



# FOSSEE Semester Internship Report

On

## Osdag on Web

**Submitted by**

**Faran Imam**

*3rd Year Computer Engineering B.Tech Student,*

*Zakir Husain College of Engineering and Technology*

*Aligarh Muslim University*

Aligarh

**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 16, 2025

# Acknowledgments

I would like to express my sincere gratitude to FOSSEE for providing me with the invaluable opportunity to undertake this semester-long fellowship. This experience has been instrumental in my professional growth and understanding of open-source software development.

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.

Special thanks are due to Parth Karia, my mentor and project staff member at the Osdag team, whose expert guidance and continuous support were invaluable throughout my internship. His technical knowledge and encouraging approach enabled navigation through complex challenges in module development and 3D model rendering for Osdag-web.

Recognition must also go to my fellow interns, Raghav and Aum Ghelani, who were excellent collaborators throughout this journey. Working as a team, we were able to tackle challenging problems and learn from each other's perspectives.

Finally, appreciation extends to the entire Osdag team for creating a collaborative environment that fostered learning and innovation. This fellowship has equipped me with valuable technical skills and a deeper appreciation for the open-source philosophy in education.

# 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 [?, ?, ?]. 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.

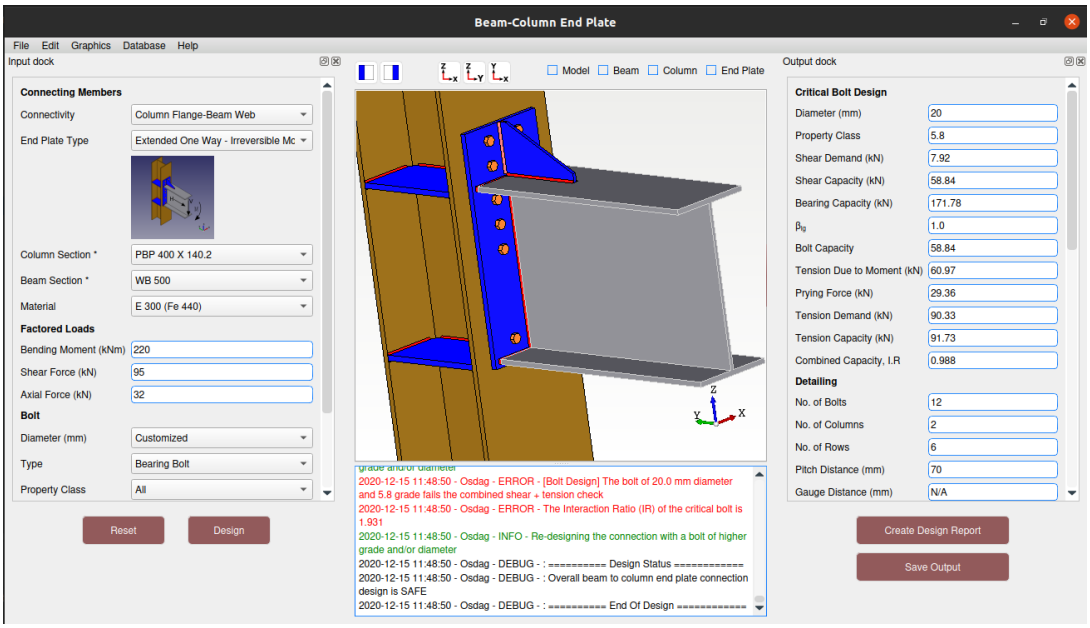


Figure 1.2: Osdag GUI

### 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

The screening task for Web App Development of Osdag was designed to evaluate applicants' ability to contribute to the development of Osdag's web-based platform. The task comprised three main components:

### 2.1.1 Setup and Deployment

Set up and run Osdag-on-cloud, ensuring proper functionality on local or preferred cloud environment.

### 2.1.2 Module Development

Create a UI for a cloud module of Osdag that resembles its desktop version. Implement an endpoint in Django/REST for any one of the following modules, mirroring the existing fin plate or end plate implementations:

- Beam-to-beam splices (end plate)
- Beam-to-beam splices (cover plate bolted)
- Beam-to-column connections (end plate)
- Column-to-column splices (cover plate bolted)
- Column-to-column splices (cover plate welded)

- Column-to-column connections (end plate)

### 2.1.3 Frontend and Backend Stack

Develop the UI for the selected module using React and implement endpoints using Django and Django REST Framework.

## 2.2 Tasks Done

### 2.2.1 Alternative Development Approach

Instead of setting up the complete Osdag-on-cloud environment locally, I created a new Vercel project to focus on demonstrating core development capabilities through rapid prototyping and streamlined development.

### 2.2.2 Frontend Development

Selected the beam-to-beam cover plate bolted module for implementation. Successfully recreated the exact UI of the desktop application using React, including:

- Pixel-perfect replication of form controls and input validation
- Responsive design while maintaining consistency with the original interface
- Complete user interaction patterns that mirror the desktop version

### 2.2.3 Backend API Implementation

Developed backend functionality using Django and Django REST Framework:

- Created RESTful API endpoints for data communication
- Implemented data validation and processing logic
- Established seamless data flow from React frontend to Django backend

### 2.2.4 Challenges and Outcome

While frontend and backend integration was successful, challenges were encountered in implementing 3D model rendering capabilities. Despite this limitation, the completed work in UI replication and backend API development demonstrated sufficient technical competency to secure selection for the fellowship program.

# Chapter 3

# Internship Task 1 Title

## 3.1 Task 1: Installation Manual Content

The comprehensive installation manual developed covers the complete setup process for Osdag-web development environment. The manual provides structured guidance to reduce installation time from weeks to hours.

### 3.1.1 Prerequisites and Initial Setup

The installation process begins with essential prerequisites:

- Osdag Desktop application installation (mandatory requirement)
- Anaconda installation for Python environment management
- PostgreSQL database setup with proper configuration
- Visual Studio Build Tools for Windows compatibility

### 3.1.2 Environment Configuration

The manual details the creation of isolated development environments:

Listing 3.1: Conda Environment Setup

```
conda create -n osdag python=3.8
conda activate osdag
```

### 3.1.3 Database Setup and Configuration

PostgreSQL configuration includes role creation and database initialization:

Listing 3.2: PostgreSQL Database Setup

```
CREATE ROLE osdagdeveloper PASSWORD 'password' SUPERUSER CREATEDB
CREATEROLE INHERIT REPLICATION LOGIN;
CREATE DATABASE "postgres_Intg_osdag" WITH OWNER osdagdeveloper;
```

### 3.1.4 Project Dependencies and Migration

The manual includes dependency installation and database migration steps:

Listing 3.3: Project Setup Commands

```
pip install -r requirements.txt
python populate_database.py
python update_sequences.py
python manage.py migrate
```

### 3.1.5 Troubleshooting Solutions

The manual provides solutions for common installation errors:

- Visual C++ Build Tools requirement for pycosat compilation
- PythonOCC library installation using conda-forge
- Typing extensions compatibility issues for Python 3.8+
- Node.js dependencies for React frontend

### 3.1.6 Alternative Installation Method

For complex installation failures, the manual includes a pre-configured conda environment package that can be downloaded and integrated directly into the user's Anaconda installation, providing a fallback solution when standard installation encounters persistent issues.

The manual successfully reduces the setup complexity and provides clear guidance for both standard installation and troubleshooting scenarios.

### 3.1.7 Current Installation Evolution

Following this internship, fellow intern Aum Ghelani has developed an automated installation method using shell scripting, CLI tools, and Docker containerization. This new approach eliminates the need for manual configuration and reduces installation time from hours to minutes.

The Docker-based solution provides:

- Consistent environments across different operating systems
- Automated dependency resolution
- Elimination of platform-specific configuration issues

While the manual installation guide remains valuable for understanding system architecture and troubleshooting, the automated Docker approach is now the recommended installation method for new developers joining the Osdag-web project.

# Chapter 4

# CAD Model API and 3D Rendering Fixes

## 4.1 Task 2: Problem Statement

The CAD Model Generation API in the Osdag web application was completely non-functional, failing to generate any CAD models for structural connections. Initially, the API generated only the complete model assembly but failed to produce individual component sections like Beam, Column, Plate, and Connector sections. The React Three Fiber 3D rendering component had geometry parsing issues where all sections appeared visually identical due to material and texture rendering problems.

## 4.2 Task 2: Tasks Done

Fixed the Django-based CAD Model API by implementing proper session management for multiple connection types including FinPlate, CleatAngle, EndPlate, SeatedAngle, CoverPlateBolted, CoverPlateWelded, BeamBeamEndPlate, and BeamToColumnEndPlate connections. Enhanced the iterative section generation process to create individual component models based on connection type. Implemented robust BREP to OBJ conversion using FreeCAD subprocess calls with proper error handling. Restructured the React Three Fiber component to properly parse multiple OBJ files, extract geometries for each component, and apply distinct materials with different colors and textures. Added comprehensive logging and memory-efficient file handling using BytesIO for JSON responses.

## 4.3 Task 2: Python Code

This section presents the key fixes implemented in the CAD Model Generation API and 3D rendering component. The Python script handles multiple connection session types, generates component-specific CAD models, and converts BREP files to OBJ format. The JavaScript component provides interactive 3D visualization with proper material differentiation between structural components.

### 4.3.1 Description of the Script

The solution consists of two main components:

- Session Management: The API detects active connection sessions from multiple cookie types and identifies the specific connection module being used.
- Dynamic Section Generation: Based on connection type, the system generates appropriate component sections using a mapping system.
- File Conversion Pipeline: BREP files are converted to OBJ format using FreeCAD commands and stored in memory using BytesIO for efficient JSON responses.
- 3D Rendering: React Three Fiber component parses OBJ data, extracts geometries, and renders with distinct materials for visual differentiation.

### 4.3.2 Python Code

The key parts of the fixed CAD Model Generation API are shown below:

Listing 4.1: CAD Model API Session Management and Section Generation

```
%--------------------begin code-------------
Get design session id for all supported connection types
session_cookies = {
"FinPlate": request.COOKIES.get("fin_plate_connection_session"),
"CleatAngle": request.COOKIES.get("cleat_angle_connection_session"),
"EndPlate": request.COOKIES.get("end_plate_connection_session"),
"SeatedAngle": request.COOKIES.get("seated_angle_connection"),
"CoverPlateBolted": request.COOKIES.get("
   cover_plate_bolted_connection_session"),
```

```
"BeamToColumnEndPlate": request.COOKIES.get("
    beam_to_column_end_plate_connection_session")
}
Determine sections based on the session type
if session_type == "FinPlate":
sections = ["Model", "Beam", "Column", "Plate"]
elif session_type == "CleatAngle":
sections = ["Model", "Beam", "Column", "cleatAngle"]
elif session_type == "EndPlate":
sections = ["Model", "Beam", "Column", "Plate"]
elif session_type == "BeamToColumnEndPlate":
sections = ["Model", "Beam", "Column", "Connector"]
Generate models for each section
for section in sections:
try:
path = module_api.create_cad_model(input_values, section, cookie_id)
if not path:
continue
    # Convert BREP to OBJ using FreeCAD
    output_obj_path = path_to_file.replace(".brep", ".obj")
    command_with_arg = f'{command} {macro_path} {path_to_file} {
        output_obj_path}'
    process = subprocess.Popen(command_with_arg.split(), stdout=
        subprocess.PIPE, stderr=subprocess.PIPE)

    # Read the generated OBJ file into BytesIO
    output_obj = BytesIO()
    with open(output_obj_path, "rb") as obj_file:
        output_obj.write(obj_file.read())

    output_files[section] = output_obj.getvalue().decode("utf-8")

except Exception as e:
    print(f"Exception while generating {section}: {e}")
%-------------------- end code ---------------
```

### 4.3.3 Explanation of the Python Code

- Line 2-8: Implements comprehensive session cookie detection for all supported connection types using a dictionary mapping approach to identify active sessions.
- Line 11-19: Dynamic section mapping system that determines required components based on connection type, enabling module-specific model generation for different structural connections.
- Line 22-24: Core iterative generation loop that creates CAD models for each section using the Osdag module API with proper error handling for failed generations.
- Line 27-29: BREP to OBJ file conversion using FreeCAD subprocess calls with proper command construction and process management.
- Line 31-35: Memory-efficient file handling using BytesIO to read generated OBJ files and store them as strings for JSON serialization and response.
- Line 37-38: Exception handling with detailed logging to track generation failures and continue processing remaining sections.

### 4.3.4 React UI Code

The user interface component that handles section selection and renders the 3D canvas:

Listing 4.2: 3D Model UI Component with Section Selection

```
1  %--------------------begin code-------------
2  {/* Middle - 3D Model */}
3  <div className="superMainBody_mid">
4    <div className="options-container">
5      {options.map((option) => (
6        <div
7          key={option}
8          className="option-wrapper"
9          onClick={() => setSelectedView(option)}
10       >
11         <div
12           className={`option-box ${
13             selectedView === option ? "selected" : ""
```

```
          }`}
        ></div>
        <span className="option-label">{option}</span>
      </div>
    ))}
  </div>
{loading ? (
<div className="modelLoading">
<p>Loading Model...</p>
</div>
) : renderBoolean ? (
<div className="cadModel">
<Canvas
gl={{ antialias: true, preserveDrawingBuffer: true }}
onCreated={({ gl }) => {
gl.setClearColor("#ADD8E6");
}}
>
<PerspectiveCamera
       ref={cameraRef}
       makeDefault
       position={cameraPos}
       fov={fov}
       near={0.1}
       far={1000}
     />
<Suspense
fallback={
<Html>
<p>Loading 3D Model...</p>
</Html>
}
>
<Model
         modelPaths={cadModelPaths}
         selectedView={selectedView}
         key={modelKey}
       />
</Suspense>
```

```
53 </Canvas>
54 </div>
55 ) : (
56 <div className="modelback"></div>
57 )}
58 </div>
59 %-------------------- end code ---------------
```

### 4.3.5 Explanation of the React UI Code

- Line 3-15: Dynamic option selection interface that maps through available sections and provides clickable buttons for switching between Model, Beam, Column, and Plate views.
- Line 17-21: Loading state management with conditional rendering to display loading messages while models are being processed.
- Line 23-32: Three.js Canvas setup with antialiasing, light blue background color, and proper drawing buffer preservation for screenshot functionality.
- Line 33-40: Camera configuration with perspective projection, adjustable position and field of view for optimal model viewing.
- Line 41-49: Suspense wrapper for lazy loading of 3D components with fallback loading indicators and model path management.
- Line 44-47: Model component integration that receives parsed CAD data and selected view state for dynamic section rendering.

### 4.3.6 JavaScript Code

The key parts of the fixed React Three Fiber rendering component:

Listing 4.3: 3D Model Geometry Extraction and Material Application

```
1 %--------------------begin code-------------
2 // Parse OBJ data when modelPaths change
3 useEffect(() => {
4 if (modelPaths) {
```

```
try {
const loader = new OBJLoader();
const parsedData = Object.fromEntries(
Object.entries(modelPaths).map(([key, objData]) => {
return [key, loader.parse(objData)];
})
);
setParsedModels(parsedData);
} catch (error) {
console.error("Error parsing .obj data:", error);
}
}
}, [modelPaths]);
// Extract geometry from parsed objects
const getGeometry = (obj) => {
let g;
obj.traverse((c) => {
if (c.type === "Mesh") {
c.material.map = texture;
c.material.needsUpdate = true;
g = c.geometry;
}
});
return g;
};
// Beam Section with distinct gray material
{selectedView === "Beam" && geometryBeam && (
<>
<mesh geometry={geometryBeam} scale={0.008} position={[0, 0, 4]}
   rotation={[Math.PI / -2, 0, 0]}>
<meshPhysicalMaterial
     color="#808080"
     metalness={0.25}
     roughness={0.3}
     opacity={1.0}
     transparent={true}
     transmission={0.008}
   />
</mesh>
```

```
43 <primitive
44 object={
45 new THREE.LineSegments(
46 new THREE.EdgesGeometry(geometryBeam, 15),
47 new THREE.LineBasicMaterial({ color: "black" })
48 )
49 }
50 scale={0.008}
51 rotation={[Math.PI / -2, 0, 0]}
52 position={[0, 0, 4]}
53 />
54 </>
55 )}
56 %-------------------- end code ---------------
```

### 4.3.7 Explanation of the Code

- Line 3-19: Dynamic section mapping system that determines required components based on connection type, enabling module-specific model generation.
- Line 20-29: Enhanced OBJ parsing logic using OBJLoader that processes multiple model sections received from the backend API.
- Line 30-42: Geometry extraction function that traverses Three.js object hierarchy to locate mesh geometries and applies textures properly.
- Line 43-56: Beam component rendering with distinct gray metallic material and edge geometry for better visual definition and component differentiation.

### 4.3.8 Visual Results

The implementation successfully generates distinct visual representations for different structural components as shown in the following figures:

### 4.3.9 Task Outcome and Results

The implementation successfully resolved all identified issues in the CAD Model Generation system:

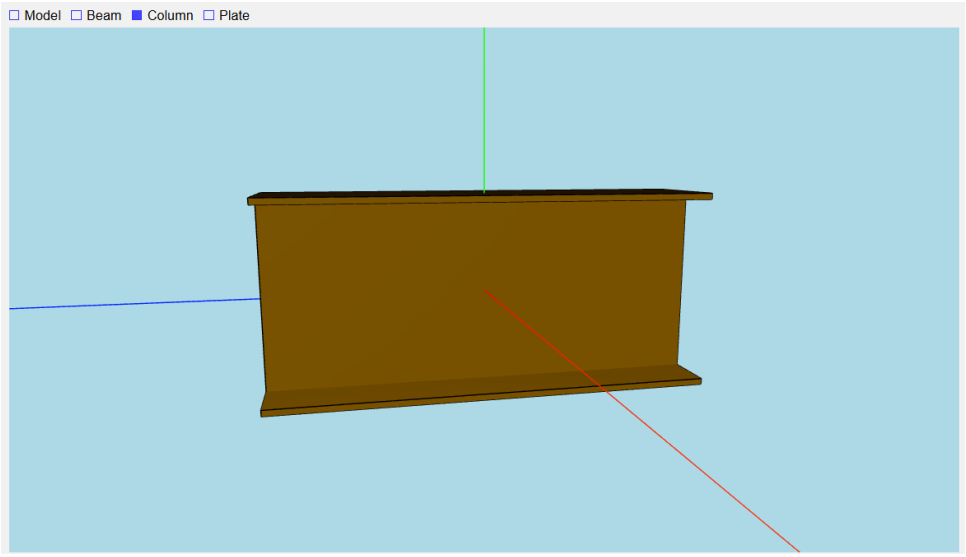


Figure 4.1: Column Section - Bronze colored I-beam column component with distinct material properties

- Functional API: The CAD Model API now properly generates models for all supported connection types without failures.
- Component-Specific Generation: Individual sections (Model, Beam, Column, Plate, Connector) are now generated based on connection type, enabling detailed component analysis.
- Improved 3D Visualization: The React Three Fiber component properly renders different components with distinct materials, making visual differentiation possible between structural elements.
- Robust Error Handling: Comprehensive error handling ensures the system continues operation even when individual sections fail to generate.
- Enhanced User Experience: Users can now visualize complete assemblies and individual components with proper material representation in the web interface.

The fixed system now supports eight different connection types and generates appropriate component sections for each, significantly improving the CAD visualization capabilities of the Osdag web application.

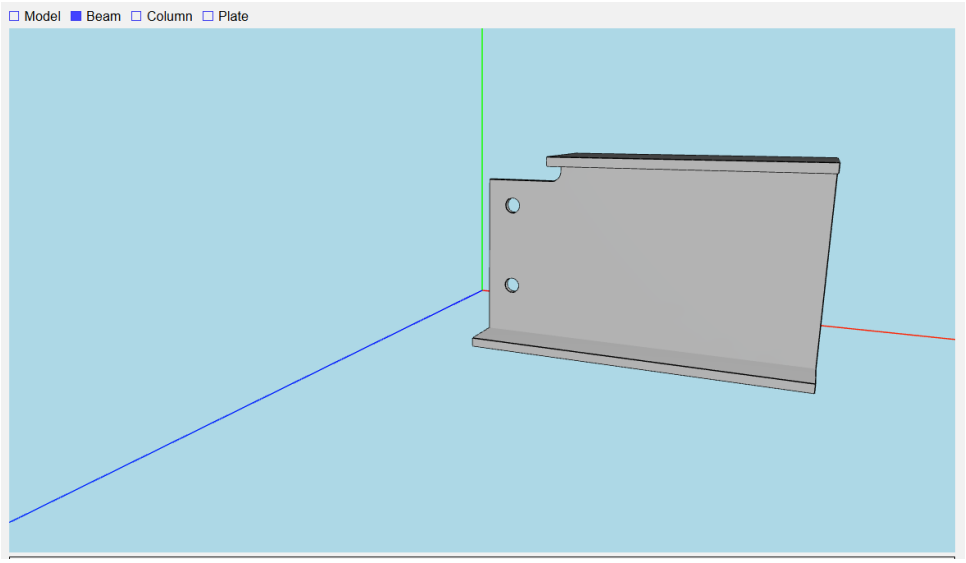


Figure 4.2: Beam Section - Gray metallic beam component showing proper geometry extraction and material application

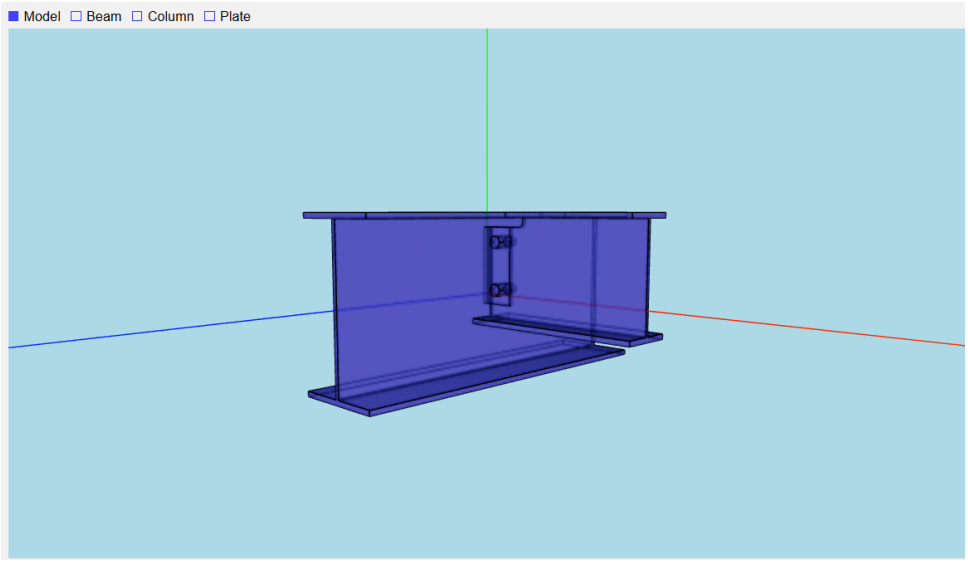


Figure 4.3: Complete Model Assembly - Transparent view showing the entire structural connection with end plate and bolts

# Chapter 5

# Output Dock Implementation

## 5.1 Task 3: Problem Statement

The internship task involved fixing existing modules in the Osdag steel connection design software, specifically addressing issues with four critical modules: FinPlate, CleatAngle, EndPlate, and SeatedAngle. The primary challenge was resolving the OutputDock functionality where output values were not displaying properly due to incorrect mapping and data structure inconsistencies. The core problems identified were conflicting key names between modules despite representing different parameters, inconsistent mapping between backend calculations and frontend display components, and data formatting mismatches between API responses and React component expectations.

## 5.2 Task 3: Tasks Done

The task was approached systematically by analyzing the data flow from backend calculations to frontend display components. A suffix-based naming convention was implemented to differentiate between parameters that had similar names but represented different physical quantities or belonged to different structural components. The React components were modified to handle the new data structure format, ensuring consistent mapping between calculation results and display elements. Key methodologies implemented included suffix-based key differentiation for duplicate parameter names, structured data formatting with key-label-value triplets, and modal-based detailed view implementation for spacing and capacity parameters.

## 5.3 Task 3: Python Code

This section presents the Python implementation for the output dock functionality, specifically focusing on the CleatAngle module. The code demonstrates how output values are generated, formatted, and transmitted to the frontend components for proper display.

### 5.3.1 Description of the Script

The output generation system is structured as follows:

- Module Creation: The system creates connection modules from input parameters using the create$_{f}rom_{i}nputfunction.DataExtraction : Multipleoutputsourcesareconsolidatedinc$

- Key Disambiguation: Suffix-based naming is applied to prevent conflicts between similar parameter names from different structural components.

- Format Standardization: All output parameters are converted to a consistent key-label-value format for frontend consumption.

- Error Handling: The system includes validation and logging mechanisms to track calculation processes.

### 5.3.2 Python Code

The Python script for output generation is shown below, demonstrating the backend processing for the CleatAngle connection module.

Listing 5.1: Output Generation with Suffix-based Key Mapping for CleatAngle Module

```
1 %--------------------begin code-------------
2 def generate_output(input_values: Dict[str, Any]) -> Dict[str, Any]:
3 """
4 Generate, format and return the input values from the given output
      values.
5 Output format (json):
6 {
7 "Bolt.Pitch": {
8 "key": "Bolt.Pitch",
9 "label": "Pitch Distance (mm)",
```

```
"value": 40
}
}
"""
output = {}  # Dictionary for formatted values
module = create_from_input(input_values)  # Create module from input
print('module : ', module)
print('type of module : ', type(module))

# Generate output values in unformatted form
raw_output_text = module.output_values(True)
raw_output_spacing = module.spacing(True)

# Generate bolt capacity details for both supported and supporting legs
raw_bolt_capacity_supported = module.bolt_capacity_details_supported(
    True)
raw_bolt_capacity_suporting = module.bolt_capacity_details_suporting(
    True)

# Add suffixes to duplicate-prone supported keys
raw_supported = [
    (f"{key}_supported", label, typ, value, visible)
    for key, label, typ, value, visible in raw_bolt_capacity_supported
    if key  # only if key is not None
]

raw_supporting = [
    (f"{key}_supporting", label, typ, value, visible)
    for key, label, typ, value, visible in raw_bolt_capacity_suporting
    if key
]

logs = module.logs
raw_output = raw_output_spacing + raw_output_text + raw_supported +
    raw_supporting

# Loop over all the text values and add them to output dict
for param in raw_output:
    if param[2] == "TextBox":  # If the parameter is a text output
```

```
46          key = param[0]     # id/key
47          label = param[1]   # label text
48          value = param[3]   # Value as string
49
50          output[key] = {
51              "key": key,
52              "label": label,
53              "value": value
54          }  # Set label, key and value in output
55
56  return output, logs
57  %-------------------- end code ---------------
```

### 5.3.3 Explanation of the Python Code

- Line 1-13: Function definition and documentation explaining the expected output format with key-label-value structure for frontend consumption.
- Line 14-18: Module initialization from input parameters and debugging output to track module creation process.
- Line 20-23: Raw data extraction from multiple sources including basic output values, spacing parameters, and bolt capacity details for both structural legs.
- Line 25-37: Implementation of suffix-based naming convention to differentiate between supported and supporting leg parameters, preventing key conflicts in the output dictionary.
- Line 39-41: Consolidation of all raw output data into a single list for unified processing.
- Line 43-53: Iteration through all parameters, filtering for TextBox type outputs, and formatting them into the standardized key-label-value structure required by the frontend components.

### 5.3.4 Frontend Integration Code

The React component integration handles the formatted output data:

Listing 5.2: React useEffect Hook for Output Data Processing

```
%--------------------begin code-------------
useEffect(() => {
if (displayOutput) {
try {
console.log("Actual Output", designData);
const formatedOutput = {};
        for (const [key, value] of Object.entries(designData)) {
            const newKey = key;
            const label = value.label;
            const val = value.value;

            if (val !== undefined && val !== null) {
                formatedOutput[newKey] = { label, val };
            }
        }

        setOutput(formatedOutput);
        console.log("formated Output", formatedOutput);
    } catch (error) {
        console.log(error);
        setOutput(null);
    }
}
}, [designData]);
// Output component rendering
{<CleatAngleOutputDock output={output} />}
%-------------------- end code ---------------
```

### 5.3.5 Explanation of the Frontend Code

- Line 1-2: useEffect hook triggered when displayOutput state changes to process incoming design data from the backend API.
- Line 4-6: Error handling wrapper with debugging output to track the actual output data structure received from the backend.
- Line 7-15: Data transformation loop that extracts key, label, and value from each

output parameter while filtering out undefined or null values.

- Line 17-22: State management to update the output data and comprehensive error handling with fallback to null state.
- Line 25: Component integration where the processed output data is passed to the CleatAngleOutputDock component for rendering.

### 5.3.6 Output Mapping Implementation

The CleatAngleOutputDock component uses a custom mapping system to organize output parameters:

Listing 5.3: Custom Output Parameter Mapping for CleatAngle Module

```
%--------------------begin code-------------
const customMapping = {
"Cleat Angle": [
{ key: "Cleat.Angle", label: "Cleat Angle Designation" },
{ key: "Plate.Height", label: "Height (mm)" },
{ key: "Cleat.Shear", label: "Shear Yielding Capacity (kN)" },
{ key: "Cleat.BlockShear", label: "Block Shear Capacity (kN)" },
{ key: "Cleat.MomDemand", label: "Moment Demand (kNm)" },
{ key: "Cleat.MomCapacity", label: "Moment Capacity (kNm)" },
],
"Bolts on Supported Leg": [
{ key: "Bolt.Line", label: "Bolt Columns (nos)" },
{ key: "Bolt.OneLine", label: "Bolt Rows (nos)" },
{ key: "Bolt.Force (kN)", label: "Bolt Force (kN)" },
{ key: "Bolt.Capacity_sptd", label: "Bolt Value (kN)" },
{ key: "CapacityModal_supported", label: "Capacity Details" },
{ key: "SpacingModal", label: "Spacing" },
],
"Bolts on Supporting Leg": [
{ key: "Cleat.Spting_leg.Line", label: "Bolt Columns (nos)" },
{ key: "Cleat.Spting_leg.OneLine", label: "Bolt Rows (nos)" },
{ key: "Cleat.Spting_leg.Force", label: "Bolt Force (kN)" },
{ key: "Bolt.Capacity_spting", label: "Bolt Value (kN)" },
{ key: "CapacityModal_supporting", label: "Capacity Details" },
{ key: "SpacingModal", label: "Spacing" },
```

```
26 ],
27 };
28 %-------------------- end code ---------------
```

### 5.3.7 Explanation of the Output Mapping

- Line 1-9: Cleat Angle section mapping that displays fundamental cleat angle properties including designation, height, and capacity parameters.
- Line 10-17: Supported leg bolt configuration mapping with suffixed keys to differentiate from supporting leg parameters.
- Line 18-25: Supporting leg bolt configuration with distinct key names to prevent conflicts with supported leg parameters.
- Line 15-16, 22-23: Modal trigger keys for detailed capacity and spacing information that open popup dialogs with additional technical details.

### 5.3.8 Task Outcome and Results

The implementation successfully resolved all identified issues in the Output Dock system. All four modules (FinPlate, CleatAngle, EndPlate, SeatedAngle) now properly display calculated output values without mapping conflicts. The suffix-based naming convention eliminated key conflicts between similar parameters from different structural components. Standardized key-label-value format ensures reliable data transmission between backend calculations and frontend display components, significantly improving the usability of the Osdag web application.

# Chapter 6

# Module Enhancement and File Management System

## 6.1 Task 3: Problem Statement

The Osdag web application's module system lacked file management capabilities and required manual module identification for design report generation. The File menu dropdown was non-functional, and there was no CAD model download or screenshot capture functionality for 3D models.

## 6.2 Task 3: Tasks Done

Enhanced the design report API to automatically detect active module sessions from cookies. Implemented file management system supporting input operations (.osi format), design reports, log export, and 3D model download in multiple formats (OBJ, BREP, STEP, IGES). Developed screenshot capture functionality. Extended support across all four connection modules: FinPlate, CleatAngle, SeatedAngle, and EndPlate.

## 6.3 Task 3: Python and JavaScript Code

### 6.3.1 Description of the Enhancement

The solution consists of three main components:

- Automatic Module Detection: API identifies active modules through cookie analysis.
- File Management System: Complete file operations and model exports.
- 3D Model Integration: Screenshot capture and CAD download functionality.

### 6.3.2 Python Code: Design Report API Enhancement

Listing 6.1: Enhanced Design Report API

```
class CreateDesignReport(APIView):
def post(self, request):
metadata = request.data.get('metadata')
    # Map cookies to create functions
    module_cookie_map = {
        'fin_plate_connection_session': fin_plate_create_from_input,
        'end_plate_connection_session': end_plate_create_from_input,
        'cleat_angle_connection_session': cleat_angle_create_from_input
            ,
        'seated_angle_connection': seated_angle_create_from_input,
    }

    # Automatic module detection
    for cookie_key, create_func in module_cookie_map.items():
        if cookie_key in request.COOKIES:
            cookie_id = request.COOKIES.get(cookie_key)
            create_module_func = create_func
            break

    # Get design data from database
    designObject = Design.objects.get(cookie_id=cookie_id)
    input_values = designObject.input_values

    # Generate report
    try:
        module = create_module_func(input_values)
        resultBoolean = module.save_design(metadata_final)

```

```
28          if(resultBoolean):
29              return Response({'success': 'Design report created'})
30      except Exception as e:
31          return Response({"message": "Error generating report"})
```

### 6.3.3 Explanation of the Python Code

- Line 6-11: Module mapping system associating cookies with create functions for automatic detection.
- Line 14-18: Dynamic cookie detection loop identifying active module session.
- Line 21-22: Database integration to retrieve design parameters.
- Line 25-30: Module instantiation and report generation with file handling.

### 6.3.4 JavaScript Code: File Management System

Listing 6.2: File Operations Handler

```
1  const handleClick = (option) => {
2  switch (option.name) {
3  case 'Load Input':
4  loadInput();
5  break;
6  case 'Save Input':
7  saveInput();
8  break;
9  case 'Create Design Report':
10 setCreateDesignReportBool(true);
11 break;
12 case 'Save 3D Model':
13 (async () => {
14 const options = {
15 types: [
16 { description: "OBJ File", accept: { "application/octet-stream": [".obj
       "] }},
17 { description: "BREP File", accept: { "application/octet-stream": [".
       brep"] }},
```

```
18 { description: "STEP File", accept: { "application/octet-stream": [".
       step"] }},
19 ],
20 suggestedName: "3dmodel",
21 };
22     const handle = await window.showSaveFilePicker(options);
23     const blob = await downloadCADModel(handle.name.split(".").pop());
24
25     if (blob) {
26       const writable = await handle.createWritable();
27       await writable.write(blob);
28       await writable.close();
29     }
30   })();
31   break;
32 case 'Save Cad Image':
33   triggerScreenshotCapture();
34   break;
35 }
36 };
```

### 6.3.5 Explanation of the JavaScript Code

- Line 1-28: Switch statement handling file operations including input, design reports, and 3D model operations.
- Line 11-20: File picker implementation for 3D model downloads supporting multiple CAD formats.
- Line 26-27: Screenshot capture integration triggering image saving functionality.

### 6.3.6 JavaScript Code: Screenshot Capture

Listing 6.3: Screenshot Capture System

```
1 async function saveImageWithDialog(canvas) {
2 const options = {
3 types: [
4 { description: "PNG Image", accept: { "image/png": [".png"] }},
```

```
5  { description: "JPEG Image", accept: { "image/jpeg": [".jpeg"] }},
6  ],
7  suggestedName: "cad_model_snapshot",
8  };
9  const handle = await window.showSaveFilePicker(options);
10 const fileExtension = handle.name.split(".").pop();
11 const mimeType = fileExtension === "png" ? "image/png" : "image/jpeg";
12 const blob = await new Promise((resolve) => {
13 canvas.toBlob(resolve, mimeType);
14 });
15 const writable = await handle.createWritable();
16 await writable.write(blob);
17 await writable.close();
18 }
19 const ScreenshotCapture = ({ screenshotTrigger, selectedView }) => {
20 const { gl, invalidate } = useThree();
21 useEffect(() => {
22 if (screenshotTrigger && selectedView === "Model") {
23 invalidate();
24 saveImageWithDialog(gl.domElement);
25 }
26 }, [screenshotTrigger]);
27 return null;
28 };
```

### 6.3.7 Explanation of the Screenshot Code

- Line 2-8: File picker configuration supporting multiple image formats (PNG, JPEG).
- Line 10-12: MIME type detection and canvas-to-blob conversion for image output.
- Line 19-23: Screenshot capture logic with view validation and frame synchronization.

### 6.3.8 Visual Results

### 6.3.9 Task Outcome and Results

The module enhancement implementation successfully resolved the identified issues:

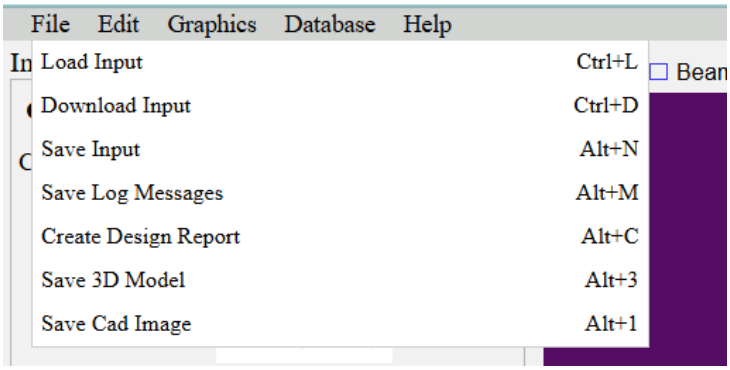


Figure 6.1: Enhanced File Menu Dropdown with complete file operations

- Automatic Module Detection: Design report API automatically identifies active modules through cookie analysis.
- File Operations: Complete input file management with load, save, and download capabilities.
- CAD Export: Multi-format 3D model downloads (OBJ, BREP, STEP, IGES).
- Screenshot Functionality: Image capture in multiple formats (PNG, JPEG, BMP, TIFF).
- Cross-Module Support: Consistent functionality across all four connection modules.

The enhanced system significantly improves workflow management and provides professional documentation capabilities.

# Chapter 7

# Module Creation

## 7.1 Problem Statement

The next task was to create complete new modules. Beam-to-Beam EndPlate and Beam-to-Beam CoverPlateBolted modules were both created in 5 steps: Basic Setup, InputDock creation, OutputDock creation and setting logs, rendering 3D models and finally fixing the design report function along with file functionalities.

## 7.2 Step 1: Basic Setup

### 7.2.1 Backend Setup - Module Registration

The basic setup involves creating module-specific files in both backend and frontend folders. We need to register our new module in the system and create the necessary file structure. First, go to osdag_api/module_finder.py and add the new module to the module dictionary:

Listing 7.1: Adding Module to Module Dictionary

```
%--------------------begin code-------------
module_dict : Dict[str, ModuleApiType] = {
'Fin Plate Connection': fin_plate_connection,
'End Plate Connection': end_plate_connection,
'Cleat Angle Connection': cleat_angle_connection,
'Seated Angle Connection': seated_angle_connection,
'Cover Plate Bolted Connection': cover_plate_bolted_connection,
```

```
}
%-------------------- end code ---------------
```

### 7.2.2 Creating Module API File

Go to osdag_api/modules folder and create a file named cover_plate_bolted_connection.py. Copy the content from any existing module file in that folder exactly as it is. This file will handle the API calls for our module.

### 7.2.3 Session Management Setup

Navigate to Osdag/web_api/session_api.py and add module details in both CreateSession and DeleteSession classes:

Listing 7.2: Adding Session Management for Cover Plate Module

```
%--------------------begin code-------------
elif request.COOKIES.get("cover_plate_bolted_connection_session") is
    not None:
print("cover plate bolted connection is there ")
return JsonResponse({"status": "set"}, status=200)
cookie_keys = {
"Fin Plate Connection": "fin_plate_connection_session",
"End Plate Connection": "end_plate_connection_session",
"Cleat Angle Connection": "cleat_angle_connection_session",
"Seated Angle Connection": "seated_angle_connection",
"Cover Plate Bolted Connection": "cover_plate_bolted_connection_session
    ",
}
%-------------------- end code ---------------
```

Also add the module details in the developed_modules section of the same file.

### 7.2.4 Creating Output View File

Create a file named coverplatebolted_outputView.py in the modules folder. Copy code from any existing module's outputView.py file and edit the module-specific details:

Listing 7.3: Module-Specific Output View Configuration

```
%--------------------begin code-------------
obtaining the session, module_id, input_values
cookie_id = request.COOKIES.get('cover_plate_bolted_connection_session'
    )
module_api = get_module_api('Cover Plate Bolted Connection')
input_values = request.data
tempData = {
'cookie_id': cookie_id,
'module_id': 'Cover Plate Bolted Connection',
'input_values': input_values
}
%-------------------- end code ---------------
```

### 7.2.5 Creating Input Data File

Go to osdag/web_api/inputdata and create cover_plate_bolted_input.py. Copy code from any existing module file. This file will be edited in step 2 when we configure dropdown lists.

### 7.2.6 URL Configuration

Add the module route in osdag/urls.py:

Listing 7.4: Adding URL Route for Module Access

```
%--------------------begin code-------------
path('calculate-output/Cover-Plate-Bolted-Connection',
CoverPlateBoltedOutputData.as_view(),name="Cover-Plate-Bolted-
    Connection"),
%-------------------- end code ---------------
```

### 7.2.7 Frontend Component Setup

Go to Osdagclient/src/components/shearConnection/ and create coverplatebolted.jsx. Copy code from any existing module like finplate.jsx to this file. Edit the session management code in coverplatebolted.jsx:

Listing 7.5: Frontend Session Management Configuration

```
%--------------------begin code-------------
useEffect(() => {
createSession("Cover Plate Bolted Connection");
}, []);
useEffect(() => {
return () => {
if (
location.pathname !=
"/design/connections/beam-to-beam-splice/cover_plate_bolted"
) {
deleteSession("Cover Plate Bolted Connection");
}
};
}, []);
%-------------------- end code ---------------
```

### 7.2.8 Module Context Integration

Edit osdagclient/src/context/moduleState.jsx and add the module to createSession and deleteSession APIs:

Listing 7.6: Module Context API Integration

```
%--------------------begin code-------------
else if (module_id == "Cover Plate Bolted Connection") {
getBeamMaterialList("Cover-Plate-Bolted-Connection");
}
%-------------------- end code ---------------
```

### 7.2.9 App Route Configuration

Finally, add the component route in App.jsx inside osdagclient folder:

Listing 7.7: Adding Component Route in App.jsx

```
%--------------------begin code-------------
import CoverPlateBolted from "./components/shearConnection/
    CoverPlateBolted";
<Route
```

```
4 path="/design/:designType/beam-to-beam-splice/cover_plate_bolted"
5 element={<CoverPlateBolted />}
6 />
7 %-------------------- end code ---------------
```

This completes the basic setup for the new module, creating all necessary files and connections between frontend and backend components.

## 7.3 Step 2: InputDock Creation

### 7.3.1 Finding Input Values from Desktop App

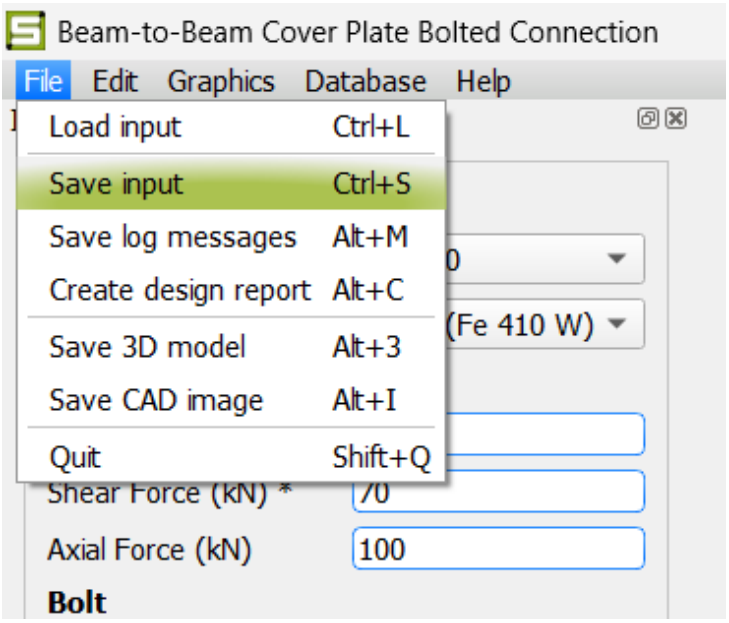


Figure 7.1: Save Input option in desktop Osdag application

To create the input form for the web app, we first need to know what input values the desktop app uses. We do this by:

- Opening the desktop Osdag app
- Going to the Beam-to-Beam Cover Plate Bolted module
- Clicking File ¿ Save Input to download a .osi file
- Opening this .osi file in notepad to see all the input parameters

### 7.3.2 Setting Up Input Values in Code

```
Connector.Material: E 250 (Fe 410 W)A
Connector.Web_Plate.Thickness_List: *id001
Design.Design_Method: Limit State Design
Detailing.Corrosive_Influences: 'No'
Detailing.Edge_type: Sheared or hand flame cut
Detailing.Gap: '3'
Load.Axial: '100'
Load.Moment: '50'
Load.Shear: '70'
Material: E 250 (Fe 410 W)A
Member.Designation: MB 300
Member.Material: E 250 (Fe 410 W)A
Module: Beam-to-Beam Cover Plate Bolted Connection
```

Figure 7.2: Input values from .osi file opened in notepad

Read the .osi file downloaded above, then edit the input values in the coverplate-bolted.jsx file, our React component that we created in step 1.

Listing 7.8: Setting Up Input Values for Cover Plate Module in coverplatebolted.jsx

```
%--------------------begin code-------------
const [inputs, setInputs] = useState({
bolt_hole_type: "Standard",
bolt_diameter: [],
bolt_grade: [],
bolt_slip_factor: "0.3",
bolt_tension_type: "Pre-tensioned",
bolt_type: "Bearing Bolt",
flange_plate_preferences: "Outside",
flange_plate_thickness: [],
connector_material: "E 250 (Fe 410 W)A",
web_plate_thickness: [],
design_method: "Limit State Design",
detailing_corr_status: "No",
detailing_edge_type: "Sheared or hand flame cut",
detailing_gap: "3",
load_axial: "100",
load_moment: "70",
load_shear: "50",
material: "E 250 (Fe 410 W)A",
member_designation: "MB 300",
member_material: "E 250 (Fe 410 W)A",
```

```
module: "Beam-to-Beam Cover Plate Bolted Connection",
});
%-------------------- end code ---------------
```

### 7.3.3 Getting Dropdown Lists from Backend

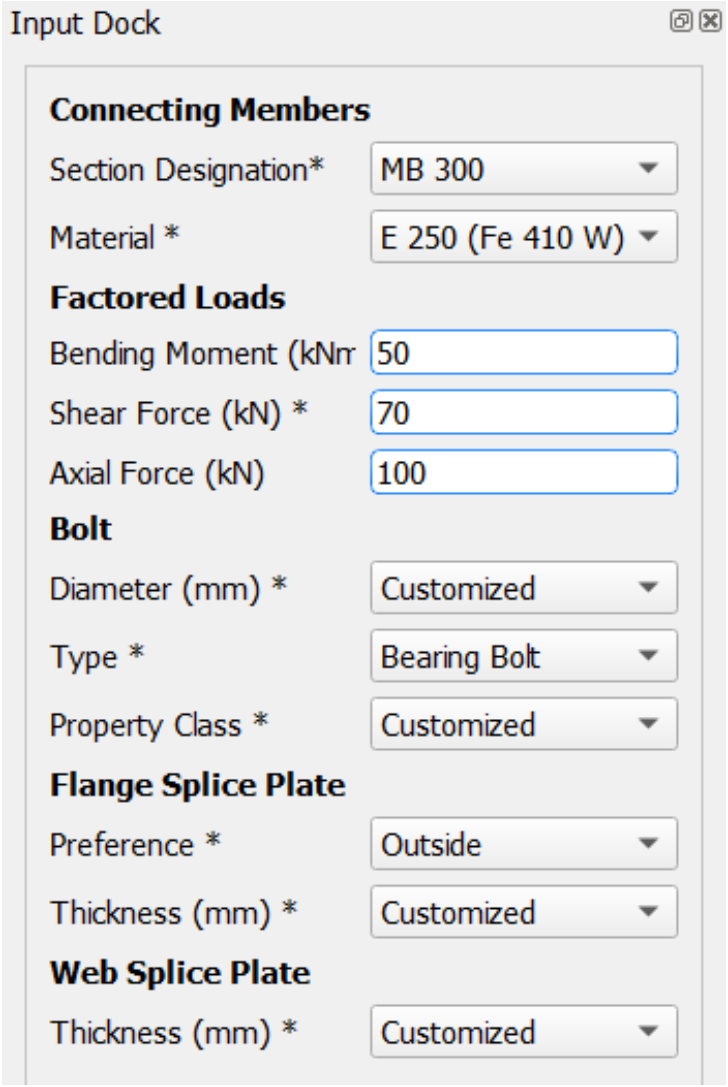


Figure 7.3: Input dock interface showing required dropdown lists

Then analyze the list requirements for the input form by checking all dropdown menus with options like beam sizes, materials, and bolt diameters. We get these lists from the backend using Context API inside the coverplatebolted.jsx file:

Listing 7.9: Getting Required Lists for Dropdown Menus

```
%--------------------begin code-------------
const {
```

```
beamList,
materialList,
boltDiameterList,
thicknessList,
propertyClassList,
createSession,
createDesign,
} = useContext(ModuleContext);
%-------------------- end code ---------------
```

### 7.3.4 Setting Up Backend Python File

After setting the APIs, we need to edit the file cover_plate_bolted.py inside osdag_api/modules that we created in step 1. Now create the instance of the BeamCoverPlate class:

Listing 7.10: Creating Backend Module Handler

```
%--------------------begin code-------------
from design_type.connection.beam_cover_plate import BeamCoverPlate

def create_module() -> BeamCoverPlate:
    """Creates the beam cover plate module"""
    module = BeamCoverPlate()
    module.set_osdaglogger(None)
    return module

def create_from_input(input_values: Dict[str, Any]) -> BeamCoverPlate:
    """Creates module from user input values"""
    try:
        module = create_module()
    except Exception as e:
        print('Error creating module:', e)
    return module
%-------------------- end code ---------------
```

### 7.3.5 Making Sure Everything Works Together

After setting up the input values, we need to set up the same input values in both the handleSubmit() and saveOutput() functions. This ensures that when a user fills out the form

and clicks the design button, all the data gets sent to the backend correctly.RetryClaude can make mistakes. Please double-check responses.Researchbeta Sonnet 4

## 7.4 Step 3: OutputDock Creation and Logs

### 7.4.1 Fixing Backend Errors

When you click the design button after setting input values, you will encounter errors that need to be fixed in the backend file created earlier.

Go to osdag_api/modules/cover_plate_bolted_connection.py file. First, update the validate_input_new function values according to the input values set in step 2. Then, work on the create_from_input() function:

Listing 7.11: Setting Input Values in Backend Module

```
%--------------------begin code-------------
def create_from_input(input_values: Dict[str, Any]) -> BeamCoverPlate:
"""Create an instance of the fin plate connection module design class
    from input values."""
# validate_input(input_values)
try :
module = create_module()  # Create module instance.
except Exception as e :
print('e in create_module : ' , e)
print('error in creating module')
# Set the input values on the module instance.
try :
    print("INPUT SET FOR FINAL OUTPUT",input_values)
    module.set_input_values(input_values)
except Exception as e :
    traceback.print_exc()
    print('e in set_input_values : ' , e)
    print('error in setting the input values')

return module
%-------------------- end code ---------------
```

### 7.4.2 Fixing Module-Specific Backend File

You need to go inside the module-specific backend file (BeamCoverPlate in this case) and remove self from some functions because we are creating an instance of the module. For example: Change:

- self.warn_text(self) to self.warn_text()
- self.member_capacity(self) to self.member_capacity()

The exact errors will be visible in the terminal, making it easy to identify which functions need fixing.

### 7.4.3 Setting Up Logs System

In the main backend file BeamCoverPlate, you need to create an empty array for logs in the initialization:

Listing 7.12: Initializing Logs Array in Module

```
%--------------------begin code-------------
def init(self):
super(BeamCoverPlate, self).init()
self.design_status = False
self.logs = []
%-------------------- end code ---------------
```

### 7.4.4 Converting Logger Statements to Logs Array

Wherever you see logger statements in the entire file, you need to append the same message to the logs array:

Listing 7.13: Converting Logger Statements to Logs Array

```
%--------------------begin code-------------
logger.warning(
" : You are using a section (in red color) that is not available in
   latest version of IS 808")
self.logs.append({"msg": " : You are using a section (in red color)
   that is not available in latest version of IS 808", "type": "warning
   "})
```

```
logger.info(
" : You are using a section (in red color) that is not available in
    latest version of IS 808")
self.logs.append({"msg": " : You are using a section (in red color)
    that is not available in latest version of IS 808", "type": "info"})
%-------------------- end code ---------------
```

This conversion is necessary because the desktop version of Osdag uses logger, but we need to append those print statements to an array to display them on the website. The logs will be returned to the main file for frontend display.

### 7.4.5 OutputDock Creation

Now check the generate_output function in the cover_plate_bolted_connection.py file, you have to create the OutputDock in the same way discussed in chapter 5.

Listing 7.14: Generate Output Function for OutputDock

```
%--------------------begin code-------------
def generate_output(input_values: Dict[str, Any]) -> Dict[str, Any]:
"""
Generate, format and return the input values from the given output
    values.
Output format (json): {
"Bolt.Pitch":
"key": "Bolt.Pitch",
"label": "Pitch Distance (mm)"
"value": 40
}
}
"""
print("************")
output = {}  # Dictionary for formatted values
module = create_from_input(input_values)  # Create module from input.
print('module : ' , module)
print('type of module : ' , type(module))
%-------------------- end code ---------------
```

### 7.4.6 Creating Frontend OutputDock Component

Create a module-specific OutputDock file named CoverPlateBoltedOutputdock.jsx inside osdagclient/src/components. This component will display the calculated results and logs on the frontend interface. The OutputDock creation process involves formatting the backend calculation functions results into a user-friendly interface that displays all the design parameters and their calculated values in an organized manner.

You're right, there's a LaTeX syntax error. Here's the corrected version:

## 7.5 Step 4: 3D Model Rendering

### 7.5.1 Backend CAD Model API Configuration

To render 3D models, start by configuring the backend CAD model API. Go to Osdag/web_api/cad_model_api.py and add the code for the coverplatebolted module. First, add the session cookie for the new module:

Listing 7.15: Adding Session Cookie for Cover Plate Module

```
%--------------------begin code-------------
def get(self, request: HttpRequest):
# Get design session id
fin_plate_cookie_id = request.COOKIES.get("fin_plate_connection_session")
cleat_angle_cookie_id = request.COOKIES.get("cleat_angle_connection_session")
end_plate_cookie_id = request.COOKIES.get("end_plate_connection_session")
seated_angle_cookie_id = request.COOKIES.get("seated_angle_connection")
cover_plate_bolted_cookie_id = request.COOKIES.get("cover_plate_bolted_connection_session")
%-------------------- end code ---------------
```

Then update the session validation check:

Listing 7.16: Session Validation Update

```
%--------------------begin code-------------
```

```
2 if not fin_plate_cookie_id and not cleat_angle_cookie_id and not
    seated_angle_cookie_id and not end_plate_cookie_id and not
    cover_plate_bolted_cookie_id :
3 return JsonResponse({"status": "error", "message": "Please open a
    module"}, status=400)
4 %-------------------- end code ---------------
```

### 7.5.2 Determining 3D Model Sections

To understand which sections need to be rendered, examine the original module backend file design_type/connection/beam_cover_plate.py and search for 3D functions:

Listing 7.17: 3D Components Configuration in Backend Module

```
1  %--------------------begin code-------------
2  def get_3d_components(self):
3  components = []
4  t1 = ('Model', self.call_3DModel)
5  components.append(t1)
6
7  t2 = ('Beam', self.call_3DBeam)
8  components.append(t2)
9
10 t4 = ('Cover Plate', self.call_3DPlate)
11 components.append(t4)
12
13 return components
14 def call_3DPlate(self, ui, bgcolor):
15 from PyQt5.QtWidgets import QCheckBox
16 from PyQt5.QtCore import Qt
17 for chkbox in ui.frame.children():
18 if chkbox.objectName() == 'Cover Plate':
19 continue
20 if isinstance(chkbox, QCheckBox):
21 chkbox.setChecked(Qt.Unchecked)
22 ui.commLogicObj.display_3DModel("Connector", bgcolor)
23 %-------------------- end code ---------------
```

Based on this analysis, the Cover Plate Bolted module has three sections: Model, Beam,

and Connector. Add the section configuration to the CAD model API:

Listing 7.18: Section Configuration for Cover Plate Module

```
%--------------------begin code-------------
elif session_type == "CoverPlateBolted":
sections = ["Model", "Beam", "Connector"]
%-------------------- end code ---------------
```

### 7.5.3 Frontend 3D Rendering Setup

Move to the frontend file coverplatebolted.jsx and configure the 3D rendering components:

Listing 7.19: 3D Rendering State Configuration

```
%--------------------begin code-------------
const [renderBoolean, setRenderBoolean] = useState(false);
const [modelKey, setModelKey] = useState(0);
const [loading, setLoading] = useState(false);
const [selectedView, setSelectedView] = useState("Model");
const options = ["Model", "Beam", "Connector"];
const { position: cameraPos, fov } = useViewCamera("CoverPlateBolted",
    selectedView);
const cameraRef = useRef();
const [screenshotTrigger, setScreenshotTrigger] = useState(false);
const triggerScreenshotCapture = () => {
setScreenshotTrigger(true);
};
%-------------------- end code ---------------
```

### 7.5.4 3D Canvas and Model Display

Configure the 3D canvas with section selection and model rendering:

Listing 7.20: 3D Canvas Configuration with Section Selection

```
%--------------------begin code-------------
<div className="superMainBody_mid">
  <div className="options-container">
    {options.map((option) => (
      <div
```

```
        key={option}
        className="option-wrapper"
        onClick={() => setSelectedView(option)}
      >
        <div
          className={`option-box ${
            selectedView === option ? "selected" : ""
          }`}
        ></div>
        <span className="option-label">{option}</span>
      </div>
    ))}
  </div>
  {loading ? (
    <div className="modelLoading">
      <p>Loading Model...</p>
    </div>
  ) : renderBoolean ? (
    <div className="cadModel">
      <Canvas gl={{ antialias: true, preserveDrawingBuffer: true }}
      onCreated={({ gl }) => {
        gl.setClearColor("#ADD8E6");
      }}>
        <PerspectiveCamera
          ref={cameraRef}
          makeDefault
          position={cameraPos}
          fov={fov}
          near={0.1}
          far={1000}
        />
        <Suspense
          fallback={
            <Html>
              <p>Loading 3D Model...</p>
            </Html>
          }
        >
          <Model
```

```
45              modelPaths={cadModelPaths}
46              selectedView={selectedView}
47              key={modelKey}
48            />
49            <ScreenshotCapture
50              screenshotTrigger={screenshotTrigger}
51              setScreenshotTrigger={setScreenshotTrigger}
52              selectedView={selectedView}
53            />
54          </Suspense>
55        </Canvas>
56      </div>
57    ) : (
58      <div className="modelback"></div>
59    )}
60  </div>
61  %-------------------- end code ---------------
```

The 3D model rendering system allows users to view different sections of the structural connection (Model, Beam, Connector) with interactive controls and screenshot capabilities. The implementation follows the same pattern established in Chapter 4 for consistent 3D visualization across all modules.

## 7.6 Step 5: Fixing DesignReport Function and File Functionalities

### 7.6.1 Problem with DesignReport Function

The DesignReport function was not working because the backend API couldn't determine if the report generation was successful. The save_design() function in the main module file was missing a return statement to confirm successful completion. Follow the steps in Chapter 6 to fix designreport and file functionalities.

### 7.6.2 Fixing the save_design Function

Navigate to the main backend file design_type/connection/beam_cover_plate.py and locate the save_design() function in the BeamCoverPlate class. The function currently looks like this:

Listing 7.21: Original save_design Function

```
%--------------------begin code-------------
class BeamCoverPlate(MomentConnection):
def save_design(self, popup_summary):
    fname_no_ext = popup_summary['filename']

    CreateLatex.save_latex(CreateLatex(), self.report_input, self.
        report_check, popup_summary,
                           self.rel_path, Disp_2d_image, Disp_3d_image,
                              module=self.module)
%-------------------- end code ---------------
```

### 7.6.3 Adding Return True Statement

Add a return True statement at the end of the save_design() function. This tells the backend DesignReport API that the function has successfully created the report:

Listing 7.22: Fixed save_design Function with Return Statement

```
%--------------------begin code-------------
class BeamCoverPlate(MomentConnection):
def save_design(self, popup_summary):
    fname_no_ext = popup_summary['filename']

    CreateLatex.save_latex(CreateLatex(), self.report_input, self.
        report_check, popup_summary,
                           self.rel_path, Disp_2d_image, Disp_3d_image,
                              module=self.module)


    return True
%-------------------- end code ---------------
```

### 7.6.4 Understanding File Functionalities

The file functionalities for the Cover Plate Bolted module work the same way as other modules in Osdag. By following the implementation patterns from Chapter 6, you can understand how these features work:

- Save Input - saves current design parameters to a .osi file
- Load Input - loads design parameters from a saved .osi file
- Save Design Report - generates and downloads the complete PDF design report
- Save 3D Model - exports the 3D CAD model files
- Save CAD Image - saves screenshots of the 3D model views

### 7.6.5 Backend-Frontend Communication

The return True statement ensures proper communication between the frontend and backend when generating design reports. Without this return statement, the frontend cannot confirm whether the report generation was successful, causing the functionality to appear broken to users. By following Chapter 6's detailed explanations and adding this simple return statement, both the DesignReport function and all file functionalities will work correctly for the Cover Plate Bolted module.

# Chapter 8

# Modular Component Architecture and State Management

## 8.1 Task 3: Problem Statement

The Osdag web application initially had monolithic component files like CoverPlate-Bolted.jsx (1000+ lines) where each engineering module was implemented as a separate, complete component. This created major problems: code duplication across modules, inconsistent behavior, difficult maintenance, and slow development of new modules. Each module (FinPlate, BeamBeamEndPlate, CoverPlateBolted) had nearly identical patterns but was written separately, leading to repeated code and inconsistent user experiences.

## 8.2 Task 3: Tasks Done

Transformed the monolithic structure into a modular, three-tier architecture with shared components, configuration-driven modules, and reusable hooks. Created universal components that work for all modules while allowing complete customization through configuration files. Successfully restructured three existing modules to use the new architecture, reducing code from 1000+ lines per module to under 20 lines while maintaining full functionality. Established a pattern where new modules can be added by simply creating configuration files without writing new UI components.

## 8.3 Task 3: Architecture Overview

The new modular structure follows a clear three-tier pattern:

Listing 8.1: Frontend Modular Architecture

```
%--------------------begin code-------------
src/modules/
          shared/                          #     Universal components & logic
                components/
                      EngineeringModule.jsx    #     Main universal component
                      InputSection.jsx         #     Dynamic input rendering
                      BaseOutputDock.jsx       #     Universal output with modals
                      CustomizationModal.jsx   #     Reusable modal component
                      DesignReportModal.jsx    #     PDF report generation
                hooks/
                    useEngineeringModule.js  #     Centralized state management
          coverPlateBolted/                #     Module-specific implementations
                CoverPlateBolted.jsx         #     20-line entry component
                configs/
                      coverPlateBoltedConfig.js      #     Input & validation config
                      coverPlateBoltedOutputConfig.js #     Output & modal config
                components/
                    CoverPlateBoltedOutputDock.jsx #     Output wrapper component
          beamBeamEndPlate/                #     Another module following same pattern
          BeamBeamEndPlate.jsx
          configs/
```

```
                beamBeamEndPlateConfig.js
                beamBeamEndPlateOutputConfig.js
          components/
          BeamBeamEndPlateOutputDock.jsx
%-------------------- end code ---------------
```

### 8.3.1 How the Architecture Works

The modular system has three key layers:

- Shared Layer: Contains universal components that work for all modules
- Configuration Layer: Each module defines its behavior through config files
- Module Layer: Minimal entry components that connect configs to shared components

### 8.3.2 Universal Engineering Module

The core component that handles all modules:

Listing 8.2: Universal Module Component

```
%--------------------begin code-------------
export const EngineeringModule = ({
moduleConfig,           //     Configuration object defining behavior
OutputDockComponent,    //     Module-specific output component
menuItems,             //     Navigation menu items
title,                 //     Module title
}) => {
// Get all state and functions from centralized hook
const {
inputs, setInputs, output, loading, handleSubmit, handleReset
} = useEngineeringModule(moduleConfig);
return (
<div className="module_base">
{/* Navigation Menu */}
<div className="module_nav">
{menuItems.map((item, index) => (
<MomentDropdownMenu key={index} {...item} />
```

```
18 ))}
19 </div>
20   <div className="superMainBody">
21     {/* Dynamic Input Section */}
22     <div className="InputDock">
23       {moduleConfig.inputSections.map((section, index) => (
24         <InputSection
25           key={index}
26           section={section}          //     Config defines fields
27           inputs={inputs}
28           setInputs={setInputs}
29         />
30       ))}
31     </div>
32
33     {/* 3D Model Viewer */}
34     <div className="superMainBody_mid">
35       <Canvas>
36         <Model modelPaths={cadModelPaths} selectedView={selectedView}
               />
37       </Canvas>
38     </div>
39
40     {/* Output Display */}
41     <div className="superMain_right">
42       <OutputDockComponent output={output} />
43     </div>
44   </div>
45 </div>
46 );
47 };
48 %-------------------- end code ---------------
```

### 8.3.3 Explanation of Universal Component

- Line 1-5: Takes configuration and components as props, making it work for any module

- Line 7-9: Uses shared hook for consistent state management across all modules
- Line 17-24: Renders input sections dynamically based on module configuration
- Line 27-31: Universal 3D model rendering that works with all connection types
- Line 34-36: Module-specific output component while maintaining consistent layout

### 8.3.4 Configuration-Driven Module Setup

Each module is defined through simple configuration objects:

Listing 8.3: Module Configuration Example

```
%--------------------begin code-------------
export const coverPlateBoltedConfig = {
sessionName: "Cover Plate Bolted Connection",
designType: "Cover-Plate-Bolted-Connection",
// Default values when module loads
defaultInputs: {
bolt_diameter: [],
load_shear: "50",
material: "E 250 (Fe 410 W)A",
member_designation: "MB 300",
},
// Define input validation rules
validateInputs: (inputs) => {
if (!inputs.member_designation || inputs.load_shear === "") {
return { isValid: false, message: "Please input all the fields" };
}
return { isValid: true };
},
// Define input sections and fields
inputSections: [
{
title: "Connecting Members",
fields: [
{
key: "member_designation",
label: "Section Designation*",
type: "select",
```

```
options: "beamList",     //     Links to data source
required: true
},
{
key: "load_shear",
label: "Shear Force(kN)",
type: "number"
}
]
}
]
};
%-------------------- end code ---------------
```

### 8.3.5 Explanation of Configuration System

- Line 1-3: Basic module identification and API configuration
- Line 5-10: Default input values that initialize the form fields
- Line 12-17: Custom validation function specific to this module's requirements
- Line 19-34: Input field definitions that automatically generate the UI
- Line 26-29: Field configuration links to shared data sources like beam lists

### 8.3.6 Simple Module Entry Point

Each module needs only a minimal entry component:

Listing 8.4: Minimal Module Implementation

```
%--------------------begin code-------------
import React from 'react';
import { EngineeringModule } from '../shared/components/
    EngineeringModule';
import { coverPlateBoltedConfig } from './configs/
    coverPlateBoltedConfig';
import CoverPlateBoltedOutputDock from './components/
    CoverPlateBoltedOutputDock';
```

```
import { menuItems } from '../shared/utils/moduleUtils';
function CoverPlateBolted() {
return (
<EngineeringModule
moduleConfig={coverPlateBoltedConfig}        //     Module behavior
OutputDockComponent={CoverPlateBoltedOutputDock}  //     Output display
menuItems={menuItems}                        //     Navigation
title="Cover Plate Bolted Connection"        //     Page title
/>
);
}
export default CoverPlateBolted;
%-------------------- end code ---------------
```

### 8.3.7 Dynamic Output System

The BaseOutputDock component handles all output display needs:

Listing 8.5: Universal Output Component

```
%--------------------begin code-------------
export const BaseOutputDock = ({ output, outputConfig }) => {
const [activeModals, setActiveModals] = useState({});
const handleDialog = (key) => {
const modalConfig = outputConfig.modals[key];
if (modalConfig) {
openModal(modalConfig.type, key);    //     Open appropriate modal
}
};
const renderModalContent = (modalType, activeSection, output) => {
const config = outputConfig.modalTypes[modalType];
if (config.layout === "two-column") {
  return (
    <div className="spacing-main-body">
      <div className="spacing-left-body">
        {fieldsData.map(({ key, label }) => (
          <div className="spacing-left-body-align">
            <h4>{label}</h4>
            <Input value={output[key]?.val || ""} disabled />
          </div>
```

```
21          ))}
22        </div>
23        <div className="spacing-right-body">
24          <img src={getImageForModal('spacing')} alt="Spacing" />
25        </div>
26      </div>
27    );
28  }
29  };
30  return (
31  <div className="subMainBody scroll-data">
32  {Object.entries(outputConfig.sections).map(([sectionName, fields]) => (
33  <div key={sectionName}>
34  <h3>{sectionName}</h3>
35  {fields.map((field) => (
36  <Input
37  value={output[field.key]?.val || ""}
38  onClick={() => handleDialog(field.key)}    //     Modal trigger
39  />
40  ))}
41  </div>
42  ))}
43  </div>
44  );
45  };
46  %-------------------- end code ---------------
```

### 8.3.8 Explanation of Output System

- Line 1-2: Accepts output data and configuration to render any module's results
- Line 4-8: Handles modal opening based on field configuration
- Line 10-26: Renders different modal layouts (two-column, single-column, etc.) based on config
- Line 29-38: Dynamically generates output sections and fields from configuration
- Line 36: Automatically handles modal triggers for detailed views

### 8.3.9 Benefits of the New Architecture

The modular system provides significant improvements:

- Code Reduction: From 1000+ lines per module to under 20 lines
- No Duplication: All modules share the same core components
- Easy Addition: New modules require only configuration files
- Consistent Experience: All modules behave and look the same
- Easy Maintenance: Bug fixes automatically apply to all modules
- Better Testing: Shared components can be tested once for all modules

### 8.3.10 Task Outcome and Results

The modular architecture transformation successfully achieved:

- Successfully Restructured: More modules (FinPlate, BeamBeamEndPlate, CoverPlateBolted and many more) now use the shared architecture
- Eliminated Duplication: Removed over 2500 lines of repeated code across modules
- Consistent UI: All modules share the same look, feel, and behavior patterns
- Easier Maintenance: Changes to shared components automatically benefit all modules
- Future-Ready: Easy to add new field types, validation rules, and features

This architecture has transformed development from building complete components to simply configuring behavior, making the codebase more maintainable and enabling simple addition of new engineering modules.

# Chapter 9

# Conclusions

## 9.1 Tasks Accomplished

During this internship, I successfully completed several critical tasks that enhanced the Osdag web application's functionality and architecture. I developed a comprehensive PDF installation manual with step-by-step setup instructions, system requirements, and troubleshooting guides to make the software accessible to users with varying technical expertise. One of my major accomplishments was fixing the completely non-functional CAD Model Generation API and implementing a robust 3D rendering system using React Three Fiber. This involved restoring model generation for all connection types, implementing proper session management, fixing BREP to OBJ conversion pipeline, and enhancing geometry parsing with proper material differentiation for visual components. I transformed the static output display system into a dynamic, configuration-driven BaseOutputDock architecture that supports multiple modal layouts with automatic field mapping. Additionally, I enhanced existing modules with improved validation systems, implemented comprehensive file management capabilities including upload/download functionality and CSV export, and integrated customizable design report generation. The most significant achievement was successfully restructuring the entire application from monolithic components to a scalable, modular architecture. I transformed three existing modules (FinPlate, BeamBeamEndPlate, CoverPlateBolted) into a shared component system, eliminating over 2500 lines of duplicated code and reducing module implementation complexity from 1000+ lines to under 20 lines per module while maintaining full functionality.

## 9.2 Skills Developed

This internship provided extensive learning opportunities that significantly enhanced my technical expertise across multiple domains. I gained advanced proficiency in React.js, mastering component architecture, hooks, state management, and modern React patterns. Working with React Three Fiber introduced me to 3D graphics programming, geometry manipulation, material rendering, and creating interactive 3D applications for engineering visualization. My JavaScript skills evolved considerably as I worked with ES6+ features, asynchronous programming concepts, and functional programming patterns. I also developed strong CSS skills for advanced styling techniques, responsive design, and component-based styling systems. On the backend side, I learned Django integration, working with REST APIs, session management, and handling complex data flows between frontend and backend systems. The project exposed me to crucial software engineering concepts including modular architecture design, configuration-driven development patterns, and component reusability principles. I gained valuable experience in code refactoring, transforming legacy monolithic code into maintainable, shared components while implementing centralized state management and optimizing data flow patterns. Beyond technical skills, I developed strong problem-solving abilities through system design challenges, performance optimization tasks, and complex integration work with CAD systems and file format handling. The experience of building and restructuring complex engineering applications has provided me with a solid foundation in modern web development and software architecture, preparing me for future challenges in software engineering and technical problem-solving.

# Chapter A

# Appendix

## A.1 Work Reports

| | InternshipWork | | |
|---|---|---|---|
| **Name** | **Faran Imam** | | |
| **Mentor** | **Mr. Parth** | | |
| **Project** | **Osdag** | | |
| **Intership** | **FOSSEE Semester Long Internship 2025** | | |
| **Date** | **Day** | **Task** | **Work Hours** |
| 11 February 2025 | Tuesday | Joining \| Installed Osdag \| Did Initial Testing | 4hrs |
| 12 February 2025 | Wednesday | Reading and understanding Reports and CAD file | 4hrs |
| 13 February 2025 | Thursday | Understanding and Report creation | 4hrs |
| 14 February 2025 | Friday | Understanding and Report creation | 5hrs |
| 15 February 2025 | Saturday | Understanding and Report creation | 8hrs |
| 16 February2025 | Saturday | Understanding and Report creation | 12hrs |
| 17 February 2025 | Sunday | Holiday | |
| 18 February 2025 | Monday | Exam Leave(google meet attended) | |
| 19 February 2025 | Tuesday | Exam Leave(google meet attended) | |
| 20 February 2025 | Wednesday | Exam Leave(google meet attended) | |
| 21 February 2025 | Thursday | Exam Leave(google meet attended) | |
| 22 February 2025 | Friday | Exam Leave | |
| 23 February 2025 | Saturday | Exam Leave | |
| 24 February 2025 | Sunday | Osdag-web setup on local | 4hrs |
| 25 February 2025 | Monday | Osdag-web setup on local | 4hrs |
| 26 February 2025 | Tuesday | Osdag-web setup on local | 4hrs |
| 27 February 2025 | Wednesday | Osdag-web setup on local | 4hrs |
| 28 February 2025 | Thursday | Osdag-web setup on local | 4hrs |
| 1 March 2025 | Friday | Creating the Installing Manual of Osdag-web | 4hrs |
| 2 March 2025 | Saturday | Creating the Installing Manual of Osdag-web | 4hrs |
| 3 March 2025 | Sunday | Leaning react three fibre and understanding the Osdag-web code. | 4hrs |
| 4 March 2025 | Monday | Leaning react three fibre and understanding the Osdag-web code. | 4hrs |
| 5 March 2025 | Tuesday | Leaning react three fibre and understanding the Osdag-web code. | 4hrs |
| 6 March 2025 | Wednesday | Leaning react three fibre and understanding the Osdag-web code. | 4hrs |

| | InternshipWork | | |
|---|---|---|---|
| **Name** | **Faran Imam** | | |
| **Mentor** | **Mr. Parth** | | |
| **Project** | **Osdag** | | |
| **Intership** | **FOSSEE Semester Long Internship 2025** | | |
| 7 March 2025 | Thursday | Leaning react three fibre and understanding the Osdag-web code. | 4hrs |
| 8 March 2025 | Friday | Leaning react three fibre and understanding the Osdag-web code. | 4hrs |
| 9 March 2025 | Saturday | Fixing Backend API for rendering 3d models | 4hrs |
| 10 March 2025 | Sunday | Fixing Backend API for rendering 3d models | 4hrs |
| 11 March 2025 | Monday | Fixing Backend API for rendering 3d models | 4hrs |
| 12 March 2025 | Tuesday | Fixing Frontend to render the 3d models and design it. | 4hrs |
| 13 March 2025 | Wednesday | Fixing Frontend to render the 3d models and design it. | 4hrs |
| 14 March 2025 | Thursday | Fixing Frontend to render the 3d models and design it. | 4hrs |
| 15 March 2025 | Friday | Finplate model working, fixing cleat angle | 4hrs |
| 16 March 2025 | Saturday | Fixing cleat Angle | 4hrs |
| 17 March 2025 | Sunday | Again fixing the Installation Manual of Osdag-web | 4hrs |
| 18 March 2025 | Monday | Again fixing the Installation Manual of Osdag-web | 4hrs |
| 19 March 2025 | Tuesday | Fixing Seated Angle and End Plate Module | 4hrs |
| 20 March 2025 | Wednesday | Fixing Seated Angle and End Plate Module | 4hrs |
| 21 March 2025 | Thursday | Exam Leave | |
| 22 March 2025 | Friday | Exam Leave | |
| 23 March 2025 | Saturday | Exam Leave | |
| 24 March 2025 | Sunday | Exam Leave | |
| 25 March 2025 | Monday | Exam Leave | |
| 26 March 2025 | Tuesday | Fixing Seated Angle | 6hrs |
| 27 March 2025 | Wednesday | Fixing Seated Angle | 6hrs |
| 28 March 2025 | Thursday | Fixing End Plate | 6hrs |
| 29 March 2025 | Friday | Fixing End Plate | 6hrs |
| 30 March 2025 | Saturday | Fixing End Plate | 6hrs |
| 31 March 2025 | Sunday | Eid Leave | |

| | InternshipWork | | |
|---|---|---|---|
| **Name** | **Faran Imam** | | |
| **Mentor** | **Mr. Parth** | | |
| **Project** | **Osdag** | | |
| **Intership** | **FOSSEE Semester Long Internship 2025** | | |
| 1 April 2025 | Monday | Eid Leave | |
| 2 April 2025 | Tuesday | Fixing the whole design of Osdag-web | 5hrs |
| 3 April 2025 | Wednesday | Fixing the whole design of Osdag-web | 5hrs |
| 4 April 2025 | Thursday | Fixing the whole design of Osdag-web | 5hrs |
| 5 April 2025 | Friday | Fixing the whole design of Osdag-web | 5hrs |
| 6 April 2025 | Saturday | Fixing the designing of 3d models | 5hrs |
| 7 April 2025 | Sunday | Fixing the designing of 3d models | 5hrs |
| 8 April 2025 | Monday | Fixing Output Dock of Seated angle and cleat Angle | 6hrs |
| 9 April 2025 | Tuesday | Fixing Output Dock of Seated angle and cleat Angle | 6hrs |
| 10 April 2025 | Wednesday | Fixing Output Dock of Seated angle and cleat Angle | 6hrs |
| 11 April 2025 | Thursday | Creating CoverPlateBolted Module | 4hrs |
| 12 April 2025 | Friday | Creating CoverPlateBolted Module | 4hrs |
| 13 April 2025 | Saturday | Creating CoverPlateBolted Module | 4hrs |
| 14 April 2025 | Sunday | Creating CoverPlateBolted Module | 4hrs |
| 15 April 2025 | Monday | Creating CoverPlateBolted Module | 4hrs |
| 16 April 2025 | Tuesday | Creating CoverPlateBolted Module | 4hrs |
| 17 April 2025 | Wednesday | Creating CoverPlateBolted Module | 4hrs |
| 18 April 2025 | Thursday | Creating CoverPlateBolted Module | 4hrs |
| 19 April 2025 | Friday | Creating CoverPlateBolted Module | 4hrs |
| 20 April 2025 | Saturday | Creating CoverPlateBolted Module | 4hrs |
| 21 April 2025 | Sunday | Creating CoverPlateBolted Module | 4hrs |
| 22 April 2025 | Monday | Creating CoverPlateBolted Module | 4hrs |
| 23 April 2025 | Tuesday | Creating CoverPlateBolted Module | 4hrs |
| 24 April 2025 | Wednesday | Creating CoverPlateBolted Module | 4hrs |
| 25 April 2025 | Thursday | Creating CoverPlateBolted Module | 4hrs |

| | InternshipWork | | |
|---|---|---|---|
| **Name** | **Faran Imam** | | |
| **Mentor** | **Mr. Parth** | | |
| **Project** | **Osdag** | | |
| **Intership** | **FOSSEE Semester Long Internship 2025** | | |
| 26 April 2025 | Friday | Creating BeamtoBeam End Plate Module | 4hrs |
| 27 April 2025 | Saturday | Creating BeamtoBeam End Plate Module | 4hrs |
| 28 April 2025 | Sunday | Creating BeamtoBeam End Plate Module | 4hrs |
| 29 April 2025 | Monday | Creating BeamtoBeam End Plate Module | 4hrs |
| 30 April 2025 | Tuesday | Creating BeamtoBeam End Plate Module | 4hrs |
| 1 May 2025 | Wednesday | Creating BeamtoBeam End Plate Module | 4hrs |
| 2 May 2025 | Thursday | Creating BeamtoBeam End Plate Module | 4hrs |
| 3 May 2025 | Friday | Creating BeamtoBeam End Plate Module | 4hrs |
| 4 May 2025 | Saturday | Creating BeamtoBeam End Plate Module | 4hrs |
| 5 May 2025 | Sunday | Exam Leave | |
| 6 May 2025 | Monday | Exam Leave | |
| 7 May 2025 | Tuesday | Exam Leave | |
| 8 May 2025 | Wednesday | Exam Leave | |
| 9 May 2025 | Thursday | Exam Leave | |
| 10 May 2025 | Friday | Exam Leave | |
| 11 May 2025 | Saturday | Exam Leave | |
| 12 May 2025 | Sunday | Exam Leave | |
| 13 May 2025 | Monday | Exam Leave | |
| 14 May 2025 | Tuesday | Exam Leave | |
| 15 May 2025 | Wednesday | Creating BeamtoBeam End Plate Module | 4hrs |
| 16 May 2025 | Thursday | Creating BeamtoBeam End Plate Module | 4hrs |
| 17 May 2025 | Friday | Creating BeamtoBeam End Plate Module | 4hrs |
| 18 May 2025 | Saturday | Creating BeamtoBeam End Plate Module | 4hrs |
| 19 May 2025 | Sunday | Creating BeamtoBeam End Plate Module | 4hrs |
| 20 May 2025 | Monday | Restructing the Code | 6hrs |

| | | | |
|---|---|---|---|
| | **InternshipWork** | | |
| **Name** | **Faran Imam** | | |
| **Mentor** | **Mr. Parth** | | |
| **Project** | **Osdag** | | |
| **Intership** | **FOSSEE Semester Long Internship 2025** | | |
| 21 May 2025 | Tuesday | Restructing the Code | 6hrs |
| 22 May 2025 | Wednesday | Restructing the Code | 6hrs |
| 23 May 2025 | Thursday | Restructing the Code | 6hrs |
| 24 May 2025 | Friday | Restructing the Code | 6hrs |
| 25 May 2025 | Saturday | Restructing the Code | 6hrs |
| 26 May 2025 | Sunday | Restructing the Code | 6hrs |
| 27 May 2025 | Monday | Restructing the Code | 6hrs |
| 28 May 2025 | Tuesday | Restructing the Code | 6hrs |
| 29 May 2025 | Wednesday | Restructing the Code | 6hrs |
| 30 May 2025 | Thursday | Restructing the Code | 6hrs |
| 31 May 2025 | Friday | Restructing the Code | 6hrs |