolted
    tension_bolted.openNewTab.connect(self.handle_add_tab)
    tension_bolted.downloadDatabase.connect(self.
        download_Database)
    self.main_widget_layout.addWidget(tension_bolted)
    index = self.tab_bar.currentIndex()
    self.tab_bar.setTabText(index, title)
    # Show docking Icons
```

```
    self.tab_widget_content[index][1] = True
    current_tab_data = self.tab_widget_content[index]
    self.update_docking_icons(current_tab_data[1],
       current_tab_data[2], current_tab_data[3], current_tab_data
       [4])
```

Addition of new modal for showing bolt spacing

This modal dynamically draws and shows a schematic for bolt placement according to the calculations

```
from PySide6.QtWidgets import (QApplication, QDialog, QWidget,
   QVBoxLayout,
                               QHBoxLayout, QLabel, QGraphicsView,
                                  QSizeGrip,
                               QGraphicsScene, QScrollArea)
from PySide6.QtGui import QColor
from PySide6.QtCore import Qt
from PySide6.QtGui import QPainter, QPen, QFont
from PySide6.QtGui import QPolygonF, QBrush
from PySide6.QtCore import QPointF
from osdag_gui.ui.components.dialogs.custom_titlebar import
   CustomTitleBar
from osdag_core.Common import *

class TensionBoltedDetails(QDialog):
    def __init__(self, connection_obj, rows=3, cols=2 , main =
       None):
        super().__init__()
        app = QApplication.instance()
        self.theme = app.theme_manager
        self.connection = connection_obj

        data = connection_obj.spacing(True)
        self.member_height = connection_obj.plate.height
```

```
        print(data)

        self.param_map = {}
        for elem in data[2:]:
            self.param_map[elem[1]] = elem[3]

        print(self.param_map)
        self.initUI()

    def setupWrapper(self):
        self.setWindowFlags(Qt.FramelessWindowHint | Qt.
            WindowSystemMenuHint)
        self.setObjectName("spacing_capacity_details")
        main_layout = QVBoxLayout(self)
        main_layout.setContentsMargins(1, 1, 1, 1)
        main_layout.setSpacing(0)
        self.title_bar = CustomTitleBar()
        self.title_bar.setTitle("Bolt Pattern")
        main_layout.addWidget(self.title_bar)
        self.content_widget = QWidget(self)
        main_layout.addWidget(self.content_widget, 1)
        size_grip = QSizeGrip(self)
        size_grip.setFixedSize(16, 16)
        overlay = QHBoxLayout()
        overlay.setContentsMargins(0, 0, 4, 4)
        overlay.addStretch(1)
        overlay.addWidget(size_grip, 0, Qt.AlignBottom | Qt.
            AlignRight)
        main_layout.addLayout(overlay)

    def initUI(self):
        self.setupWrapper()
```

```
        # Center the window on the screen with the same
            dimensions
        screen = QApplication.primaryScreen()
        screen_geometry = screen.availableGeometry()
        width, height = 900, 500
        x = screen_geometry.x() + (screen_geometry.width() -
            width) // 2
        y = screen_geometry.y() + (screen_geometry.height() -
            height) // 2
        self.setGeometry(x, y, width, height)

        # Main layout
        content_layout = QVBoxLayout(self.content_widget)
        content_layout.setContentsMargins(0, 0, 0, 0)
        content_layout.setSpacing(0)
        # Create scroll area for the entire content
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setHorizontalScrollBarPolicy(Qt.
            ScrollBarAsNeeded)
        scroll_area.setVerticalScrollBarPolicy(Qt.
            ScrollBarAsNeeded)

        scroll = QWidget()
        scroll.setObjectName("spacing_scroll_widget")

        main_layout = QHBoxLayout(scroll)

        # Left panel for parameter display
        left_panel = QWidget()
        left_layout = QVBoxLayout()
        params={}
        for key,value in self.param_map.items():
            print(key)
```

```
            if "Bolt" in key:
                key = key.split()[1].lower()
            else:
                key = key.split()[0].lower()
            params[key] = value
        # Parameter display labels
        # Display the parameter values
        for key, value in self.param_map.items():
            param_layout = QHBoxLayout()
            param_label = QLabel(f'{key}')
            value_label = QLabel(f'{value}')
            param_layout.addWidget(param_label)
            param_layout.addWidget(value_label)
            left_layout.addLayout(param_layout)

        left_layout.addStretch()
        left_panel.setLayout(left_layout)

        # Right panel for the drawing using QGraphicsView
        self.scene = QGraphicsScene()
        self.view = QGraphicsView(self.scene)
        if self.theme.is_light():
            self.view.setBackgroundBrush(QBrush(Qt.white))
        else:
            self.view.setBackgroundBrush(QBrush(QColor("#4A4A4A")
               ))
        self.view.setRenderHint(QPainter.Antialiasing)

        # Create and add the drawing to the scene
        self.createDrawing(params)

        # Add panels to main layout
        main_layout.addWidget(left_panel, 1)
        main_layout.addWidget(self.view, 3)
```

```
        # Ensure the view shows all content
        self.view.fitInView(self.scene.sceneRect(), Qt.
            KeepAspectRatio)

        scroll_area.setWidget(scroll)
        content_layout.addWidget(scroll_area)

    def createDrawing(self, params):

        # Extract parameters
        print(params)
        end = params['end']
        if 'gauge' in params:
            gauge = params['gauge']
        edge = params['edge']
        pitch = params['pitch']
        hole_diameter = self.connection.bolt_diameter_min
        self.rows = params["rows"]
        self.cols = params["columns"]
        print(f"rows: {self.rows}, cols: {self.cols}")
        # Calculate dimensions

        self.member_height = 2 * end + pitch *(self.rows -1)
        height = self.member_height

        width = (edge+gauge*(self.cols-1) + 100)
        self.length = width
        # Set up pens
        outline_pen = QPen(Qt.blue, 2)
        if self.theme.is_light():
            dimension_pen = QPen(Qt.black, 1.5)
        else:
            dimension_pen = QPen(QColor("#8A8A8A"), 1.5)
```

```
        weld_fill = QBrush(Qt.red)

        # Dimension offsets
        h_offset = 40
        v_offset = 60

        # Create scene rectangle with extra space for dimensions
        self.scene.setSceneRect(-h_offset, -v_offset,
                                width + 2*v_offset, height + 2*
                                   h_offset)

        mid_offset = 5

        # Draw rectangle
        # Top edge
        self.scene.addLine(0, 0, width, 0, dimension_pen)

        # Add broken line at right to denote continuation of
           member
        self.scene.addLine(width, 0, width, height/2 -mid_offset,
            dimension_pen)
        self.scene.addLine(width, height/2 -mid_offset, width-
           mid_offset, (height - mid_offset)/2, dimension_pen)
        self.scene.addLine(width-mid_offset, (height - mid_offset
           )/2, width+mid_offset, (height + mid_offset)/2,
           dimension_pen)
        self.scene.addLine(width+mid_offset, (height + mid_offset
           )/2, width, height/2 + mid_offset, dimension_pen)
        self.scene.addLine(width, height/2 + mid_offset, width,
           height, dimension_pen)

        # Bottom and left edges
        self.scene.addLine(width, height, 0, height,
           dimension_pen)
```

```
        self.scene.addLine(0, height, 0, 0, dimension_pen)


        # Draw holes
        for row in range(self.rows):
            for col in range(self.cols):
                # Start from edge distance (center of first hole)
                x_center = edge
                for i in range(col):
                    x_center += gauge

                # Center of hole is at (x_center, y_center)
                # Subtract hole_diameter/2 to draw ellipse
                   properly from top-left
                # Also, add pitch distance depending for row
                x = x_center - hole_diameter / 2
                y_center = end + pitch * (row)
                y = y_center - hole_diameter / 2

                print(f"row: {row}, col: {col}, x: {x}, y: {y}")
                self.scene.addEllipse(x, y, hole_diameter,
                   hole_diameter, outline_pen)

        print(params,dimension_pen)
        # Add dimensions
        self.addDimensions(params, dimension_pen)

    def addDimensions(self, params, pen):
        # Extract parameters
        end = params['end']
        if 'gauge' in params:
            gauge = params['gauge']
        pitch = params['pitch']
        edge = params['edge']
```

```
height=self.member_height
width=self.length

# Offsets for dimension lines
h_offset = 20
v_offset = 30

# Add horizontal dimensions
x_start = 0
x_segments = []

# Add vertical dimensions
y_start = 0
y_segments = []

# First edge
x_segments.append(('edge', x_start, x_start + edge))
x_start += edge
for i in range(self.cols - 1):
    x_segments.append(('gauge' ,x_start,x_start+gauge ))
    x_start+=gauge

# First end
y_segments.append(('end', y_start, y_start + end))
y_start += end
for i in range(self.rows - 1):
    y_segments.append(('pitch' ,y_start,y_start+pitch ))
    y_start+=pitch
# Add remaining height
y_segments.append(('remain', y_start, height))

# Draw each segment
```

```
        # Horizontal dimensions
        for label, x1, x2 in x_segments:
            value = x2 - x1
            self.addHorizontalDimension(x1, -h_offset, x2, -
                h_offset, f"{value:.1f}", pen)

        # Vertical dimensions
        for label, y1, y2 in y_segments:
            value = y2 - y1
            self.addVerticalDimension(-v_offset/2, y1, -v_offset
                /2, y2, f"{value:.1f}", pen)

        # Add right side dimension
        self.addVerticalDimension(width + v_offset, 0, width +
            v_offset, height, str(height), pen)

    def addHorizontalDimension(self, x1, y1, x2, y2, text, pen):
        self.scene.addLine(x1, y1, x2, y2, pen)
        arrow_size = 5
        ext_length = 10
        self.scene.addLine(x1, y1 - ext_length/2, x1, y1 +
            ext_length/2, pen)
        self.scene.addLine(x2, y2 - ext_length/2, x2, y2 +
            ext_length/2, pen)

        points_left = [
            (x1, y1),
            (x1 + arrow_size, y1 - arrow_size/2),
            (x1 + arrow_size, y1 + arrow_size/2)
        ]
        polygon_left = self.scene.addPolygon(QPolygonF([QPointF(x
            , y) for x, y in points_left]), pen)
        if self.theme.is_light():
            polygon_left.setBrush(QBrush(Qt.black))
```

```
        else:
            polygon_left.setBrush(QBrush(QColor("#4A4A4A")))

        points_right = [
            (x2, y2),
            (x2 - arrow_size, y2 - arrow_size/2),
            (x2 - arrow_size, y2 + arrow_size/2)
        ]
        polygon_right = self.scene.addPolygon(QPolygonF([QPointF(
           x, y) for x, y in points_right]), pen)
        if self.theme.is_light():
            polygon_right.setBrush(QBrush(Qt.black))
        else:
            polygon_right.setBrush(QBrush(QColor("#4A4A4A")))

        text_item = self.scene.addText(text)
        font = QFont()
        font.setPointSize(5)
        text_item.setFont(font)
        if self.theme.is_light():
            text_item.setDefaultTextColor(Qt.black)
        else:
            text_item.setDefaultTextColor(Qt.white)

        if y1 < 0:
            text_item.setPos((x1 + x2) / 2 - text_item.
               boundingRect().width() / 2, y1 - 25)
        else:
            text_item.setPos((x1 + x2) / 2 - text_item.
               boundingRect().width() / 2, y1 + 5)

    def addVerticalDimension(self, x1, y1, x2, y2, text, pen):
        self.scene.addLine(x1, y1, x2, y2, pen)
        arrow_size = 5
```

```
        ext_length = 10
        self.scene.addLine(x1 - ext_length/2, y1, x1 + ext_length
           /2, y1, pen)
        self.scene.addLine(x2 - ext_length/2, y2, x2 + ext_length
           /2, y2, pen)

        if y2 > y1:
            points_top = [
                (x1, y1),
                (x1 - arrow_size/2, y1 + arrow_size),
                (x1 + arrow_size/2, y1 + arrow_size)
            ]
            polygon_top = self.scene.addPolygon(QPolygonF([
               QPointF(x, y) for x, y in points_top]), pen)
            if self.theme.is_light():
                polygon_top.setBrush(QBrush(Qt.black))
            else:
                polygon_top.setBrush(QBrush(QColor("#4A4A4A")))

            points_bottom = [
                (x2, y2),
                (x2 - arrow_size/2, y2 - arrow_size),
                (x2 + arrow_size/2, y2 - arrow_size)
            ]
            polygon_bottom = self.scene.addPolygon(QPolygonF([
               QPointF(x, y) for x, y in points_bottom]), pen)
            if self.theme.is_light():
                polygon_bottom.setBrush(QBrush(Qt.black))
            else:
                polygon_bottom.setBrush(QBrush(QColor("#4A4A4A"))
                   )
        else:
            points_top = [
                (x2, y2),
```

```
                (x2 - arrow_size/2, y2 + arrow_size),
                (x2 + arrow_size/2, y2 + arrow_size)
            ]
            polygon_top = self.scene.addPolygon(QPolygonF([
               QPointF(x, y) for x, y in points_top]), pen)
            if self.theme.is_light():
                polygon_top.setBrush(QBrush(Qt.black))
            else:
                polygon_top.setBrush(QBrush(QColor("#4A4A4A")))

            points_bottom = [
                (x1, y1),
                (x1 - arrow_size/2, y1 - arrow_size),
                (x1 + arrow_size/2, y1 - arrow_size)
            ]
            polygon_bottom = self.scene.addPolygon(QPolygonF([
               QPointF(x, y) for x, y in points_bottom]), pen)
            if self.theme.is_light():
                polygon_bottom.setBrush(QBrush(Qt.black))
            else:
                polygon_bottom.setBrush(QBrush(QColor("#4A4A4A"))
                   )

        text_item = self.scene.addText(text)
        font = QFont()
        font.setPointSize(5)
        text_item.setFont(font)
        if self.theme.is_light():
            text_item.setDefaultTextColor(Qt.black)
        else:
            text_item.setDefaultTextColor(Qt.white)

        if x1 < 0:
            text_item.setPos(x1 - 10 - text_item.boundingRect().
```

```
                width(), (y1 + y2) / 2 - text_item.boundingRect().
                height() / 2)
        else:
            text_item.setPos(x1 + 15, (y1 + y2) / 2 - text_item.
                boundingRect().height() / 2)
```

4.2.2 Other changes and fixes

- Fix an app crash because of not setting the *view_cube* to None when resetting the CAD widget after locking the dock, which causes a crash when unlocking the input dock and starting design.
- Fix empty output dock due to start of calculation, even when the inputs are not set properly, when loading inputs. This was happening due to *output_title_change* function calculating the values, before the inputs are set.
- Fix not showing proper information on hover for "Tension Member Bolted" module.
- Changed the beam color to match the current standard for the app.

Chapter 5

Conclusions

5.1 Tasks Accomplished

5.1.1 Implementation for various modules in web app

Various modules were not implemented in the web app like "Tension Member Bolted", "Tension Member Welded", "Lap Joint Bolted", "Lap Joint Welded", "Butt Joint Bolted", "Butt Joint Welded". This added to the base capability of the web app, making it more useful and enriched, slowly catching up to the desktop app.

5.1.2 Refactoring and fixes of the desktop app

Refactoring of the "Tension Member Bolted" module was done according to the structure of the codebase, making sure to modularize the core logic of the module as well as conform to the new GUI of the desktop app. Few fixes were also performed to fix an application crash as well as empty output dock on loading input from file. This made sure that the app doesn't behaved in an undefined way as well as add more features to it.

5.2 Skills Developed

5.2.1 Technical skills

- **Python**: Proficiency in backend refactoring, debugging crashes and undefined behaviour.

- **JavaScript and React**: Better understanding of how to create components, and integration of the components with the backend.
- **WebSocket**: Knowledge was gained regarding persistent connections between frontend and backend for realtime updates using graphs.
- **Configuration-Driven Development**: Understanding of designing flexible, reusable systems through configuration files to minimize code duplication and enhance scalability.

5.2.2 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

- OSDAG Repository
- OSDAG Web 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.