FOSSEE Autumn Semester long Internship Report

On

Development of Website version for Osdag

Submitted by

Abhijith Sogal V

3rd Year B.Tech Student, Department of Civil

National Institute of Technology Karnataka, Surathkal

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 9, 2026

Acknowledgments

- I express my sincere gratitude to all those who supported me throughout this fellowship. This journey would not have been possible without the guidance and encouragement of many individuals, and I am deeply thankful for the opportunity to contribute to this project.
- Project staff in the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- I would also like to thank my colleagues and fellow interns for the collaborative spirit and for making this a memorable and enriching experience

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.

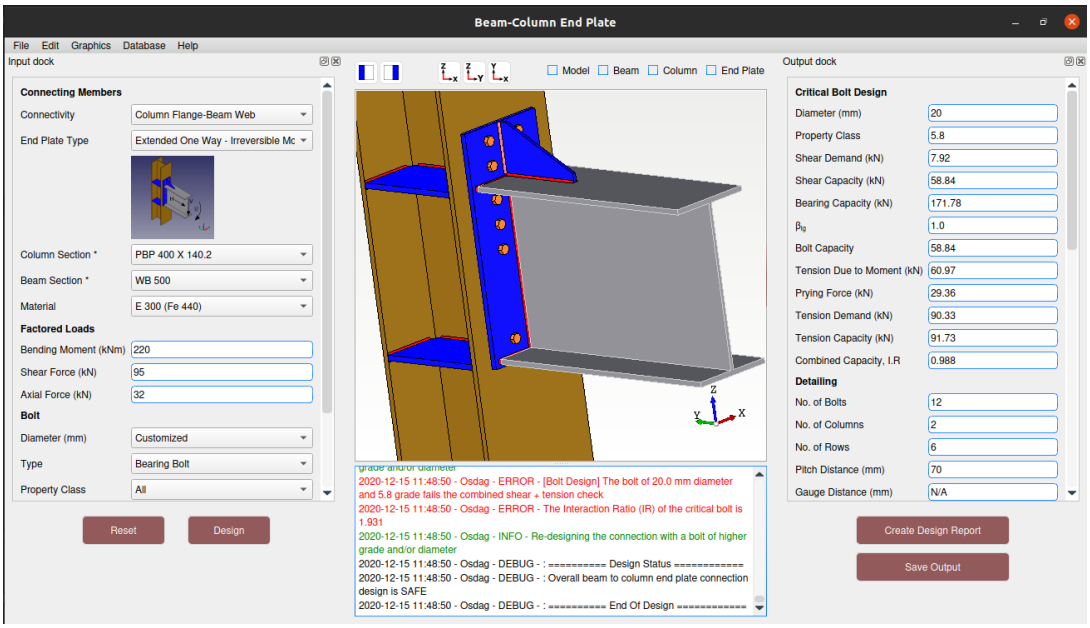

Figure 1.2: Osdag GUI

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

Project Architecture and Core Updates

2.1 Task 1: Problem Statement

The Osdag Web application required significant architectural improvements to maintain security, improve performance, and ensure scalability. The existing system had outdated dependencies, unclear separation between calculation logic and user interface, and lacked comprehensive documentation of the architecture. The main challenges included:

- Outdated Django framework and Python dependencies with potential security vulnerabilities
- Unclear project structure making it difficult for new developers to understand the codebase
- Lack of documentation explaining how frontend and backend communicate
- Need for a modular architecture that allows easy addition of new design modules

2.2 Task 1: Tasks Done

2.2.1 Backend Dependency Upgrades

To keep the system secure and compatible with modern standards, the backend framework was upgraded.

- **Django Upgrade:** The Django version and other Python dependencies were upgraded. This ensures the backend is faster and doesn't have old security vulnerabilities.
- **Refactoring:** The `moduleContext` and `useEngineeringModule.js` files were cleaned up. This reduces the amount of messy code and makes the state management (how the app remembers data) much smoother.

2.2.2 Backend Django Architecture

The backend follows a modular architecture that separates routing, views, module resolution, and calculation engines. This design allows new modules to be added with minimal changes to the core system.

URL Routing System

Django's URL routing maps HTTP requests to view classes. Each design module has a dedicated endpoint under the `calculate-output/` path prefix. The routing is defined in `osdag/urls.py`:

- **Module-specific routes:** Each module has its own endpoint, for example:
  - `calculate-output/FinPlateConnection` → `OutputData.as_view()`
  - `calculate-output/Simply-Supported-Beam` → `SimplySupportedBeamOutputData.as_vie`
  - `calculate-output/Cleat-Angle-Connection` → `CleatAngleOutputData.as_view()`
- **Generic routes:** Common endpoints for all modules:
  - `populate` → Input data population (dropdowns, lists)
  - `design/cad/` → CAD model generation
  - `api/projects/` → Project management
  - `api/module-routes/` → Module routing information

Backend API Flow

The backend architecture provides a comprehensive set of API endpoints that handle all aspects of the design process. The following endpoints are defined in `osdag/urls.py`:

Input Data and Population Endpoints:

- **populate** - Input Data Population
  - **View File:** `osdag/web_api/inputData_view.py` (InputData class)
  - **Method:** GET
  - **Purpose:** Populates dropdown lists and input options for the frontend
  - **Parameters:** `moduleName` (module identifier)
  - **Response:** Returns lists of sections (beams, columns, angles), materials, bolt options, thicknesses, etc.
- **design/input_values/** - Input Values API
  - **View File:** `osdag/web_api/input_data_api.py` (InputValues class)
  - **Method:** GET/POST
  - **Purpose:** Retrieves or sets input values for design modules
- **design** - Design View
  - **View File:** `osdag/web_api/inputData_view.py` (DesignView class)
  - **Method:** GET/POST
  - **Purpose:** Handles design-related operations

Output Calculation Endpoints:

- **calculate-output/¡Module-Key¿** - Design Calculation Examples
  - **View Files:**
    - `osdag/web_api/outputCalc_view.py` (OutputData) - For FinPlateConnection
    - `osdag/web_api/endplate_outputView.py` (EndPLateOutputData) - For End-Plate-Connection
    - `osdag/web_api/cleatangle_outputView.py` (CleatAngleOutputData) - For Cleat-Angle-Connection

        * `osdag/web_api/seatedangle_outputView.py` (SeatedAngleOutputData) - For SeatedAngleConnection
    - **Method:** POST
    - **Purpose:** Performs design calculations and returns results
    - **Parameters:** Module key and all input values (JSON body)
    - **Response:** Returns formatted output data, logs, and success status
- **design/output_values/** - Output Values API
    - **View File:** `osdag/web_api/output_data_api.py` (OutputValues class)
    - **Method:** GET/POST
    - **Purpose:** Retrieves or processes output values

CAD Model Endpoints:

- **design/cad/** - CAD Model Generation
    - **View File:** `osdag/web_api/cad_model_api.py` (CADGeneration class)
    - **Method:** POST
    - **Purpose:** Generates 3D CAD models for visualization
    - **Parameters:** Module inputs and section names
    - **Response:** Returns base64-encoded STL files for each section (Model, Beam, Column, Plate, etc.)
- **design/downloadCad/** - CAD Model Download
    - **View File:** `osdag/web_api/cad_model_download.py` (CADDownload class)
    - **Method:** GET/POST
    - **Purpose:** Downloads generated CAD files

Report Generation Endpoints:

- **api/report/generate-initial/** - Initial Report Generation

  - **View File:** `osdag/web_api/report_customization_api.py` (GenerateInitialReport class)
  - **Method:** POST
  - **Purpose:** Generates initial LaTeX file and returns section list
  - **Parameters:** Output data and company details
  - **Response:** Returns report ID and list of available sections

- **api/report/parse-sections/** - Parse Report Sections
  - **View File:** `osdag/web_api/report_customization_api.py` (ParseReportSections class)
  - **Method:** POST
  - **Purpose:** Parses LaTeX file and extracts section hierarchy
  - **Parameters:** Report ID
  - **Response:** Returns structured section list

- **api/report/customize/** - Custom PDF Generation
  - **View File:** `osdag/web_api/report_customization_api.py` (CustomizeReport class)
  - **Method:** POST
  - **Purpose:** Generates customized PDF with selected sections
  - **Parameters:** Report ID and selected sections list
  - **Response:** Returns compiled PDF file

- **company-logo/** - Company Logo Management
  - **View File:** `osdag/web_api/design_report_csv_view.py` (CompanyLogoView class)
  - **Method:** GET/POST
  - **Purpose:** Handles company logo upload and retrieval for reports

Module and Design Preference Endpoints:

- **modules** - Get Available Modules
  - **View File:** `osdag/web_api/modules_api.py` (GetModules class)
  - **Method:** GET
  - **Purpose:** Returns list of all available design modules
  - **Response:** Returns module metadata (key, name, image, path)
- **design-preferences/** - Design Preferences
  - **View File:** `osdag/web_api/design_pref_api.py` (DesignPreference class)
  - **Method:** GET/POST
  - **Purpose:** Manages design preferences (bolt types, weld types, etc.)
- **materialDetails/** - Material Details
  - **View File:** `osdag/web_api/design_pref_api.py` (MaterialDetails class)
  - **Method:** GET
  - **Purpose:** Returns material properties and details

User Authentication Endpoints:

- **user/login/** - User Login
  - **View File:** `osdag/web_api/user_view.py` (LoginView class)
  - **Method:** POST
  - **Purpose:** Authenticates user with email and password
- **user/signup/** - User Registration
  - **View File:** `osdag/web_api/user_view.py` (SignupView class)
  - **Method:** POST
  - **Purpose:** Registers new user account
- **user/logout/** - User Logout

  - **View File:** `osdag/web_api/user_view.py` (LogoutView class)
  - **Method:** POST
  - **Purpose:** Logs out authenticated user

- **user/forgetpassword/** - Password Recovery
  - **View File:** `osdag/web_api/user_view.py` (ForgetPasswordView class)
  - **Method:** POST
  - **Purpose:** Handles password reset requests

- **user/checkemail/** - Email Verification
  - **View File:** `osdag/web_api/user_view.py` (CheckEmailView class)
  - **Method:** POST
  - **Purpose:** Verifies email address availability

- **googlesso/** - Google SSO
  - **View File:** `osdag/web_api/google_sso_api.py` (GoogleSSOView class)
  - **Method:** GET/POST
  - **Purpose:** Handles Google Single Sign-On authentication

- **user/set-refresh/** - Refresh Token Management
  - **View File:** `osdag/web_api/user_view.py` (SetRefreshTokenCookieView class)
  - **Method:** POST
  - **Purpose:** Sets refresh token in HTTP-only cookie

Project Management Endpoints:

- **api/projects/** - Project CRUD Operations
  - **View File:** `osdag/web_api/project_api.py` (ProjectAPI class)
  - **Method:** GET (list), POST (create)
  - **Purpose:** Lists user projects or creates new project

  - **Response:** Returns project list or created project details

- **api/projects/¡project_id¿/** - Project Details
  - **View File:** `osdag/web_api/project_api.py` (ProjectDetailAPI class)
  - **Method:** GET, PUT, DELETE
  - **Purpose:** Retrieves, updates, or deletes specific project

- **api/projects/by-name/¡project_name¿/** - Project by Name
  - **View File:** `osdag/web_api/project_api.py` (ProjectByNameAPI class)
  - **Method:** GET
  - **Purpose:** Retrieves project by name

- **api/projects/¡project_id¿/osi** - Project OSI Download
  - **View File:** `osdag/web_api/osi_api.py` (ProjectOsiDownload class)
  - **Method:** GET
  - **Purpose:** Downloads OSI file for a project

OSI File Management Endpoints:

- **api/save-osi-from-inputs/** - Save OSI from Inputs
  - **View File:** `osdag/web_api/osi_api.py` (SaveOsiFromInputs class)
  - **Method:** POST
  - **Purpose:** Creates OSI file from current design inputs
  - **Parameters:** Module ID, name, and inputs
  - **Response:** Returns OSI file data or file URL

- **api/open-osi/** - Upload and Parse OSI
  - **View File:** `osdag/web_api/osi_api.py` (OpenOsiUpload class)
  - **Method:** POST
  - **Purpose:** Uploads and parses OSI file

  - **Response:** Returns module ID, name, and parsed inputs

- **api/open-osi/¡osifile_id¿/** - Get OSI by ID
  - **View File:** `osdag/web_api/osi_api.py` (OpenOsiById class)
  - **Method:** GET
  - **Purpose:** Retrieves OSI file by database ID

- **api/module-routes/** - Module Routes
  - **View File:** `osdag/web_api/osi_api.py` (ModuleRoutes class)
  - **Method:** GET
  - **Purpose:** Returns mapping of module keys to frontend routes
  - **Response:** Returns dictionary of module abbreviations to route paths

- **user/saveinput/** - Save Input File
  - **View File:** `osdag/web_api/user_view.py` (SaveInputFileView class)
  - **Method:** POST
  - **Purpose:** Saves input file (OSI) to user's account

- **user/obtain-input-file/** - Get Input File
  - **View File:** `osdag/web_api/user_view.py` (ObtainInputFileView class)
  - **Method:** GET
  - **Purpose:** Retrieves saved input files for user

- **api/save-osi/** - Save OSI (DRF)
  - **View File:** `osdag/web_api/user_view.py` (SaveInputFileView class)
  - **Method:** POST
  - **Purpose:** Alternative endpoint for saving OSI files using Django REST Framework

The typical user workflow follows this sequence:

1. User opens module page → Frontend calls `populate` to get dropdown options

2. User enters inputs and clicks Calculate → Frontend calls `calculate-output/<Module-Key>` to get design results
3. After successful calculation → Frontend calls `design/cad/` to generate 3D models
4. User requests report → Frontend calls `api/report/generate-initial/` then `api/report/custom` to generate PDF
5. User saves project → Frontend calls `api/projects/` to save project data
6. User loads OSI file → Frontend calls `api/open-osi/` to parse and load inputs

View Classes (Django REST Framework)

Views handle HTTP requests and return JSON responses. They use Django REST Framework's `APIView` class pattern:

- **Generic View Pattern:** Most modules use a generic `OutputData` view that:
  1. Extracts `Module` key from request body
  2. Resolves the module adapter using `get_module_api()`
  3. Calls `module_api.generate_output(input_values)`
  4. Returns JSON response with {`data, logs, success`}
- **Dedicated Views:** Some modules (e.g., Simply Supported Beam) have dedicated view classes for custom logic or validation.
- **Error Handling:** Views catch exceptions and return appropriate error responses with status codes (400 for client errors, 201 for success).

Module Resolution System

The module resolution system maps module keys (strings) to their corresponding adapter classes. This is implemented in `osdag_api/module_finder.py`:

- **Module Dictionary:** A dictionary mapping module keys to adapter modules:
  - `'FinPlateConnection'` → `fin_plate_connection`
  - `'Simply-Supported-Beam'` → `simply_supported_beam`

  - 'Cleat-Angle-Connection' → cleat_angle_connection

- **Module API Type:** All adapters implement a common protocol (ModuleApiType) with required methods:
  - get_required_keys() → List of required input keys
  - validate_input() → Input validation
  - create_from_input() → Create engine instance from inputs
  - generate_output() → Generate formatted output
  - create_cad_model() → Generate CAD files
- **Module Registration:** New modules are registered in:
  - osdag_api/module_finder.py → module_dict mapping
  - osdag_api/__init__.py → developed_modules list and module_dict array (for frontend)

Module Adapter Pattern

Each module has an adapter in osdag_api/modules/ that bridges the web API and the calculation engine:

- **Input Mapping:** Adapters normalize frontend input keys to match what the osdag_core engine expects.
- **Engine Integration:** Adapters instantiate the appropriate engine class from osdag_core:
  - Fin Plate → osdag_core.design_type.connection.fin_plate_connection.FinPlateConn
  - Simply Supported Beam → design_type.flexural_member.flexure.Flexure
  - Tension Bolted → design_type.tension_member.tension_bolted.TensionBolted
- **Output Formatting:** Adapters convert engine output (tuples, dictionaries) into a flat JSON structure:

```
{
    "Bolt.Pitch": {
```

```
            "key": "Bolt.Pitch",
            "label": "Pitch Distance (mm)",
            "val": 40
        },
        ...
    }
```

- **CAD Generation:** Adapters handle CAD model creation using `cad.common_logic.CommonDesig` and write BREP/STEP/IGES files.

osdag_core Integration

The `osdag_core` package contains the calculation engines. Adapters interact with these engines through a standardized interface:

- **Engine Instantiation:** `create_module()` or `create_from_input()` creates engine instances.
- **Input Setting:** Engines use `set_input_values(dict)` to receive design parameters.
- **Output Methods:** Engines provide methods like:
  - `output_values(True)` → Design check results
  - `spacing(True)` → Spacing and detailing values
  - `capacities(...)` → Connection capacities (for connections)
  - `section_capacities(...)` → Section-specific capacities
- **Logging:** Engines can set up loggers to capture design warnings and errors.

Request/Response Flow

The complete flow from frontend request to backend response:

1. **Frontend Request:** React component sends POST request to `calculate-output/<Module-Key>` with JSON body containing:

- `"Module": "<Module-Key>"`
- All input fields (e.g., `"Bolt.Diameter": "24", "Load.Shear": "140"`)

2. **URL Routing:** Django routes request to appropriate view class.
3. **View Processing:** View extracts `Module` key and calls `get_module_api(module_name)`.
4. **Module Resolution:** `module_finder` returns the adapter module.
5. **Output Generation:** View calls `module_api.generate_output(input_values)`:
   - Adapter validates inputs
   - Adapter creates engine instance: `module = create_from_input(input_values)`
   - Adapter calls engine methods: `output_values()`, `spacing()`, etc.
   - Adapter formats output into flat JSON structure
   - Adapter collects logs/warnings
6. **Response:** View returns JSON:

```
{
    "data": { ... formatted output ... },
    "logs": [ ... array of log messages ... ],
    "success": true
}
```

2.2.3 Frontend React Architecture

The frontend is built with React and follows a component-based architecture with centralized state management.

Component Structure

The frontend uses a hierarchical component structure:

- **EngineeringModule:** The main shell component that wraps all design modules. Located at `osdagclient/src/modules/shared/components/EngineeringModule.jsx`. It provides:

  - Input dock (left panel) for user inputs
  - Output dock (right panel) for design results
  - 3D CAD viewer (center) using React Three Fiber
  - Header with navigation and controls
  - Modal dialogs for design preferences, reports, etc.

- **Module-Specific Pages:** Each module has its own page component (e.g., `FinPlate.jsx`, `SimplySupportedBeam.jsx`) that:
  - Configures `moduleConfig` with design-specific settings
  - Renders `EngineeringModule` with appropriate props
  - Provides module-specific `OutputDockComponent`

- **Shared Components:** Reusable components in `osdagclient/src/modules/shared/`:
  - `BaseOutputDock.jsx` → Generic output renderer
  - `InputSection.jsx` → Input form sections
  - `CustomizationModal.jsx` → Report customization
  - `DesignReportModal.jsx` → Design report generation

State Management

The frontend uses React Context API for global state management:

- **ModuleContext:** Manages module-specific data and operations. Located at `osdagclient/src/context/ModuleState.jsx`. Provides:
  - Dropdown data (beamList, columnList, materialList, etc.)
  - Design results (output, logs, CAD paths)
  - Core functions: `getModuleData()`, `createDesign()`, `createCADModel()`, etc.

- **UserContext:** Manages user authentication and session data.

- **GlobalContext:** Manages application-wide state (module list, design types).

- **useEngineeringModule Hook:** Custom hook that:

  - Connects to ModuleContext
  - Manages local state (inputs, output, loading)
  - Handles form submission to backend
  - Manages input locking/unlocking
  - Handles project loading/saving

Module Configuration

Each module defines a `moduleConfig` object that describes its behavior:

- **Required Fields:**
  - `designType` → Backend module key (must match `Module` in request)
  - `cameraKey` → Identifier for 3D view options
  - `sessionName` → Session storage key
  - `inputSections` → Array of input form sections
  - `defaultInputs` → Default input values
- **Optional Fields:**
  - `modalConfig` → Modal dialog configurations
  - `cadOptions` → CAD view section names
  - `osiKeyMap` → Mapping from OSI file keys to input keys

API Integration

The frontend communicates with the backend through REST API calls:

- **Base URL:** `http://127.0.0.1:8000/` (configurable)
- **Output Calculation:** POST to `calculate-output/<Module-Key>` with:

```
{
    "Module": "<Module-Key>",
    "Bolt.Diameter": "24",
```

```
    "Load.Shear": "140",
    ...
}
```

- **Input Data Population:** GET to `populate?moduleName=<Module-Key>` returns:
  - Section lists (beams, columns, angles)
  - Material lists
  - Bolt options (diameters, grades, types)
  - Thickness lists
- **CAD Generation:** POST to `design/cad/` with module inputs, returns base64-encoded CAD files per section.
- **Project Management:** REST API endpoints under `api/projects/` for CRUD operations.

Routing System

React Router handles client-side navigation:

- **Module Routes:** Defined in `osdagclient/src/constants/modules.js`:
  - `MODULE_ROUTES` → Maps module keys to route paths
  - `MODULE_SUBMODULES` → Groups modules by category
  - `CONNECTIONS_TAB_CONTENT` → Connection module cards
  - `GENERIC_SUBMODULE_CONTENT` → Other module cards
- **Module Selection Page:** `SelectModulePage.jsx` displays module cards and navigates on click.
- **Module Pages:** Each module has its route (e.g., `/design/connections/shear/fin_plate`) that renders the module page.

2.2.4 Frontend-Backend Communication

The communication between frontend and backend follows a standardized contract.

Request Format

All calculation requests must include:

- **Module Key:** `"Module": "<Module-Key>"` (required)
- **Input Fields:** All fields returned by `get_required_keys()` from the adapter
- **Content-Type:** `application/json`
- **Method:** POST

Response Format

All responses follow this structure:

```
{
    "data": {
        "<key>": {
            "key": "<key>",
            "label": "<Human-readable label>",
            "val": <value>
        },
        ...
    },
    "logs": [
        "<log message 1>",
        "<log message 2>",
        ...
    ],
    "success": true|false
}
```

Error Handling

- **Client Errors (400):** Invalid inputs, missing required fields, module not found
- **Server Errors (500):** Internal calculation errors, engine failures

- **Success (201):** Design completed successfully
- **Logs:** Warnings and errors are included in the `logs` array even on success

Data Flow Example

A complete example for Fin Plate Connection:

1. User fills input form in React component
2. User clicks "Calculate" button
3. `useEngineeringModule` hook calls `handleSubmit()`
4. Hook sends POST to `calculate-output/FinPlateConnection` with inputs
5. Django routes to `OutputData.as_view()`
6. View resolves `fin_plate_connection` adapter
7. Adapter creates `FinPlateConnection` engine instance
8. Engine performs calculations
9. Adapter formats output and logs
10. View returns JSON response
11. Hook updates state with output
12. `EngineeringModule` displays output in Output Dock
13. CAD generation request is sent (if enabled)
14. 3D model is displayed in viewer

2.3 Task 1: Python Code

This section presents Python code examples from the backend architecture, specifically the module adapter pattern and view classes.

2.3.1 Description of the Code

The code examples demonstrate:

- **Module Resolution:** How modules are registered and resolved using a dictionary mapping
- **View Class:** How Django REST Framework views handle HTTP requests and generate responses
- **Module Adapter:** How adapters bridge the web API and calculation engines

2.3.2 Python Code

The module finder code is shown below:

Listing 2.1: Module Resolution System

```
1  from osdag_api.modules import (
2      fin_plate_connection,
3      simply_supported_beam,
4      cleat_angle_connection
5  )
6
7  module_dict = {
8      'FinPlateConnection': fin_plate_connection,
9      'Simply-Supported-Beam': simply_supported_beam,
10     'Cleat-Angle-Connection': cleat_angle_connection,
11 }
12
13 def get_module_api(module_id: str):
14     """Return the api for the specified module"""
15     module = module_dict[module_id]
16     return module
```

2.3.3 Explanation of the Code

- **Line 1-4**: Imports module adapter classes from the modules package.

- **Line 6-10**: Defines the `module_dict` dictionary that maps module keys (strings) to their adapter modules.
- **Line 12-15**: The `get_module_api()` function takes a module ID string and returns the corresponding adapter module.

2.4 Task 1: Documentation

The architecture documentation was contributed to the Osdag Developer Guide. The main documentation file is `NEW_MODULE_GUIDE.md`, which provides comprehensive instructions on:

- How to add new design modules to the system
- Backend adapter pattern and implementation
- Frontend component structure and routing
- Module registration and resolution
- CAD model generation workflow
- Request/response contract specifications

The guide follows the architecture described in this chapter and serves as a reference for developers adding new modules to the Osdag Web application.

Chapter 3

Major UI and Functional Updates

3.1 Task 2: Problem Statement

The Osdag Web interface needed significant updates to match the functionality and user experience of the Desktop version. Additionally, several new features were required to improve usability, visualization, and project management. The main challenges included:

- Web interface behavior did not match the Desktop version
- Lack of interactive 3D visualization features (hover effects, multiple view selection)
- No custom PDF report generation capability
- Limited project management features (no OSI file integration, basic authentication)
- Performance issues with 3D CAD rendering due to FreeCAD dependency
- Inconsistent UI appearance across different pages and modes

3.2 Task 2: Tasks Done

Following the team discussions, the Web Interface was updated to match the behavior of the Desktop version and several new features were implemented.

3.2.1 User Interface Overview

The application provides a consistent user experience across three main pages, each available in both light and dark modes.

- **Homepage:** The landing page where users can access recent projects and navigate to different sections of the application. [Screenshot: Homepage in light mode and dark mode will be added here]
- **Module Selection Page:** A page that displays all available design modules organized by categories (Connections, Tension Members, Flexural Members, etc.). Users can browse and select the module they want to use. [Screenshot: Module Selection Page in light mode and dark mode will be added here]
- **Module Page:** The main design interface with Input Dock (left panel), Output Dock (right panel), and 3D CAD viewer (center). This is where users enter design parameters, view results, and interact with the 3D model. [Screenshot: Module Page with Input/Output docks in light mode and dark mode will be added here]

3.2.2 Header and Project Management

- **Dock Management:** A new Header system was introduced to help manage the Input/Output docks better.
- **About Section:** An "About" dropdown was added in the header so users can see version info and credits.
- **Project Deletion:** Previously, the "Recent Projects" list was static. This component was refactored so users can now delete old projects they don't need anymore.
- **Loading Improvements:** The project loading process was simplified in the main content area, making it feel snappier when switching between designs.

3.2.3 Branding and Appearance

- **Logos:** The FOSSEE logo was removed and Ministry of Education (MoE), Ministry of Steel (MoS), INSDAG, and ConstructSteel logos were added.

- **Dark Mode:** The dark mode styling was updated to be consistent across all pages. The bug where the toggle icon didn't change color was also fixed.

3.2.4 Input Dock Behavior (Lock/Unlock)

- **Auto-Lock:** When a design is completed, the Input Dock locks automatically.
- **Scrollable Lock:** Users can scroll to read locked inputs.
- **Safety:** Unlocking triggers a warning: *"The current designs will be lost."*
- **Reset:** Unlocking clears the CAD and output immediately to prevent data mismatch.

3.2.5 Custom PDF Report Generation

A two-step custom PDF generation process was implemented to allow users to select specific sections for their design reports.

- **Step 1 - Initial Report Generation:** Output data and basic details (company name, project title, designer, etc.) are sent to the backend. A LaTeX file is generated and a list of available sections is returned to the frontend.
- **Step 2 - Customization:** Users can select which sections to include in the final PDF. The filtered section list is sent back to the backend, which generates the customized PDF with only the selected sections.
- **Implementation:** The backend uses LaTeX filtering to include only selected sections, then compiles the filtered LaTeX to PDF using pdflatex.

3.2.6 3D Viewer Hover Feature

A hover feature was implemented in the 3D CAD viewer to provide interactive feedback when users move their mouse over different parts.

- **Hover Dictionary (hover_dict):** The backend provides a dictionary mapping part names to their labels and dimensions. This dictionary is sent along with CAD model data.

- **Visual Feedback:** When hovering over a part (beam, plate, bolt, weld, etc.), the part changes color to highlight it.
- **Dimension Display:** A tooltip appears showing the part name and its dimensions, making it easy to identify and inspect individual components of the connection.
- **Implementation:** The hover dictionary is integrated into the React Three Fiber rendering system, with proper key matching to handle singular/plural variations and case-insensitive lookups.

3.2.7 Multiple View Selection

Users can now select multiple view options simultaneously in the 3D viewer.

- **Combined Views:** Users can choose to view both Beam and Plate at the same time, or any combination of available views (Model, Beam, Column, Plate, Connector, etc.).
- **Flexible Display:** The view selection system allows multiple checkboxes to be active simultaneously, showing all selected parts together in the 3D viewer.
- **View Mapping:** The system maps view names to part names, so selecting "Plate" view shows both the plate and associated bolts, providing a more comprehensive view of the connection.

3.2.8 Spacing Details Modal

The spacing details display was improved with a custom drawing modal.

- **Interactive Modal:** A modal dialog displays spacing parameters (pitch, gauge, edge distance, end distance) alongside a custom-drawn diagram.
- **Custom Drawing:** The modal includes a custom SVG-based drawing that visually represents the bolt pattern, plate dimensions, and spacing measurements.
- **Parameter Display:** All spacing parameters are shown with their values in millimeters, making it easy to verify bolt arrangements and distances.

- **Visual Clarity:** The drawing adapts to light and dark themes, ensuring readability in both modes.

3.2.9 Email Authentication and Project Management

Email-based authentication was implemented with automatic project saving.

- **Email Login:** Users can log in using their email address, which serves as the primary identifier for their account.
- **Automatic Project Saving:** When a user is authenticated and creates or modifies a design, the project is automatically saved to the database.
- **OSI JSON Storage:** Project inputs are stored as OSI JSON in the database, allowing projects to be fully restored with all design parameters.
- **Project Association:** Each project is associated with the user's email, ensuring users can only access their own projects.

3.2.10 OSI File Integration

OSI file loading and downloading functionality was added to improve project portability.

- **OSI Loading from Homepage:** Users can load an OSI file directly from the homepage. The system automatically detects the module type from the OSI file and opens the appropriate module page.
- **Automatic Input Population:** When an OSI file is loaded, the module page is opened with all inputs automatically populated from the OSI file data.
- **OSI Download:** Users can download the current project as an OSI file, which can be shared or used to restore the project later.
- **Module Detection:** The system reads the module identifier from the OSI file and routes to the correct module page, ensuring seamless project restoration.

3.2.11 FreeCAD Dependency Removal

The FreeCAD dependency was removed to improve performance and simplify deployment.

- **Direct STL Usage:** Instead of converting BREP files on the server using FreeCAD (which was slow), STL files are now used directly.
- **Browser-Side Merging:** STL files are merged in the browser using JavaScript, eliminating the need for server-side CAD conversion.
- **Performance Improvement:** This change significantly reduces 3D model loading time and removes the heavy FreeCAD dependency from the server.
- **Simplified Deployment:** The removal of FreeCAD simplifies the deployment process and reduces server resource requirements.

3.3 Task 2: Python Code

This section presents code examples related to the UI and functional updates, specifically the custom PDF generation and OSI file handling.

3.3.1 Description of the Code

The code examples demonstrate:

- **Custom PDF Generation:** Backend API endpoints for generating initial reports and customizing PDFs with selected sections
- **OSI File Handling:** Functions for parsing, saving, and loading OSI files
- **Project Management:** Database models and API endpoints for project management with email authentication

3.3.2 Python Code

The custom PDF generation API code is shown below:

Listing 3.1: Custom PDF Report Generation API

```python
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status

class CustomizeReport(APIView):
    """
    API endpoint to generate customized PDF with selected sections.
    """

    def post(self, request):
        report_id = request.data.get('report_id')
        selected_sections = request.data.get('selected_sections', [])

        # Read original LaTeX content
        tex_file_path = f'file_storage/design_report/{report_id}.tex'
        with open(tex_file_path, 'r', encoding='utf-8') as f:
            original_latex = f.read()

        # Filter content based on selected sections
        filter_obj = LaTeXFilter()
        filtered_latex = filter_obj.filter_content(
            original_latex,
            selected_sections
        )

        # Compile filtered LaTeX to PDF
        pdf_path = compile_latex_to_pdf(filtered_latex)

        return Response({
            'success': True,
            'pdf_path': pdf_path
        }, status=status.HTTP_200_OK)
```

3.3.3 Explanation of the Code

- **Line 1-3**: Imports Django REST Framework classes for API views and responses.

- **Line 5-8**: Defines the `CustomizeReport` APIView class with a POST method handler.
- **Line 10-11**: Extracts the report ID and selected sections from the request data.
- **Line 13-16**: Reads the original LaTeX file that was generated in the first step.
- **Line 18-22**: Filters the LaTeX content to include only the selected sections.
- **Line 24-25**: Compiles the filtered LaTeX to PDF and returns the result.

3.4 Task 2: Documentation

The UI and functional updates were documented in the codebase through:

- **Component Documentation:** React components were documented with JSDoc comments explaining their props and functionality
- **API Documentation:** Backend API endpoints were documented with docstrings explaining request/response formats
- **User Guide Updates:** The user manual was updated to reflect new features like OSI file integration, custom PDF generation, and hover interactions
- **Code Comments:** Critical sections of code were commented to explain the implementation of features like multiple view selection and spacing details modal

The documentation helps new developers understand how the UI features work and how they integrate with the backend systems.

Chapter 4

DevOps and Documentation

4.1 Task 3: Problem Statement

The Osdag Web project lacked comprehensive documentation for new developers and had deployment challenges. The main issues included:

- No clear guide for developers to add new modules to the system
- Missing installation documentation for Windows users
- Complex deployment process without containerization support
- Difficulty for new team members to understand the project structure and contribution workflow

4.2 Task 3: Tasks Done

To help new team members work on the project, better documentation was added and deployment was simplified.

4.2.1 Developer Documentation

Comprehensive documentation was created to guide developers through the process of adding new modules and understanding the codebase.

- **New Module Guide:** A comprehensive guide was written on *"How to create new Shear Connection modules."* This explains exactly where to add files and how to link them in the code. The guide covers:
  - Backend adapter pattern and implementation
  - Frontend component structure and routing
  - Module registration and resolution
  - CAD model generation workflow
  - Request/response contract specifications
- **Windows Manual Guide:** A step-by-step installation guide was written for setting up Osdag on Windows. This includes:
  - Prerequisites and system requirements
  - Python environment setup
  - Database configuration
  - Frontend build process
  - Troubleshooting common issues

4.2.2 Deployment

Containerization support was added to simplify deployment across different environments.

- **Docker Support:** A `Dockerfile` was created to allow the entire application to run inside a container. This makes deployment on any server much easier. The Dockerfile includes:
  - Multi-stage build for optimized image size
  - Python dependencies installation
  - Node.js and React build process
  - Nginx configuration for serving the frontend
  - Environment variable configuration
- **Container Benefits:** The containerized deployment provides:

- Consistent environment across development, staging, and production
- Simplified dependency management
- Easy scaling and deployment
- Isolation from host system

4.3 Task 3: Python Code

This section presents code examples related to deployment and documentation, specifically the Dockerfile configuration.

4.3.1 Description of the Code

The Dockerfile demonstrates:

- **Multi-stage Build:** Separate stages for building the frontend and creating the final production image
- **Dependency Management:** Installation of Python and Node.js dependencies
- **Production Configuration:** Nginx setup for serving static files and proxying API requests

4.3.2 Python Code

The Dockerfile configuration is shown below:

Listing 4.1: Dockerfile for Osdag Web Application

```
1  # Stage 1: Build frontend
2  FROM node:18-alpine AS frontend-builder
3  WORKDIR /app
4  COPY osdagclient/package*.json ./
5  RUN npm install
6  COPY osdagclient/ ./
7  RUN npm run build
8
9  # Stage 2: Python backend
10 FROM python:3.11-slim
```

```
11 WORKDIR /app
12
13 # Install system dependencies
14 RUN apt-get update && apt-get install -y \
15     nginx \
16     && rm -rf /var/lib/apt/lists/*
17
18 # Copy requirements and install Python dependencies
19 COPY requirements.txt .
20 RUN pip install --no-cache-dir -r requirements.txt
21
22 # Copy application code
23 COPY . .
24
25 # Copy built frontend from builder stage
26 COPY --from=frontend-builder /app/build /app/static
27
28 # Configure nginx
29 COPY osdagclient/nginx.conf /etc/nginx/nginx.conf
30
31 # Expose port
32 EXPOSE 80
33
34 # Start services
35 CMD ["sh", "-c", "python manage.py migrate && \
36      python manage.py collectstatic --noinput && \
37      gunicorn osdag_web.wsgi:application --bind 0.0.0.0:8000 & \
38      nginx -g 'daemon off;'"]
```

4.3.3 Explanation of the Code

- **Line 2-7**: First stage builds the React frontend using Node.js, installing dependencies and creating a production build.
- **Line 9-11**: Second stage uses Python slim image as the base for the backend.
- **Line 13-16**: Installs system dependencies including Nginx for serving static files.
- **Line 18-20**: Installs Python dependencies from requirements.txt.

- **Line 22-23**: Copies application code and the built frontend from the builder stage.
- **Line 25-26**: Configures Nginx with custom configuration.
- **Line 28**: Exposes port 80 for HTTP traffic.
- **Line 30-33**: Starts database migrations, collects static files, runs Gunicorn for Django, and starts Nginx.

4.4 Task 3: Documentation

The documentation contributions were made to help developers and users:

- **Developer Guide:** The `NEW_MODULE_GUIDE.md` file provides comprehensive instructions on:
  - Project architecture overview
  - Step-by-step guide for adding new modules
  - Code examples and patterns
  - Testing and validation procedures
  - Common pitfalls and solutions
- **Installation Guide:** The `WINDOWS_MANUAL_INSTALLATION.md` file includes:
  - System requirements and prerequisites
  - Detailed installation steps
  - Configuration instructions
  - Verification and testing procedures
  - Troubleshooting section
- **Code Comments:** Critical sections of code were documented with:
  - Function and class docstrings
  - Inline comments explaining complex logic
  - Type hints for better code understanding

  - Usage examples in docstrings

- **API Documentation:** Backend API endpoints were documented with:
  - Request/response formats
  - Authentication requirements
  - Error codes and messages
  - Usage examples

The documentation follows best practices and helps new team members quickly understand the project structure and contribute effectively.

Chapter 5

Conclusions

5.1 Tasks Accomplished

The following tasks were successfully completed during the internship:

- **Backend Architecture Improvements:**
  - Upgraded Django framework and Python dependencies to ensure security and compatibility
  - Reorganized project directory structure for better clarity and maintainability
  - Integrated `osdag_core` directly into the web repository
  - Implemented modular architecture with clear separation of concerns
  - Established module resolution system for dynamic module loading
  - Created adapter pattern to bridge web API and calculation engines
  - Documented complete request/response flow from frontend to backend
- **Frontend Architecture Development:**
  - Refactored React components for better state management
  - Implemented Context API for global state management
  - Created reusable shared components (EngineeringModule, BaseOutputDock)
  - Established module configuration system for easy module addition
  - Implemented React Router for client-side navigation

  - Integrated REST API communication with backend

- **User Interface Updates:**
  - Updated UI to match Desktop version behavior
  - Implemented consistent light and dark mode across all pages
  - Added new Header system for better dock management
  - Implemented project deletion functionality
  - Improved project loading performance
  - Updated branding with new logos (MoE, MoS, INSDAG, ConstructSteel)
  - Fixed dark mode toggle icon color bug

- **Input/Output Dock Features:**
  - Implemented auto-lock functionality when design is completed
  - Added scrollable lock feature for reading locked inputs
  - Implemented safety warning on unlock
  - Added automatic reset of CAD and output on unlock

- **Custom PDF Report Generation:**
  - Implemented two-step PDF generation process
  - Created initial report generation with LaTeX file creation
  - Implemented section selection and filtering
  - Added PDF compilation with selected sections only

- **3D Visualization Enhancements:**
  - Implemented hover feature with color highlighting
  - Added hover dictionary for part dimensions display
  - Created multiple view selection capability
  - Implemented combined view display (e.g., Beam and Plate together)
  - Removed FreeCAD dependency for better performance

  - Implemented direct STL file usage with browser-side merging

- **Spacing Details Modal:**
  - Created interactive modal with spacing parameters
  - Implemented custom SVG-based drawing for bolt patterns
  - Added parameter display with millimeter values
  - Ensured theme compatibility (light and dark modes)

- **Authentication and Project Management:**
  - Implemented email-based authentication system
  - Added automatic project saving to database
  - Implemented OSI JSON storage for project restoration
  - Created project association with user email

- **OSI File Integration:**
  - Implemented OSI file loading from homepage
  - Added automatic module detection from OSI files
  - Created automatic input population from OSI data
  - Implemented OSI file download functionality

- **Documentation:**
  - Created comprehensive New Module Guide (NEW_MODULE_GUIDE.md)
  - Wrote Windows Manual Installation Guide (WINDOWS_MANUAL_INSTALLATION.md)
  - Documented backend architecture and API flows
  - Documented frontend component structure
  - Added code comments and docstrings throughout the codebase

- **Deployment:**
  - Created Dockerfile for containerized deployment
  - Implemented multi-stage build for optimized image size

  - Configured Nginx for serving frontend and proxying API requests
  - Simplified deployment process across different environments

5.2 Skills Developed

The following technical and professional skills were developed during the fellowship:

5.2.1 Technical Skills

- **Backend Development:**
  - Django framework and Django REST Framework for API development
  - Python programming with object-oriented design patterns
  - Module resolution and adapter pattern implementation
  - Database integration and ORM usage
  - Request/response handling and error management
  - LaTeX file generation and PDF compilation
- **Frontend Development:**
  - React.js for building user interfaces
  - React Hooks (useState, useEffect, useContext, useMemo)
  - React Router for client-side navigation
  - Context API for global state management
  - Component-based architecture design
  - REST API integration and data fetching
  - Responsive design and theme implementation
- **3D Visualization:**
  - React Three Fiber for 3D rendering
  - STL file handling and processing
  - 3D model manipulation and interaction

  - Camera controls and view management
  - Hover effects and interactive features

- **DevOps and Deployment:**
  - Docker containerization
  - Multi-stage Docker builds
  - Nginx configuration
  - Environment variable management
  - Deployment best practices

- **Version Control and Collaboration:**
  - Git for version control
  - Code review processes
  - Collaborative development workflows

- **Documentation:**
  - Technical writing and documentation
  - Code commenting and docstring writing
  - User guide creation
  - Developer guide writing

5.2.2 Professional Skills

- **Problem Solving:**
  - Analyzing complex system architectures
  - Identifying performance bottlenecks
  - Debugging and troubleshooting issues
  - Finding optimal solutions for technical challenges

- **System Design:**

  - Designing modular and scalable architectures
  - Planning API structures and data flows
  - Creating reusable component libraries
  - Implementing design patterns (Adapter, Factory, etc.)

- **Communication:**
  - Technical documentation writing
  - Code review and feedback
  - Explaining complex concepts clearly
  - Collaborating with team members

- **Project Management:**
  - Breaking down large tasks into manageable components
  - Prioritizing features and improvements
  - Meeting deadlines and deliverables
  - Managing multiple features simultaneously

- **Quality Assurance:**
  - Testing functionality across different scenarios
  - Ensuring code quality and maintainability
  - Verifying UI/UX consistency
  - Performance optimization

- **Learning and Adaptation:**
  - Quickly learning new frameworks and technologies
  - Adapting to existing codebase patterns
  - Researching and implementing best practices
  - Continuous improvement mindset

5.3 Impact and Future Work

The work completed during this internship has significantly improved the Osdag Web application:

- The modular architecture makes it easier for new developers to add design modules
- The improved UI/UX provides a better user experience matching the Desktop version
- Performance improvements (FreeCAD removal, STL optimization) make the application faster
- Comprehensive documentation helps onboard new team members quickly
- Containerized deployment simplifies the deployment process
- New features (hover, multiple views, custom PDF) enhance the application's functionality

Future improvements could include:

- Additional design modules implementation
- Further performance optimizations
- Enhanced 3D visualization features
- Mobile responsiveness improvements
- Advanced reporting capabilities
- Integration with external CAD software

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.