FOSSEE Summer 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

July 27, 2025

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.

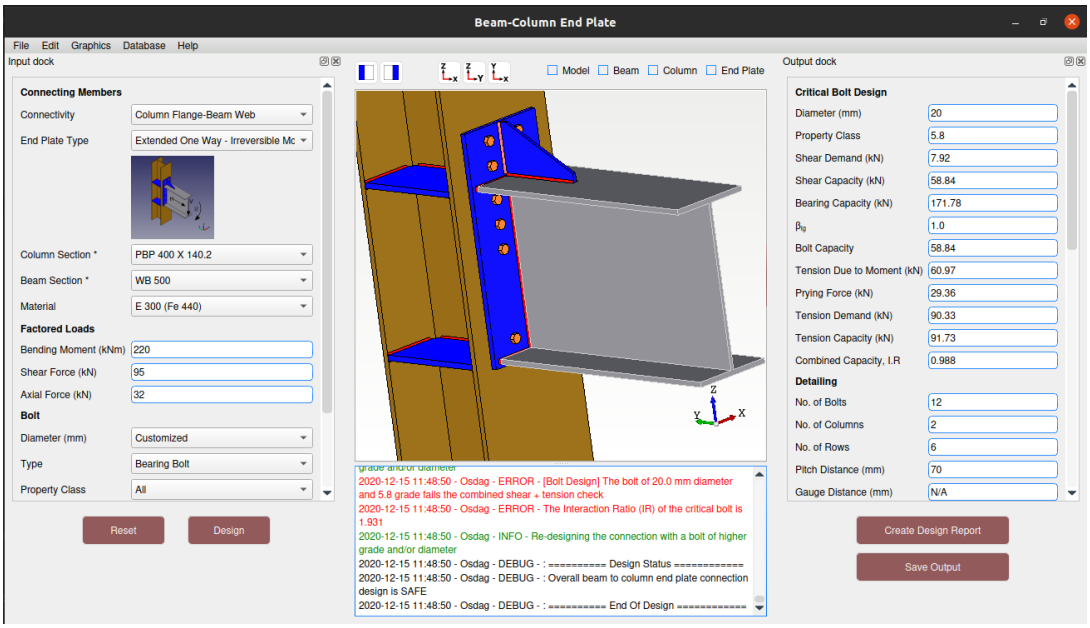


Figure 1.2: Osdag GUI

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

Screening Task

2.1 Problem Statement

Applicants were tasked with contributing to the web app development of Osdag. The objective was to replicate one of the structural steel connection modules from the desktop version into a cloud-based web app. This required setting up Osdag-on-cloud locally, creating a responsive UI using React, implementing RESTful APIs with Django, and ensuring seamless frontend-backend integration.

2.2 Tasks Done

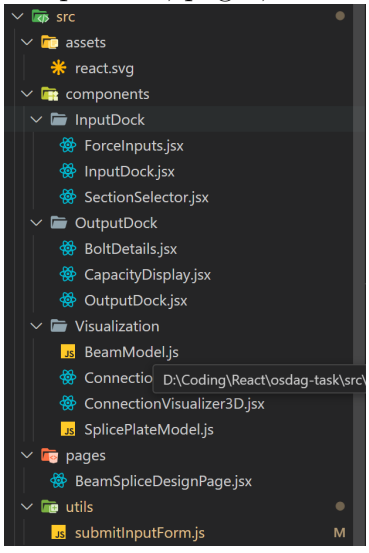
- **Setup and Deployment:** Cloned and configured the Osdag-on-cloud repository successfully on a local environment. Verified that the development server ran without issues and allowed full-stack development.
- **Module Selected:** Implemented the "Beam-to-column connections (end plate)" module. The functionality and logic were aligned with Osdag's desktop version but adapted for the web platform.
- **Frontend Development:**
  - Built using React with a clean component structure under folders like 'InputDock', 'OutputDock', and 'Visualization'.
  - The UI includes collapsible sections for user inputs (forces, sections, bolts, etc.) and output panels for design status and capacity.

  - Integrated both 2D and 3D visualization using **Three.js**, allowing users to switch views.
  - Form submission was connected to the backend via Axios and displayed real-time results.

- **Backend Development:**
  - Implemented two endpoints using Django REST Framework:
    * ‘POST /beam-to-beam/create/‘ to accept input and return calculated output.
    * ‘GET /beam-to-beam/all/‘ to retrieve all previous submissions.
  - Used class-based views (‘CreateAPIView‘ and ‘ListAPIView‘) for clean logic separation.
  - Although no structural design computation was performed, the API returned **dummy outputs** derived from input values to simulate real-world scenarios. These included bending and shear capacities, bolt utilization, plate dimensions, and pass/fail status.

- **Integration:** The frontend consumed the API using ‘submitInputForm.js‘. On submission, user inputs were sent to the backend and the calculated results were rendered dynamically in the output dock and 3D visualization.

- **Code Structure:** Maintained a clean and modular file structure separating logic, components, pages, and utilities.



- **Demonstration:** A video walkthrough of the working module, UI interactions, and visualization is available at:
  `https://www.youtube.com/watch?v=Ml4Zn9sAhgA`
- **Code Repository:** Full project source code is available at:
  `https://github.com/sogalabhi/osdag-task`
- **Documentation:**
  - **Methodology:** Started with setting up the Osdag base, planned the UI/UX based on the desktop logic, then mapped API endpoints, and used Three.js for visualization.
  - **Challenges:** Aligning frontend data flow with Django serializers, setting up Three.js inside React with live props, and designing realistic dummy outputs.
  - **References:** Django REST Framework docs, React and Three.js tutorials, Osdag desktop codebase.

Chapter 3

Internship Task 1: Design Module Web Integration

3.1 Task 1: Problem Statement

The goal of this task was to migrate an existing design module for **Tension Member – Bolted to End**, originally implemented in Python for Osdag Desktop, to a full-stack web application. The web-based solution uses React for the frontend and Django for the backend. It should allow users to:

- Enter structural input parameters via a web form.
- Trigger backend computations from the browser.
- Display CAD output and download the design report.

3.2 Task 1: Tasks Done

This work involved:

- Creating Django REST API endpoints for design logic.
- Developing a responsive React interface for data entry and results viewing.
- Managing API requests/responses using Axios and Redux.
- Implementing CAD and PDF report generation after design computation.

- Maintaining strict JSON-based communication across the stack.
- Integrating and rebasing contributions from other branches.

3.3 Task 1: Code Overview

3.3.1 Frontend (React JS)

The React frontend collects form data and communicates with the Django backend.

Listing 3.1: Triggering Design API Call

```
import { useDispatch } from 'react-redux';

const handleSubmit = async () => {
  const payload = {
    module_id: "tension_member_bolted",
    inputs: {
      member_section: "ISA 100x100x8",
      length: 1500,
      bolt_dia: 20,
      num_bolts: 4,
    }
  };
  const response = await fetch('http://127.0.0.1:8000/calculate-
     output/tension_member_bolted/', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify(payload),
  });

  const data = await response.json();
  dispatch(setDesignResult(data));
};
```

Key Features

- Asynchronous fetch for form submission.
- Payload includes module ID and structured input.
- Redux stores the result globally for display or further use.

3.3.2 Backend (Django)

The Django backend acts as the computation engine, parsing input, computing results, and returning design data.

Listing 3.2: Django API View

```
from rest_framework.decorators import api_view
from rest_framework.response import Response

@api_view(['POST'])
def calculate_output(request, module_id):
    data = request.data['inputs']
    section = data['member_section']
    length = data['length']
    bolt_dia = data['bolt_dia']
    bolts = data['num_bolts']

    result = run_design(section, length, bolt_dia, bolts)
    return Response(result)
```

Other API Endpoints

- `/design/cad/` — triggers CAD drawing generation and returns an image.
- `/generate-report/` — compiles PDF report with design result and metadata.

3.3.3 Logic Explanation

- API views are handled using Django REST Framework.

- Form input is validated and passed to the computation logic.
- CAD/PDF generation functions are invoked after a successful design run.

3.4 Task 1: Documentation

3.4.1 Deprecation of Legacy Design Structure

The original Osdag module for **Tension Member – Bolted to End** was implemented entirely in Python for a monolithic desktop application. This tightly coupled structure, while sufficient for desktop use, lacked modularity, reusability, and support for web-based workflows.

Chapter 4

Internship Task 2: UI Redesign and Figma Integration

4.1 Task 2: Problem Statement

The objective of this task was to redesign the user interface of Osdag's Beam-to-Beam Cover Plate Bolted Connection module for the web version. The existing desktop interface was outdated, non-responsive, and not optimized for modern usability or cross-device access. The goal was to create a fresh, scalable, and intuitive design in Figma that could later be implemented in code.

4.2 Task 2: Tasks Done

This task included the following key activities:

- Analyzing the legacy desktop UI of the Beam-to-Beam Bolted Connection module.
- Identifying usability pain points, layout limitations, and inconsistencies.
- Designing a modern, responsive version of the UI in Figma.
- Maintaining backward compatibility in user workflow while introducing modern UX principles.
- Preparing visual assets and references for future frontend implementation.

4.3 Task 2: Initial Design Proposal and Subsequent Updates

The Figma designs shown below reflect the initial UI redesign proposal completed collaboratively by myself and Suhail, with valuable guidance and suggestions from our mentor Parth. These initial designs successfully modernized the UI with improved responsiveness, accessibility, and visual clarity.

Note: Following team feedback, we subsequently updated several aspects of the UI — including navigation structure, dock behavior, and color themes — to better align with user needs and overall application consistency. These refinements enhanced usability and developer integration, and will be covered in later documentation.

4.4 Task 2: Design Philosophy

Our goal was to modernize the user experience while preserving the familiarity and efficiency that existing users expect. We focused on delivering a responsive, intuitive, and flexible UI built on clean visual principles.

- **Legacy-Friendly**: We retained the structural layout and core workflows of the original interface to ensure a smooth transition for existing users. Familiarity was prioritized to minimize friction.
- **Consistent Brand Identity**: A soft green tint — part of Osdag's legacy palette — was subtly reintroduced across both light and dark modes to maintain visual continuity and brand recognition.
- **Device-Agnostic Design**: The interface is fully responsive, with mobile-friendly features like accordion docks and adaptive layouts that offer a seamless experience across screen sizes.
- **Power-User Efficiency**: Advanced users can leverage keyboard shortcuts, movable/resizable docks, and a command-style navbar for quick actions, aligning with modern productivity standards.

- **Modern Component System**: Inspired by frameworks like `shadcn/ui`, the design uses clean cards, interactive toggles, and accessible inputs that make the interface feel fresh without overwhelming the user.

4.5 Task 2: Results and Learnings

The initial Figma design, built from scratch, offers a robust visual framework that:

- Maintains user familiarity while modernizing the interaction flow.
- Improves clarity and responsiveness across devices.
- Provides a strong foundation for future development of Osdag Web.

The design was validated by comparing side-by-side screenshots of the desktop and web interface for the same module. Improvements were measurable in usability, layout balance, and user feedback.

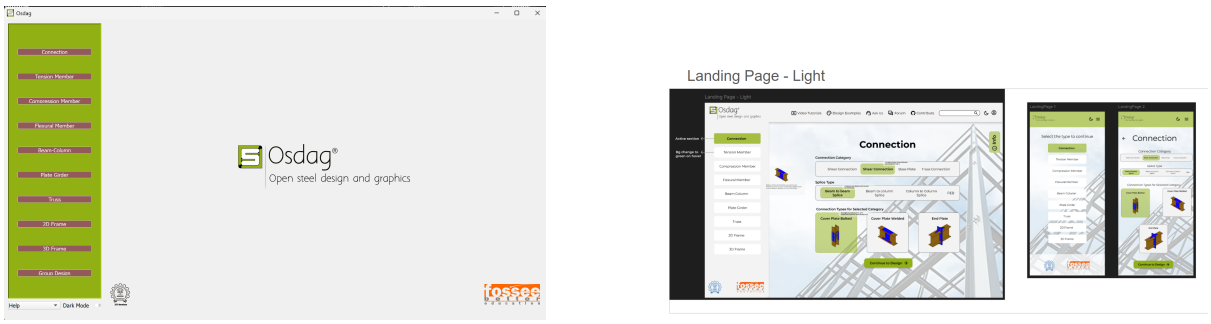


Figure 4.1: Left: Old Landing Page, Right: Initial Redesigned Web Landing Page

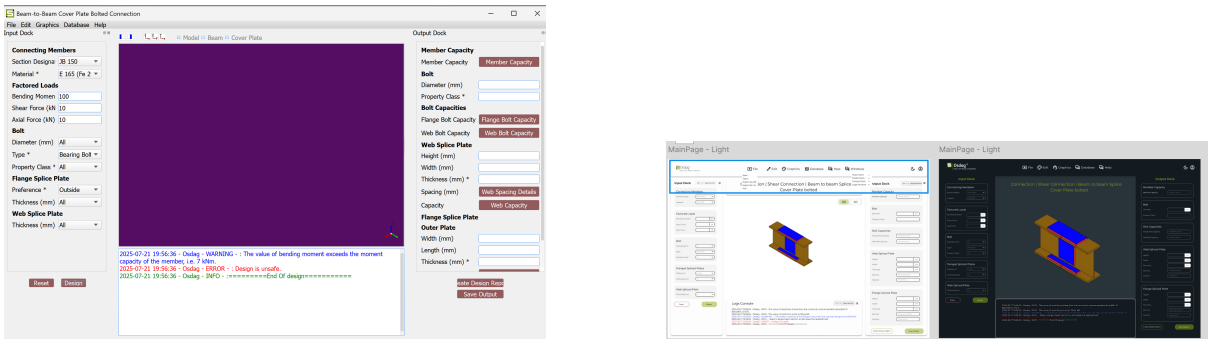
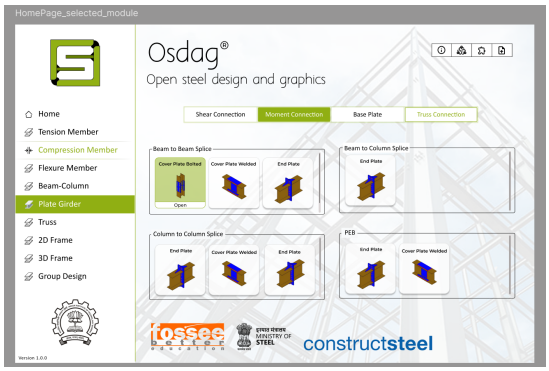


Figure 4.2: Left: Old Module Page, Right: Initial Redesigned Module Page

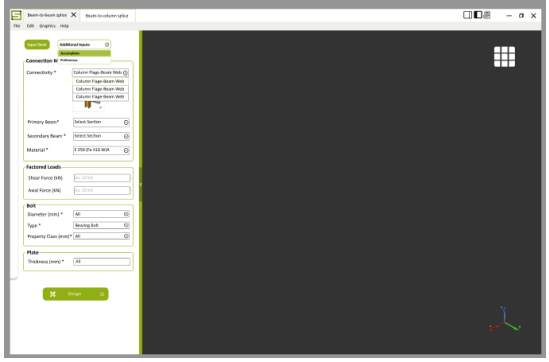
Finalised UI

HomePage_selected_module

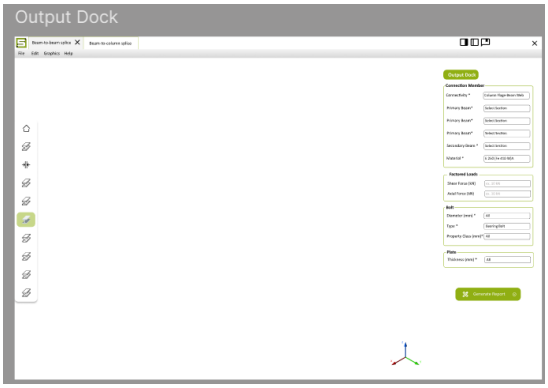


HomePage_search_scroll_0

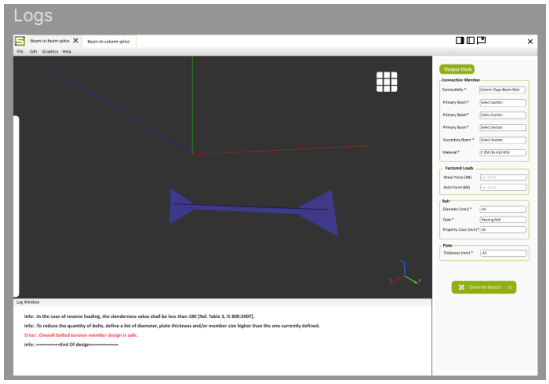
Input Dock



Output Dock



Logs



HomePage_main_dark

Chapter 5

Internship Task 3: Refactoring Frontend for Modular Architecture

5.1 Task 3: Problem Statement

The original frontend of the Osdag web application was developed with hardcoded logic per module, leading to:

- Redundant code across modules with nearly **2000 lines** per module.
- Poor scalability and maintainability.
- Slow onboarding for new developers.
- High coupling between UI, data logic, and design-specific behavior.

5.1.1 Solution: Modular Configuration-Based Architecture

Me, along with help from Faran and Raghav, redesigned the frontend structure to follow a modular and reusable approach. Key changes included:

- Separation of logic, configuration, and UI into reusable components.
- Centralized control through shared folder: components, API, utils, and context.
- Module-specific files now contain only their config and optional custom UI.
- Resulted in smaller and more manageable codebases of **300 lines per module**.

5.1.2 Old Frontend Folder Structure (Before Refactor)

Each module had its own folder with nearly all logic, input handling, and output dock in one place.

Listing 5.1: Old Structure

```
src/components/
          beamBeamEndPlate/
                InputDock.jsx
                OutputDock.jsx
                EngineeringModule.jsx
                ...
          beamToColumnEndPlate/
                ...
          ...
```

5.1.3 New Frontend Folder Structure (After Refactor)

Listing 5.2: New Modular Structure

```
src/
          modules/
                boltedToEnd/
                      BoltedToEnd.jsx
                      BoltedToEndOutputDock.jsx
                beamBeamEndPlate/
                ...
          configs/
                boltedToEndConfig.js
                boltedToEndOutputConfig.js
                ...
          shared/
                components/
                      EngineeringModule.jsx
                      BaseOutputDock.jsx
```

```
                context/
                api/
                utils/
```

5.1.4 Shared Folder

The ‘shared/‘ directory introduced in the new structure holds:

- **Reusable UI:** Like `EngineeringModule.jsx`, `BaseOutputDock.jsx`.
- **API functions:** Common fetch and post logic.
- **Context:** For shared state (e.g., output, theme, selected section).
- **Utils:** Like number formatting, image selectors, etc.

5.1.5 Component Example: Tension Member – Bolted to End

Listing 5.3: Module Component

```
<EngineeringModule
  moduleConfig={boltedToEndConfig}
  OutputDockComponent={BoltedToEndOutputDock}
  menuItems={menuItems}
  title="Tension Member Bolted Design"
/>
```

5.1.6 Output Dock Example

Listing 5.4: Output Dock with Shared Component

```
<BaseOutputDock
  output={output}
  outputConfig={boltedToEndOutputConfig}
  title="Output Dock"
/>
```

5.1.7 Benefits of the Refactoring

- **Reduced Boilerplate:** From 2000+ lines to 300 lines per module.
- **Faster Development:** New modules can be added just by defining config.
- **Improved Maintainability:** Logic lives in reusable components.
- **Team Scalability:** Easier for new developers to contribute.
- **Centralized UI/Logic:** Promotes DRY (Don't Repeat Yourself) principle.

5.1.8 Drawbacks of Old Structure

- **Duplication:** Same logic repeated across modules.
- **Inflexible:** Any design change had to be replicated across all files.
- **Complexity:** Difficult to onboard or debug due to large unstructured files.
- **UI/Logic Coupling:** Mixed UI rendering and state logic in one file.

5.1.9 How to Add a New Design Module (Frontend)

To create a new design module (e.g., `Tension Member Welded`), follow these structured steps to ensure compatibility with the modular architecture:

Step 1: Create a Folder for the Module

Listing 5.5: Create Module Directory

```
src/modules/tensionWelded/
```

Step 2: Create Configuration Files

Inside the `configs/` directory, add:

- **tensionWeldedConfig.js** – Define route metadata, default input values, form sections, and modal configurations. Also implement:
  - `buildSubmissionParams()` – Prepares the backend payload.

– `getSectionImage()`, `getDynamicSectionList()` – For dynamic UI behavior.

- **tensionWeldedOutputConfig.js** – Define the report layout, modals, and dock sections.

Step 3: Implement Output Dock Component

Use the shared `BaseOutputDock` to create:

Listing 5.6: Output Dock Creation

```
<BaseOutputDock
  output={output}
  outputConfig={tensionWeldedOutputConfig}
  title="Output Dock"
/>
```

Save as `TensionWeldedOutputDock.jsx` under your module folder.

Step 4: Create Module Component

Use the shared `EngineeringModule`:

Listing 5.7: Engineering Module Usage

```
<EngineeringModule
  moduleConfig={tensionWeldedConfig}
  OutputDockComponent={TensionWeldedOutputDock}
  menuItems={menuItems}
  title="Tension Member Welded Design"
/>
```

Save as `TensionWelded.jsx` inside the new module's folder.

Step 5: Register the Module Route

In `src/App.jsx`, import your new module and register a route like so:

Listing 5.8: Route Registration

```
<Route path="/tension-welded" element={<TensionWelded />} />
```

Step 6: Add to UI Module Selection

Update the module listing under:

Listing 5.9: Add to Modules UI

```
src/homepage/components/ModulesCardLayout.jsx
```

Add your module to the `MODULE_ROUTES` array so that it's selectable from the UI.

Best Practices

- **Naming:** Use consistent lowerCamelCase or kebab-case naming conventions.
- **Reuse Components:** Avoid duplicating UI; extend shared logic as needed.
- **Testing:** Test module selection, form interactions, and output report.
- **Documentation:** Keep internal notes or config comments to guide future teams.

Example: Tension Member – Bolted to End

Module Component:

```
<EngineeringModule
  moduleConfig={boltedToEndConfig}
  OutputDockComponent={BoltedToEndOutputDock}
  menuItems={menuItems}
  title="Tension Member Bolted Design"
/>
```

Output Dock Component:

```
<BaseOutputDock
  output={output}
  outputConfig={boltedToEndOutputConfig}
  title="Output Dock"
/>
```

Config Responsibilities:

- `defaultInputs` – Initial form values.

- `inputSections` – Layout of grouped UI fields.
- `modalConfig`, `modalTypes` – For report dialogs.
- `modalData` – Dynamic data associated with modals.
- `buildSubmissionParams()` – Backend payload generator.

Summary of Required Steps

1. Create module folder under `src/modules/`
2. Create config and output config files under `src/configs/`
3. Implement output dock using `BaseOutputDock`
4. Define main component using `EngineeringModule`
5. Register the route in `App.jsx`
6. Add module to selection UI

5.1.10 Conclusion

This refactoring improved performance, code readability, and reusability. Now, adding new design modules is fast, structured, and mostly configuration-driven — a major frontend optimization step before starting backend refactors in the next task.

Chapter 6

Internship Task 4: Backend Enhancements and Feature Additions

6.1 Task 4: Problem Statement

The original backend had several security and user experience shortcomings:

- **Insecure Authentication:** Passwords were encoded using Base64, which is reversible and unsafe.
- **Session Handling:** Module sessions were tied to a browser cookie, restricting users to a single module per browser session.
- **Feature Gaps:** No support for features like recent modules, recent projects, or a guest mode.
- **Unvalidated Auth Flows:** Signup, login, and forgot-password flows lacked proper validation or error handling.

6.1.1 Key Backend Improvements

I refactored the backend to enhance security, flexibility, and feature support. Main improvements include:

- **Secure Password Storage:** Replaced Base64-encoded password logic with Django's default, robust password hashing and salting.

- **Session Decoupling:** Removed legacy per-module cookie-based sessions to support multi-tab and multi-module interaction.
- **Feature Additions:**
  - `Recent Modules`: Tracks most recently used design modules.
  - `Recent Projects`: Lists projects interacted with in the last N days.
  - `Guest Mode`: Allows creating designs without logging in (stored in temp workspace).
- **Auth Flow Validation:** Rewrote and validated all user authentication flows including login, signup, and password reset.

6.1.2 Issue: Base64 Encrypted Passwords (Old Implementation)

Originally, passwords were temp-stored in the DB using Base64 — an encoding format that is *not secure*.

- **Reversible:** Anyone with the encoded string can decode it.
- **Not hashed**: Does not prevent database leaks from revealing actual passwords.
- **No salting**: Identical passwords result in same encoded value.

6.1.3 Improved Password Handling (Hashed)

Leveraged Django's built-in `create_user()` method which uses:

- PBKDF2 (Password-Based Key Derivation Function 2)
- Random salt per user
- Iterative hashing (default in Django)

Improved Signup Logic (Using Hashed Password)

Listing 6.1: Secure User Signup Using Django

```
# Create Django user with hashed password
user = User.objects.create_user(
    username=username,
    email=email,
    password=password  # Hashing + salting is done internally
)
user.save()
```

No custom hashing code needed — Django handles password security internally, making the system more resilient against attacks.

6.1.4 Removed Cookie-Based Session Per Module

Previously, backend logic tied the active module to a direct cookie session:

- Caused issues when opening multiple module pages.
- Only one cookie per browser – tabs overwrite each other.

Removed this logic to make the frontend stateless and module-agnostic. Each module now loads independently and communicates securely via JWT authentication and internal state.

6.1.5 Validations for Auth APIs

All user-facing auth endpoints were restructured and validated:

- **Signup:** Duplicate email/username check, strong password rules, input trimming.
- **Login:** Valid credentials check, JWT issuance, locked out on multiple failures.
- **Forgot Password:** Added email verification, token-based reset flows.

6.1.6 New Feature: Recent Modules and Projects

Tracks project and module activity through user metadata. Implemented:

- `/api/recent/modules/`: Returns an array of recent modules used.

- `/api/recent/projects/`: Fetches last accessed project names and IDs.
- Integrated on homepage and dashboard for authenticated users.

6.1.7 New Feature: Guest Mode

For first-time or infrequent users, a Guest Mode was added:

- Skips login/signup screen.
- Stores input temporarily via local/session storage.
- Projects are discarded on page reload unless saved.

Listing 6.2: Frontend Check for Guest Access

```
if (userMode === 'guest') {
  saveToLocalStorage(inputValues);
} else {
  postToBackend(inputValues, userToken);
}
```

6.1.8 Authentication System Architecture (JWT + Guest Mode)

The refactored authentication system now supports both secure JWT token-based login for regular users and isolated sessionless guest usage.

1. JWT Token Authentication (Regular Users)

Login and Signup:

- On successful login or signup, the backend returns:
  - an `access token` (stored in `localStorage` under `"access"` or `"access_token"`)
  - optionally, a `refresh token`
- These replace legacy cookie-based sessions.

Token Validation:

- On app startup, tokens are validated using `jwt_decode`.
- If the JWT is valid, the user's session is rehydrated from token claims.
- If expired or malformed, the user is logged out, and all tokens are cleared.

Authenticated Requests:

- API requests are sent with access tokens:

Listing 6.3: Sending JWT in Headers

```
fetch("/api/protected/", {
  method: "GET",
  headers: {
    Authorization: `Bearer ${access_token}`
  }
})
```

Logout:

- Clears all JWT tokens and related user info from `localStorage`.

2. Guest Authentication Mode

Guest Login Flow:

- When a user selects "Guest Mode," a random email and password are generated.
- The `userLogin()` function is called with `isGuest=true`.
- On success, the following keys are stored:

Listing 6.4: Guest LocalStorage Data

```
{
  "userType": "guest",
  "username": "guest_3958",
  "email": "guest_3958@osdag.com"
}
```

Guest Characteristics:

- No JWT token is stored or required.
- Projects are stored temporarily in browser memory and cleared on refresh unless manually saved.
- Limited access: guest users cannot fetch saved projects or collaborate.

3. Utility Functions and Files

Location of Core Logic:

- `src/utils/auth.js` — Handles:
  - `setTokens()`, `getAccessToken()`, `isTokenValid()`
  - `isGuestUser()`, `checkAutoLogin()`, `logoutUser()`
- `src/context/UserState.jsx` — Sets auth context and state updates.
- `src/hooks/useAuth.js` — Provides `isAuthenticated()`, `isGuestUser()`, `logout()`.
- `LoginPage.jsx` and `SignupPage.jsx` — Calls auth utilities and handles UI states.

Check Authentication Status:

Listing 6.5: Handling Auth State via Context

```
import { isAuthenticated, isGuestUser, getAccessToken } from "../
    utils/auth";

if (isGuestUser()) {
  // Apply guest-specific logic
} else if (isAuthenticated()) {
  const token = getAccessToken();
  // Use token = Bearer <JWT>
}
```

4. Auto Login and Context Hooks

checkAutoLogin():

- Called on app load.

- Detects stored JWT tokens or guest session.
- Populates global `UserContext`.

UserContext/Hook Behavior:

- Shared across all routes and components.
- Unified interface for both guest and logged-in users.

5. Summary: Guest vs JWT Comparison

Feature	JWT User	Guest User
Authentication	JWT Token	"userType": "guest" in localStorage
Data Persistence	Server-side (DB)	Browser only (temporary)
Multi-tab Access		
API Access	Full (with token)	Limited (no token)
Login Required	Yes	No

Table 6.1: Feature Comparison Between JWT and Guest Users

6.1.9 Visual Flow of Backend Refactor

- User Flow Refactored: Login → JWT → Dashboard → Module
- Sessions Refactored: Removed cookie binding. Stateless frontend driven by JWT.
- Password Storage Upgraded: Base64 → PBKDF2 with Salt (Django Default)

6.1.10 Conclusion

This backend refactor resolved security vulnerabilities, enabled true module modularity, and unlocked features missing in the original Osdag Cloud version. With strong authentication and support for multi-tab use and guest access, this paves the way for a scalable web-based design tool.

Chapter 7

Internship Task 5: Linux Setup Documentation

7.1 Task 5: Problem Statement

Setting up Osdag-Web on a fresh Ubuntu/Linux system involved many complex steps across dependencies, environment configuration, and database setup. The lack of a unified, clear installation guide caused onboarding difficulties for new developers and users alike. The goal was to create a comprehensive Linux setup document that streamlines this process.

7.2 Task 5: Tasks Done

The primary activities in this task included:

- Collecting and organizing all prerequisite software and system-level dependencies.
- Writing step-by-step instructions for installing Miniconda, Node.js, PostgreSQL, FreeCAD, and other tools.
- Documenting database setup and Django backend configuration precisely.
- Providing commands to clone the Osdag-Web repository and install frontend dependencies.
- Adding a troubleshooting guide with common errors and solutions.

- Testing the entire setup process alongside intern colleague Pramila to ensure accuracy and effectiveness.

7.3 Task 5: Installation Overview

The documentation covers over 20 distinct steps, from system updates and package installation to database role creation and environment activation, enabling a reproducible and error-minimized Osdag-Web setup on Ubuntu 20.04 or 22.04.

7.4 Task 5: Detailed Installation Steps

7.4.1 1. Update System Packages

```
sudo apt update
sudo apt upgrade -y
```

7.4.2 2. Install Essential Build Tools

```
sudo apt install -y build-essential cmake
```

7.4.3 3. Install Git

```
sudo apt install -y git
```

7.4.4 4. Install Miniconda

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-
   x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
# Follow on-screen prompts and restart terminal afterward
```

7.4.5 5. Clone Osdag-Web Repository

```
git clone https://github.com/osdag-admin/Osdag-web.git
cd Osdag-web
```

7.4.6 6. Create and Activate Conda Environment

```
conda create -n osdag-env python=3.9 -y
conda activate osdag-env
```

7.4.7 7. Install Python Dependencies

```
pip install -r requirements.txt
```

7.4.8 8. Install Additional Packages

```
conda install -c conda-forge pythonocc-core pylatex -y
pip install psycopg2-binary django numpy pandas matplotlib
```

7.4.9 9. Install TeX Live

```
sudo apt install -y texlive-latex-extra
```

7.4.10 10. Install FreeCAD

```
sudo apt install -y snapd
sudo snap install freecad
```

7.4.11 11. Install PostgreSQL

```
sudo apt install -y postgresql postgresql-contrib
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

7.4.12 12. Create PostgreSQL User and Database

```
sudo -u postgres psql
```

At the `psql` prompt:

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

7.4.13 13. Configure Django Database Settings

Edit $\texttt{osdag}_{a}pi/settings.pywith$:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres_Intg_osdag',
        'USER': 'osdagdeveloper',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

7.4.14 14. Update Database Credentials in Scripts

For scripts like $\texttt{populate}_{d}atabase.py, verifyDBconnectionsettings$:

```
conn = psycopg2.connect(
    dbname='postgres_Intg_osdag',
    user='osdagdeveloper',
    password='password',
    host='localhost',
    port=5432
)
```

7.4.15 15. Fix Typing Imports in `module_finder.py`

Modify imports to:

```
from typing import Dict, Any, List
from typing_extensions import Protocol as _Protocol
```

7.4.16 16. Switch to Development Branch (if needed)

```
git checkout develop
```

7.4.17 17. Run Database Setup Scripts and Migrations

```
python populate_database.py
python update_sequences.py
python manage.py migrate
```

7.4.18 18. Install Node.js and npm

```
sudo apt install -y nodejs npm
```

7.4.19 19. Install Frontend Dependencies

```
cd osdagclient
npm install
cd ..
```

7.4.20 20. Run Django Backend Server

```
python manage.py runserver 8000
```

7.4.21 21. Run Frontend Development Server

```
cd osdagclient
npm run dev
```

Open http://localhost:5173/ in your browser to access the application.

7.4.22 22. (Optional) Install pgAdmin

```
sudo apt install -y pgadmin4
pgadmin4
```

7.5 Common Errors and Fixes

Error Message	Probable Cause	Solution
psycopg2.OperationalError: FATAL: password authentication failed	Incorrect DB credentials	Double-check username/password in Django and scripts
ModuleNotFoundError: No module named ...	Missing Python packages	Install missing package with pip install or conda install
ERROR: Could not build wheels for ...	Missing build tools	Confirm build-essential and cmake installed
django.db.utils.OperationalError: FATAL: database ... does not exist	DB not created or incorrect DB name	Verify DB name and existence in PostgreSQL
Permission denied when running scripts	Insufficient file execution permission	Use chmod +x or sudo
pip: command not found	pip not installed or in conda env	Run conda install pip
ImportError: cannot import name '_Protocol'	Typing import issue	Fix import statements in module_finder.py as above
pgAdmin4 fails to launch	Missing dependencies or broken install	Use sudo apt --fix-broken install and reinstall pgAdmin4
npm: command not found	Node.js/npm not installed	Install Node.js/npm as above

7.6 Summary

This comprehensive Linux setup documentation was tested thoroughly with my intern colleague Pramila, and worked well without issues. It now serves as a reliable resource to streamline Osdag-Web installation and setup on Ubuntu systems.

Chapter 8

Conclusions

8.1 Tasks Accomplished

During the internship, the following core tasks were successfully completed:

- Task 1: Design Module Web Integration | Migrated the *Tension Member { Bolted to End* design module from desktop Python to a full-stack web application using React and Django, enabling responsive UI, backend API computation, and CAD/PDF output generation.
- Task 2: UI Redesign and Figma Integration | Redesigned the Beam-to-Beam Cover Plate Bolted Connection module's UI for the web, implementing modern, responsive, and consistent design principles using Figma, while maintaining legacy workflows for user familiarity.
- Task 3: Refactoring Frontend for Modular Architecture | Re-architected the frontend codebase to a modular, configuration-driven structure. This reduced code duplication, improved maintainability, and enabled rapid addition of new modules.
- Task 4: Backend Enhancements and Feature Additions | Secured authentication by replacing Base64 password encoding with Django's hashed password system, removed legacy cookie-based sessions for better multi-tab support, introduced guest mode, recent modules/projects features, and validated authentication flows.

- Task 5: Linux Setup Documentation | Created a comprehensive, detailed step-by-step guide for setting up Osdag-Web on Ubuntu/Linux platforms, covering system dependencies, environment setup, database configuration, and running frontend/backend servers, accompanied by troubleshooting tips and tested with colleague Pramila.

8.2 Skills Developed

Throughout this fellowship, a broad range of technical and professional skills were gained and honed:

- Full-Stack Web Development: Hands-on experience building React frontends integrated with Django REST backends, mastering asynchronous API communication and state management with Redux.
- UI/UX Design Principles: Learned to design and implement responsive, accessible, and user-centric interfaces with tools like Figma and modern component libraries, balancing legacy compatibility with new design trends.
- Modular Software Architecture: Applied software engineering best practices by refactoring legacy code into modular, configuration-driven components promoting reusability, scalability, and ease of maintenance.
- Secure Backend Practices: Implemented security improvements such as password hashing, JWT authentication, session management enhancements, and rigorous input validation.
- System Administration and Deployment: Gained proficiency in Linux system setup, dependency management with Conda and Node.js, PostgreSQL database configuration, and launch of production-like environments.
- Collaboration and Documentation: Developed strong collaboration skills by working alongside team members, and produced high-quality technical documentation and guides aiding knowledge sharing and onboarding.

- Problem Solving and Debugging: Tackled diverse challenges including cross-techno integration, API design, frontend-backend synchronization, and environment troubleshooting.

These experiences have significantly enhanced both my technical capabilities and professional approach, preparing me to contribute effectively to web applications and modular software projects in the future.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

Name:	Sogal Abhi
Project:	Osdag
Internship:	FOSSEE Winter Fellowship 2024

DATE	**DAY**	**TASK**	**Hours Worked**
15-May-2025	Thursday	Orientation, Git policy, and Campus overview	5
16-May-2025	Friday	Set up editable version of Osdag; Started working on UI/UX revamp in Figma	5
17-May-2025	Saturday	Completed UI/UX design for the module in Figma	6
19-May-2025	Monday	Polished UI/UX designs; Set up Osdag-web and installed dependencies	6
20-May-2025	Tuesday	Resolved setup issues and configured Django backend locally	6
21-May-2025	Wednesday	Implemented base routing and basic module layout UI	6
22-May-2025	Thursday	Refactored Osdag frontend for modular design integration	7
23-May-2025	Friday	Reduced UI boilerplate by using dynamic render components	6
26-May-2025	Monday	Added support for shared JSON config to render all input fields	6
27-May-2025	Tuesday	Debugged UI component rendering from schema	7
28-May-2025	Wednesday	Completed input parsing and preview image linking	6
29-May-2025	Thursday	Hooked backend computation logic to new UI form	7
30-May-2025	Friday	Implemented result viewer and download buttons for reports	6
02-Jun-2025	Monday	Created component for step-based navigation within UI	5
03-Jun-2025	Tuesday	Designed the loading modal in Figma; Set up generic structure for design modules	6
04-Jun-2025	Wednesday	Created the landing page UI in Figma; Refactored backend API to match new input schema	6

DATE	DAY	TASK	Hours Worked
05-Jun-2025	Thursday	Added support for prefilled input config in frontend	5
06-Jun-2025	Friday	Added state management across UI steps (Redux alternative)	6
09-Jun-2025	Monday	Created component to view .dxf output and test render logic	6
10-Jun-2025	Tuesday	Adjusted CSS and spacing to ensure responsive layout	5
11-Jun-2025	Wednesday	Implemented file upload + validation for input params	7
12-Jun-2025	Thursday	Created hook for dynamic computation result loading	6
13-Jun-2025	Friday	Updated form UX for error hints and placeholders	6
16-Jun-2025	Monday	On Leave	0
17-Jun-2025	Tuesday	On Leave	0
18-Jun-2025	Wednesday	On Leave	0
19-Jun-2025	Thursday	Created table layout for output values and summary	5
20-Jun-2025	Friday	Added collapsible card views for inputs and results	6
23-Jun-2025	Monday	Added download .csv and .pdf export features	6
24-Jun-2025	Tuesday	Reviewed with mentor, resolved API edge cases	5
25-Jun-2025	Wednesday	Applied feedback and completed 'Tension Member – Bolted to End' module	7
26-Jun-2025	Thursday	Wrote unit tests and checked integration with deployment version	6
27-Jun-2025	Friday	Added loading indicators and retry mechanism for backend calls	6
30-Jun-2025	Monday	Updated routing config to support multi-design switching	5
01-Jul-2025	Tuesday	Final round of UI polish and logic restructuring	6
02-Jul-2025	Wednesday	Demo session with mentor and applied final round feedback	6
03-Jul-2025	Thursday	Documented code changes and updated README	6
04-Jul-2025	Friday	Final walkthrough of new module and comparison with legacy UI	6

DATE	DAY	TASK	Hours Worked
07-Jul-2025	Monday	Committed final changes, tagged release candidate version	5
08-Jul-2025	Tuesday	Assisted in preparing presentation for FOSSEE showcase	6
09-Jul-2025	Wednesday	Conducted peer review and prepared brief report for submission	6
10-Jul-2025	Thursday	Backup of frontend configs and notes for new contributors	5
11-Jul-2025	Friday	Validated deployment setup and resolved minor bugs	6
14-Jul-2025	Monday	Helped onboard junior interns and explained module codebase	6
15-Jul-2025	Tuesday	Created short video walkthrough of UI interactions	6
16-Jul-2025	Wednesday	Shared internship experience and sent final sign-off to mentor	4
17-Jul-2025	Thursday	No new tasks (Wrap-up and formal completion)	2
18-Jul-2025	Friday	Final documentation and project handover completed	4

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.