FOSSEE - ICFOSS Internship Report

on

Development of Osdag Installers and Osdag on Cloud

Submitted by

Nandagopal VS

Under the guidance of

Prof Siddhartha Ghosh
Jitendra K and Meena J Mehta Chair Professor

and under the mentorship of

Ajmal Babu M S
Research Scholar

Department of Civil Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

October 6, 2024

Acknowledgement

I would like to express my sincere gratitude to the FOSSEE and ICFOSS team for providing me with the opportunity to undertake this internship. Their commitment to fostering learning and innovation created an enriching environment that greatly enhanced my experience.

I am especially thankful to Ajmal Babu, my mentor and project research associate in the Osdag team. His guidance and support were invaluable throughout my internship. His depth of knowledge and encouraging approach inspired me to navigate challenges effectively and grow as a developer. I truly appreciate the time he dedicated to mentoring me and sharing his insights, which have significantly contributed to my professional development.

I would also like to extend my gratitude to Professor Siddhartha Ghosh for his leadership and support of the Osdag project. His vision and commitment to advancing educational resources have made this internship possible, and I am grateful for the opportunity to contribute to such impactful work.

Additionally, I want to thank all the Osdag team for their collaboration and assistance.

Contents

Chapter 1

Introduction

1.1 FOSSEE

FOSSEE (Free/Libre and Open Source Software for Education) is an initiative under the National Mission on Education through Information and Communication Technology (NMEICT) by the Ministry of Education, Government of India. It promotes the use of open-source software in education and research, offering a cost-effective alternative to proprietary software. FOSSEE focuses on creating and supporting software solutions, conducting workshops, and developing resources to encourage the adoption of open-source tools in various academic and professional domains. The initiative also provides opportunities for students and professionals to contribute to the open-source ecosystem through internships and collaborative projects.

1.2 ICFOSS

ICFOSS (International Centre for Free and Open Source Software) is an autonomous organization established by the Government of Kerala, India, with the mission to promote the use and development of Free and Open Source Software (FOSS). It serves as a hub for innovation, research, and skill development in the FOSS ecosystem. ICFOSS works on various initiatives, including software development, capacity building, and policy advocacy to foster the use of FOSS in education, governance, and industries. The organization also collaborates with global FOSS communities and provides opportunities for students, professionals, and researchers to contribute to open-source projects.

1.3 OSDAG

Osdag is Free/Libre and Open-Source Software being developed for the design of steel structures following IS 800:2007 and other relevant design codes. OSDAG helps users in designing steel connections, members and systems using interactive Graphical User Interface (GUI). The source code is written in Python, 3D CAD images are developed using PythonOCC. GitHub is used to ensure smooth workflow between different modules and team members. It is in a path where people from around the world would be able to contribute to its development. FOSSEE's "Share alike" policy would improve the standard of the software when the source code is further modified based on the industrial and educational needs across the country. Design and Detailing Checklist (DDCL) for

different connections, members and structure designs is one of the main products of this project. It would create a repository and design guidebook for steel construction based on Indian Standard codes and best industry practices.

1.4 FOSSEE - ICFOSS Internship

The FOSSEE-ICFOSS Internship is a collaborative initiative between FOSSEE (Free/Libre and Open Source Software for Education) and ICFOSS (International Centre for Free and Open Source Software). This internship offers students a unique opportunity to work closely with the FOSSEE team for a period of 1.5 to 2 months. The program is designed to be a remote internship, allowing students to contribute to various open-source projects from anywhere while receiving guidance and mentorship from the experienced FOSSEE team. It focuses on fostering innovation, technical skill development, and contributions to the open-source community, providing hands-on experience in real-world projects.

1.5 Motivations

The FOSSEE-ICFOSS Internship provides a unique opportunity for students to contribute to impactful open-source projects while gaining hands-on experience in cutting-edge technologies. This internship is designed to help students develop their technical skills, supported by guidance and mentorship from the experienced FOSSEE team. Participants can work remotely, making the program accessible, and gain valuable knowledge in software development, documentation, and open-source advocacy. The experience not only enhances their skill set but also builds a strong foundation for future career opportunities in the tech industry.

1.6 Objectives

The primary objectives of the FOSSEE-ICFOSS Internship are to provide students with hands-on experience in open-source software development and promote the use of free and open-source tools. The internship aims to foster innovation by encouraging students to contribute to ongoing projects, enhance their problem-solving abilities, and work collaboratively with a team. Additionally, it seeks to improve students' technical skills, develop comprehensive documentation, and raise awareness about the benefits of open-source software in academia and industry.

Chapter 2

Contributions

2.1 Windows Installer Development

2.1.1 Objective

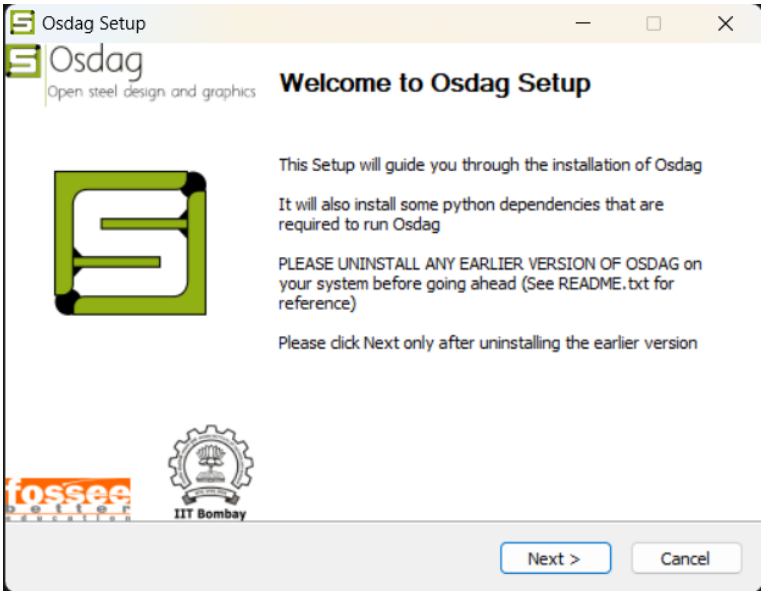
The primary objective was to update the Osdag Windows launcher to ensure compatibility with Windows 11. This involved resolving issues with the installation process for key dependencies like Miniconda, PyQt5, and LaTeX, and ensuring a smooth installation experience for users.

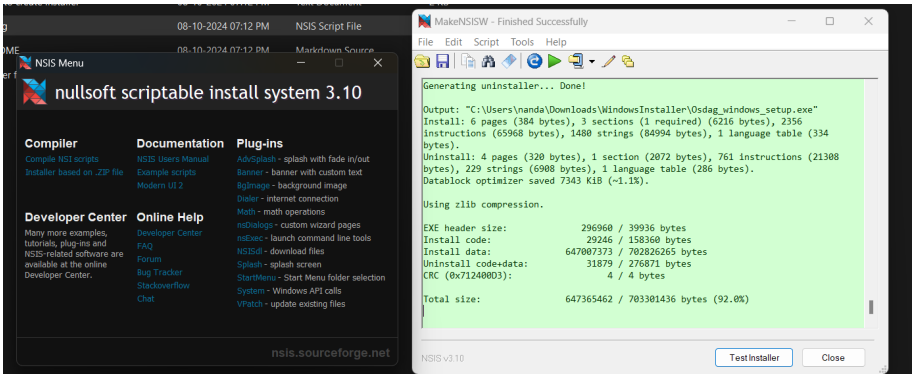
2.1.2 Tasks Completed

- Fixed NSIS Script for Miniconda Installation: Resolved an issue where the NSIS script was not properly installing Miniconda during setup.
- Fixed PyQt5 Installation Issue: Addressed and fixed compatibility problems related to PyQt5 installation on Windows 11.
- Created and Tested Windows Installer Executable: Built the installer executable (.exe) and thoroughly tested it on Windows 11 systems to confirm successful installations without errors..

2.1.3 Outcome

The updated Osdag Windows installer successfully supported Windows 11, with all major dependencies (Miniconda, PyQt5, LaTeX) correctly installed. The final Windows installer executable was fully functional and tested to provide a seamless installation experience across Windows 11 systems.





2.2 Ubuntu Installer Development

2.2.1 Objective

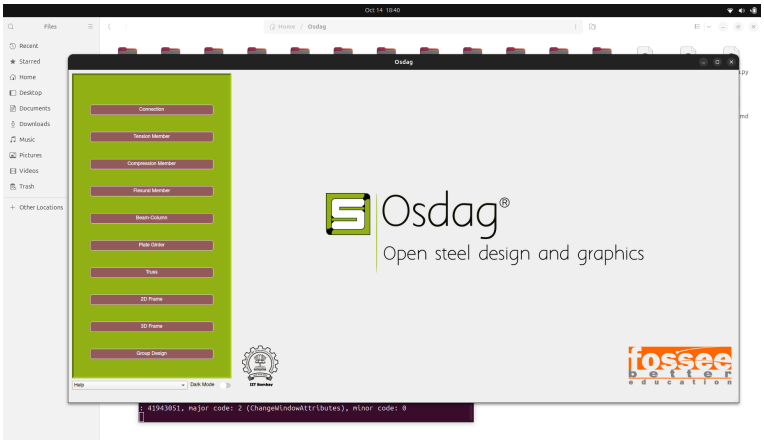
To update the OsdagInstaller to support Ubuntu 24.04. This involved adding compatibility for the latest Ubuntu version and resolving dependency issues that prevented successful installations.

2.2.2 Tasks Completed

- Added Support for Ubuntu 24.04: Updated the OsdagInstaller to ensure compatibility with Ubuntu 24.04 by adapting the script for the latest system changes.
- Resolved Dependency Issues: Fixed dependency conflicts, ensuring all required packages were installed without errors.
- Tested the Installer: Conducted comprehensive testing of the updated installer on Ubuntu 24.04, ensuring smooth installation and functionality.

2.2.3 Outcome

The updated OsdagInstaller now fully supports Ubuntu 24.04, with all dependency issues resolved. The installer was thoroughly tested and confirmed to work smoothly on the latest Ubuntu version



2.3 Osdag on Cloud Development

2.3.1 Objective

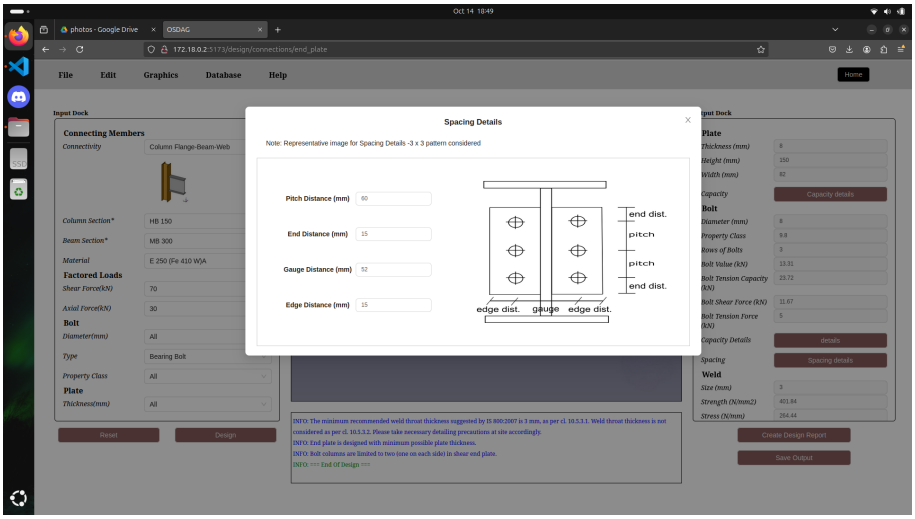
To create the End Plate module for the Osdag on Cloud platform, which involved developing the necessary UI components, writing a new backend endpoint, and integrating the module with existing backend logic.

2.3.2 Tasks Completed

- Created New UI Components for End Plate Module: Developed new UI components in React, ensuring they were responsive and user-friendly.
- Wrote New Backend Endpoint for End Plate Module: Developed a new backend endpoint for the End Plate module, while reusing the underlying logical functions from the Fin Plate module to avoid redundant code.
- Integrated Frontend with Backend APIs: Successfully connected the new UI components with the backend APIs using the newly developed endpoint.
- Fixed Cookies Issue: Resolved a frontend issue related to improper cookie handling, which was causing errors when switching between different modules.

2.3.3 Outcome

The End Plate module was successfully implemented with newly developed UI components, while the existing backend functionality was efficiently adapted for it. The cookies handling issue was fixed, resulting in smooth navigation between modules.



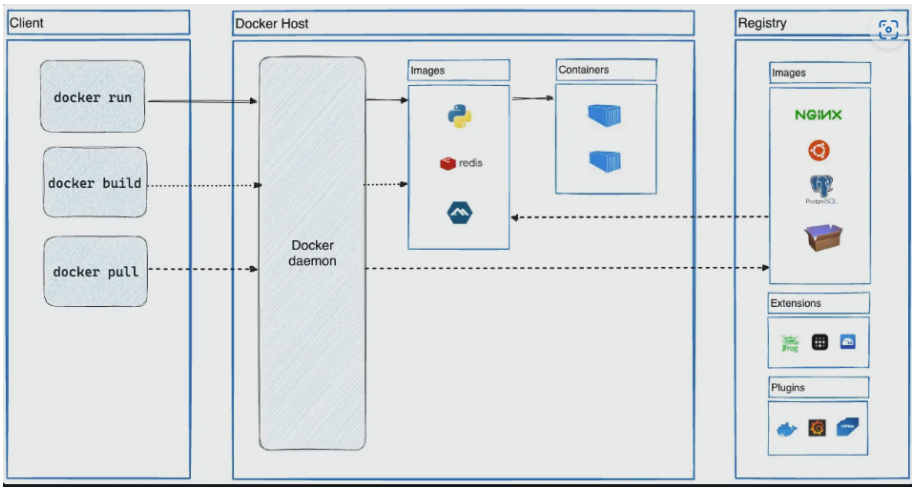
2.4 Docker Setup for Osdag on Cloud

2.4.1 Objective

The objective was to simplify and streamline the setup process for the Osdag on Cloud project by creating a Dockerized environment. This was necessary due to the significant time and challenges faced when manually setting up the project, particularly dealing with various dependency issues.

2.4.2 What is Docker?

Docker is an open-source platform that enables developers to automate the deployment, scaling, and management of applications inside lightweight, portable containers. Containers bundle an application along with its dependencies, libraries, and configuration files, ensuring that it can run consistently across different environments without the typical issues related to compatibility or conflicting dependencies.

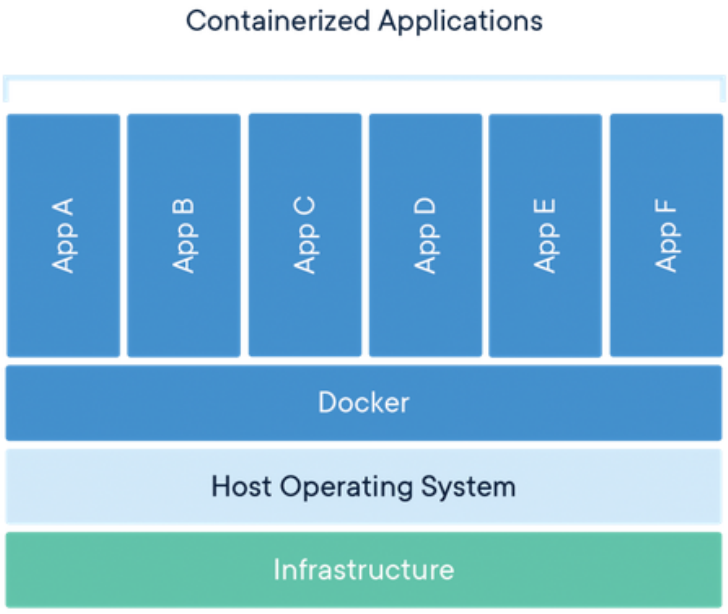


2.4.3 How Docker Works?

Docker containers are built on top of a lightweight virtualization layer, which allows them to share the host operating system's kernel, unlike traditional virtual machines that require separate OS instances. This makes Docker containers much faster to start, use fewer resources, and offer greater flexibility.

Docker uses the following key components:

- Dockerfile: A text file that defines the environment and the steps needed to build a Docker image. It includes instructions like what base image to use (e.g., Ubuntu), what dependencies to install, and how to configure the application.
- Docker Image: A snapshot of a container that contains the application and its dependencies. An image is immutable and can be reused to create multiple containers.
- Docker Container: A running instance of a Docker image. Containers are isolated from each other, ensuring that one application's environment does not interfere with another's.
- Docker Compose: A tool used to define and manage multi-container Docker applications. By using a docker-compose.yml file, developers can easily set up and run multiple containers with a single command.



2.4.4 Why use Docker?

Docker is ideal for developers who need to work with multiple applications or services, each requiring its own specific environment. It solves the "it works on my machine"

problem by providing a consistent environment for all stages of development, from local machines to production servers. It is widely adopted in software development because it reduces setup time, increases the reliability of deployments, and allows for better resource management.

By using Docker, development teams can focus on coding without worrying about setup issues or dependency conflicts, ensuring a smoother development process and more reliable software delivery.

2.4.5 Tasks completed

- Created Dockerfile and Docker Compose Configuration: Wrote a Dockerfile to create a reproducible development environment that included all necessary dependencies and configuration settings. Created a `docker-compose.yml` file to manage the various containers (frontend, backend, and database).

- Managed Dependency Issues via Docker: Packaged all necessary dependencies within Docker containers to isolate the environment and eliminate dependency conflicts. This included specific versions of libraries, database systems, and other external dependencies required for Osdag on Cloud.

2.4.6 Docker Setup

The Docker setup for the Osdag on Cloud project consists of three main containers: one for the backend, one for the frontend, and a database container based on PostgreSQL, all managed through Docker Compose. The backend Dockerfile is located in the root directory. The frontend Dockerfile is located in the osdagclient directory, sets up the React frontend environment.

Backend Docker Configuration

The Dockerfile in the root directory sets up an Ubuntu-based container for running the backend of the Osdag on Cloud project, which is a Django-based web application. The configuration includes necessary dependencies, Python packages, and environment settings for running the backend services smoothly.

Frontend Docker Configuration

The Dockerfile in the osdagclient directory configures a container for the React-based frontend of the project, using a Node.js 18-alpine image. It installs necessary dependencies and runs the Vite development server.

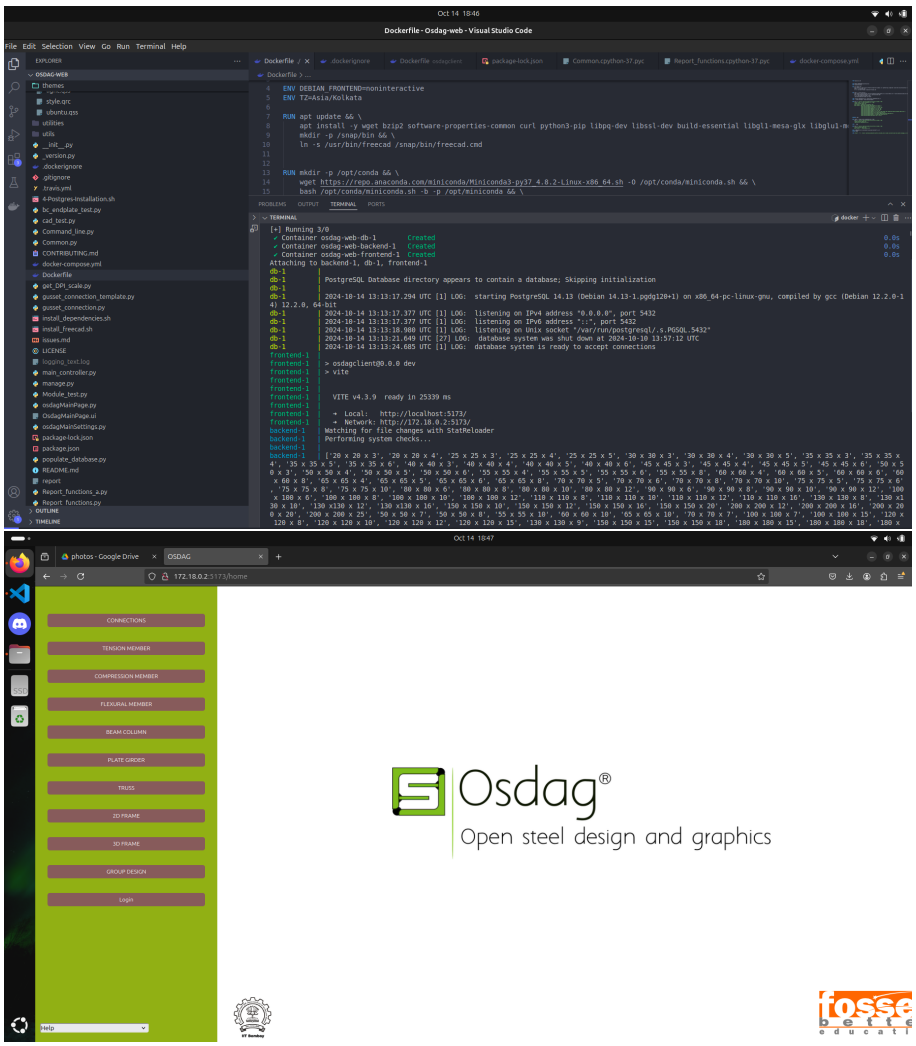
Docker Compose Configuration

The Docker Compose file defines three services: backend, frontend, and db. The backend service builds from a Dockerfile in the root directory, while the frontend service builds from a Dockerfile in the `osdagclient` directory, running the Vite development server. Both services use mounted volumes for real-time file synchronization. The db service uses a PostgreSQL 14 image and uses a named volume for persistent database storage.

2.4.7 Outcome

The Docker setup for the Osdag on Cloud project made the development and deployment processes much easier

- Simplified Local Setup: Developers can now easily set up Osdag on Cloud on their local devices without worrying about installing all dependencies, configuring the database, or managing other setup details. This greatly reduces setup time and complexity.
- Consistent Development Environment: By using Docker, all developers work in the same setup, which help avoid issues where code works on one computer but not on another.
- Easy Deployment: Docker Compose allows us to start all parts of the project with a single command, making it quick to deploy.
- Isolation: Each part of the project runs in its own container, reducing conflicts and making the system easier to manage and secure.
- Future Scalability: The container setup will enable Osdag on Cloud to scale quickly in the future. If the project grows or requires more resources, it can be easily adjusted without significant changes to the infrastructure.



2.5 Contribution to Osdag Developer Manual

During my internship, I contributed to the Osdag Developer Manual by providing comprehensive documentation to assist users and developers in understanding the project. My contributions included the following key areas:

- Installation Instructions: I developed comprehensive instructions for installing Osdag on both Windows and Ubuntu operating systems. This section offers step-by-step guidance to ensure users can successfully install the application without encountering common issues.
- Windows Installer Creation: I documented the process for creating a Windows installer for Osdag, detailing the necessary tools and scripts involved
- Installer Scripts for Linux: I provided documentation on the installer scripts utilized for the Linux version of Osdag. This section outlines the purpose of each script, how they function, and their significance in the overall installation process.
- Osdag on Cloud Documentation: I contributed to the documentation for the Osdag on Cloud project, outlining its structure and core components. This section aims to give developers a clear understanding of the project's organization and the key functionalities of each component.
- Docker Setup Instructions: I created a detailed guide on setting up Osdag on Cloud using Docker on a local device. This guide includes instructions for configuring the environment, building containers, and running the application, which simplifies the process for developers to get started without dealing with complex setups.

2.5.1 Outcome

The contributions made to the Osdag Developer Manual have significantly improved the usability and accessibility of the project for both new users and developers. By providing clear and detailed instructions, the documentation aims to reduce the learning curve associated with installing and working with Osdag.

Chapter 3

Technical Details of the Work

3.1 Technologies and Tools Used

The following technologies were employed in the development of Osdag on Cloud:

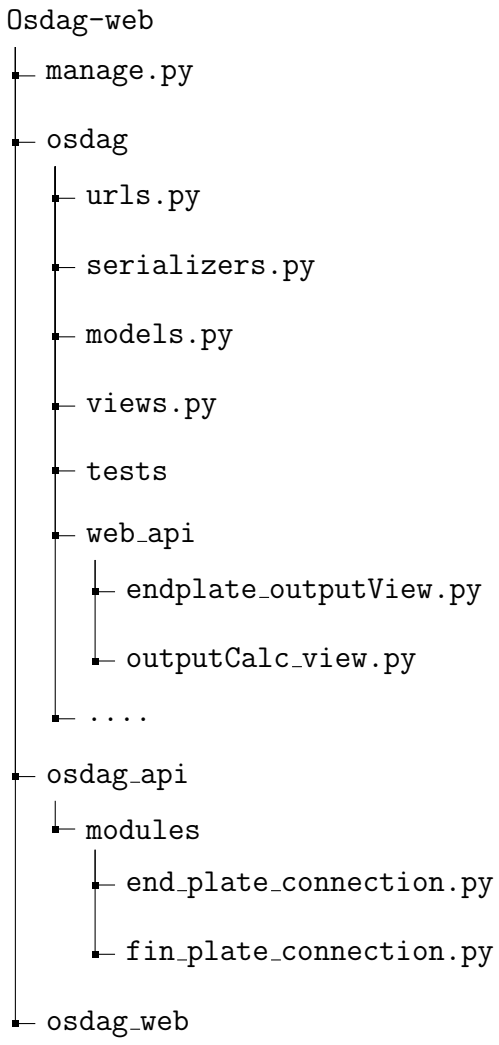
- Python for backend development.
- React and Vite for the frontend.
- PostgreSQL as the database.
- Docker for containerization.
- Git for version control and collaboration.

3.2 Osdag Cloud Architecture

Osdag on cloud mainly includes 3 components a backend , frontend and a db. Docker was used to manage these services.

3.2.1 Backend

The following section provides an overview of the API structure and the implementation of core functionalities.



1. models.py
   This file defines models for the Django application, each model maps to a single database table.These models are intended to manage and store structural design data, user account information, and material properties.

2. urls.py
   This file defines the URL routing for the Osdag Web API, mapping various endpoints to corresponding views.

   - `sessions/create/`: Creates a new session (`CreateSession` view).
   - `sessions/delete/`: Deletes an existing session (`DeleteSession` view).
   - `calculate-output/Fin-Plate-Connection`: Calculates output for Fin Plate Connection (`OutputData` view).
   - `calculate-output/End-Plate-Connection`: Calculates output for End Plate Connection (`EndPLateOutputData` view).

3. serializers.py

   This file defines a set of serializers for various models used in the Osdag Web API.Serializers allow complex data such as querysets and model instances to be

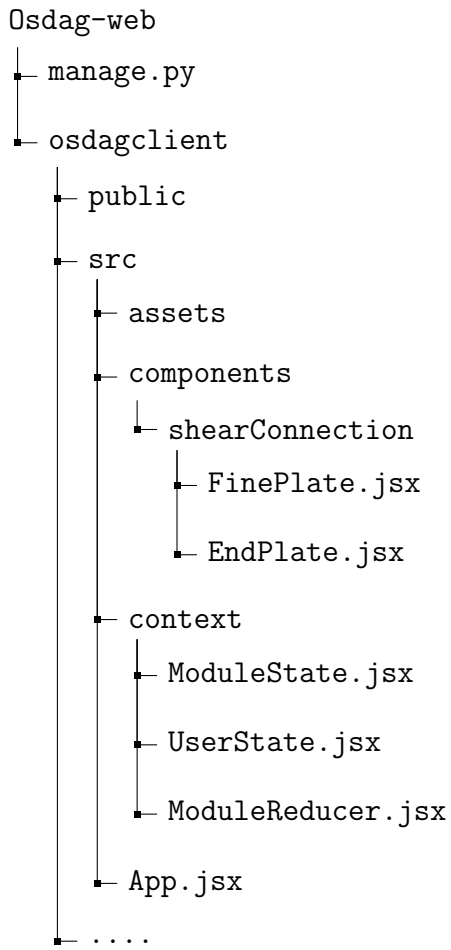
converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

4. fin_plate_connection.py
   This script is designed to validate input data, create a module for fin plate connection design, generate output values in a formatted manner, and generate a CAD model as a BREP file.

   - `create_module()`: Creates and returns an instance of the FinPlateConnection class.
   - `validate_input()`: Validates the input values against the required types. If the validation fails, appropriate custom errors are raised.
   - `generate_output()`: Generates and returns the output values from the module in a formatted JSON structure. The output includes details like pitch distance, bolt capacity, and logs generated during processing.

5. end_plate_connection.py
   Similar to fin_plate_connection.py but for End Plate Connection design.

3.2.2 Frontend



1. App.jsx
   This React file is the main entry point for the web application that manages user authentication, routing, and layout rendering

2. FinePlate.jsx
   The FinePlate.jsx component is responsible for rendering the user interface for the FinPlate module. It accepts user input, communicates with the backend API, and displays the resulting output to the user.

3. EndPlate.jsx
   Component for rendering the user interface for EndPlate module.

4. ModuleState.jsx
   This component is a React context provider that manages the state and provides various functionalities for interacting with the backend API. It handles state management for the developed modules, including connectivity lists, material details, and session management.

5. UserState.jsx
   This component is a React context provider designed to handle user authentication, authorization, and state management related to user login, signup, and file operations.

3.3 Osdag Cloud Docker Setup

3.3.1 Backend Dockerfile

This Dockerfile sets up the backend container for osdag cloud. It uses an Ubuntu 22.04 base image and -

1. Installs necessary system packages, including FreeCAD and LaTeX.

2. Downloads and installs Miniconda .

3. Copies Conda packages and the installer script into the container.

4. Sets up a new Conda environment with Python 3.7.6, installs python packages using both pip and Conda .

5. Copies the project files and installs additional dependencies from the requirements.txt file.

6. Exposes port 8000 and starts the Django development server.

```
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Asia/Kolkata

RUN apt update && \
    apt install -y wget bzip2 software-properties-common curl python3-pip
    libpq-dev libssl-dev build-essential libgl1-mesa-glx libglu1-mesa freecad
    texlive-full && \
    mkdir -p /snap/bin && \
    ln -s /usr/bin/freecad /snap/bin/freecad.cmd

RUN mkdir -p /opt/conda && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.8.2-Linux-
    x86_64.sh -O /opt/conda/miniconda.sh && \
    bash /opt/conda/miniconda.sh -b -p /opt/miniconda && \
    rm -f /opt/conda/miniconda.sh && \
    /opt/miniconda/bin/conda init bash

COPY install_dependencies.sh /app/install_dependencies.sh
COPY ./conda_packages/ /app/conda_packages/

RUN bash -c "source /opt/miniconda/etc/profile.d/conda.sh && \
    conda create -n myenv python=3.7.6 -y && \
    conda activate myenv && \
    pip install --upgrade pip pyopenssl && \
    pip install /app/conda_packages/pdflatex-0.1.3.tar.gz \
                /app/conda_packages/PyLaTeX-1.3.1.tar.gz \
                /app/conda_packages/XlsxWriter-1.2.8.tar.gz \
                /app/conda_packages/Pygments-2.6.1.tar.gz \
                /app/conda_packages/openpyxl-3.0.3.tar.gz \
                /app/conda_packages/PyYAML-5.3.1.tar.gz \
                /app/conda_packages/PyQt5-5.14.2-5.14.2-cp35.cp36.cp37.cp38-
    abi3-manylinux2014_x86_64.whl \
                /app/conda_packages/pdfkit-0.6.1-py3-none-any.whl \
                /app/conda_packages/pandas-1.0.5-cp37-cp37m-manylinux1_x86_64
    .whl \
                /app/conda_packages/pynput-1.6.8-py2.py3-none-any.whl \
                /app/conda_packages/PyGithub-1.54.1.tar.gz"

WORKDIR /app

RUN chmod +x /app/install_dependencies.sh && \
    bash -c "source /opt/miniconda/etc/profile.d/conda.sh && \
    conda activate myenv && \
    /app/install_dependencies.sh"

COPY . /app

RUN bash -c "source /opt/miniconda/etc/profile.d/conda.sh && \
```

```
    conda activate myenv && \
    pip install -r requirements.txt"

ENV LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libstdc++.so.6
EXPOSE 8000

CMD ["bash", "-c", "source /opt/miniconda/etc/profile.d/conda.sh && conda
    activate myenv && python manage.py runserver 0.0.0.0:8000"]
```

3.3.2 Frontend Dockerfile

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5173

CMD ["npm", "run", "dev"]


```

This Dockerfile sets up a container for the frontend of Osdag cloud project using the node:18-alpine image and.

1. Sets the working directory to /app.
2. Copies the package.json and package-lock.json files and installs the necessary dependencies using npm install.
3. Copies the remaining project files into the container.
4. Exposes port 5173 for the Vite development server.
5. Runs the server with the command npm run dev

3.3.3 Docker Compose

Docker Compose is a tool for defining and running multi-container applications. It does the following:

1. Backend: Builds the backend image using a Dockerfile located in the root directory, mounts the current directory to /app.
2. Frontend: Builds the React frontend using a Dockerfile located in the osdagclient directory, mounts the osdagclient directory to /app.
3. Database (db): Uses the official PostgreSQL 14 image, with user credentials and a specified database, and stores data in a named volume (postgres_data).

```
version: '3'

services:
  backend:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    depends_on:
      - db

  frontend:
    build:
      context: ./osdagclient
      dockerfile: Dockerfile
    ports:
      - "5173:5173"
    volumes:
      - ./osdagclient:/app
    environment:
      - CHOKIDAR_USEPOLLING=true
    command: npm run dev


  db:
    image: postgres:14
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydb
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

```

```
volumes:
  postgres_data:
```

3.4 Setting up Osdag Cloud

1. **Clone or Download the Repository**
   - Clone or download this repository to your local machine.
2. **Download the dependencies**
   - Download conda_packages folder and copy it to the `Osdag web` folder.
3. **Build the image**
   - Build the image using

     ```
     docker compose build
     ```

4. **Npm install**
   - Start the frontend container by

     ```
     docker compose run frontend sh
     ```

   - Install the necessary dependencies using

     ```
     npm i
     ```

5. **Setting up backend**
   - Run the containers using

     ```
     docker compose up
     ```

   - Open the bash shell inside the backend container using

     ```
     docker exec -it  osdag-web-backend-1 /bin/bash
     ```

   - Execute the following commands

     ```
     source /opt/miniconda/etc/profile.d/conda.sh
     conda activate myenv
     python populate_database.py
     python update_sequences.py
     ```

To rebuild the image after making any changes use `docker compose build`. To start all the containers use `docker compose up`

3.5 Windows Installer

NSIS (Nullsoft Scriptable Install System) is a professional open-source system to create Windows installers. It is designed to be small, flexible, and capable of creating complex installation scripts. Follow these steps to create a Windows installer for Osdag:

1. Clone or Download the Repository
    - Clone or download this repository to your local machine.
2. Download Miniconda3
    - Download `Miniconda3-py37_23.1.0-1` and copy it to the `Files` folder.
    - Rename the file to `Miniconda3-latest-Windows-x86_64.exe`.
3. Download Python Dependencies
    - Download all required Python dependencies as listed in the `install_osdag_dependencies.bat` file located in the `dependencies` folder.
    - Copy the downloaded dependencies into the `dependencies` folder.
4. Download MiKTeX
    - Download `miktex-x64.exe` and move it into the `Files/latex` folder.
5. Create Executables for Installers
    - Use `pyinstaller` to create executables of latex_package_installer.py and miktek_installer.py.
    - Move the created `.exe` files (they will be created in the `dist` folder) into the `latex` folder.
6. Copy Osdag Folder
    - Copy the `Osdag` folder into the `Files` folder.
7. Install NSIS Software
    - Install the NSIS software from the official NSIS website.
    - Move the header files (environment files) to the `include` folder inside the NSIS installed directory.
8. Compile the Installer Script
    - Compile the `osdag.nsi` file using NSIS.
    - The installer will be created in the same location as the `osdag.nsi` file.

3.6 Ubuntu Installer

The Ubuntu installer mainly includes 3 scripts

`Miniconda3-latest-Linux-x86_64.sh`
It is a shell script used to install Miniconda on Linux systems. Miniconda is a minimal installer for Conda, a package manager that simplifies the process of managing and deploying applications, environments, and packages. It is particularly useful for setting up Python environments and dependencies.

`2-install-osdag.sh`
The script begins by removing any existing Osdag directory to ensure a clean installation. The script then creates a configuration file (`Osdag.config`) with default workspace and installation directory paths. The Osdag files are copied to the user's home directory. The script then creates and activate a Conda environment (`osdagenv`) with Python 3.7.6. Various necessary dependencies are then installed using both `apt-get` and `conda`, as well as `pip` for Python packages.

`3-install-texlive.sh`
This script is designed to install and configure TeX Live on Ubuntu which is a comprehensive TeX distribution used for typesetting documents

Chapter 4

Reflections and Learnings

Throughout my internship, I had the opportunity to work with various technologies and tools that enhanced my technical skills and understanding of software development processes. Here are the key areas I focused on during my tasks:

- NSIS (Nullsoft Scriptable Install System): I learned how to create Windows installers using NSIS, which involved writing scripts to automate the installation process. This experience taught me how to create user-friendly installers and how to troubleshoot common issues that arise during installation.
- Docker: Setting up Docker for the Osdag on Cloud project was a great learning experience. I gained hands-on experience in creating Dockerfiles for both backend and frontend containers, as well as using Docker Compose to manage multi-container applications.
- React: While working on the Osdag on Cloud project, I developed UI components using React and integrated them with backend APIs. This helped me to understand more about front-end development and the importance of creating responsive and interactive user interfaces.
- Python and Django: My involvement in developing endpoints for the backend of the Osdag project enhanced my Python programming skills, particularly in the context of Django.
- Version Control with Git: Throughout my internship, I used Git for version control, which helped me understand how to track changes, collaborate with other developers, and manage code effectively.
- Problem-Solving Skills: Throughout the internship, I encountered various challenges, such as dependency issues and installation failures. Overcoming these obstacles helped me develop strong problem-solving skills and the ability to troubleshoot effectively.

Chapter 5

Conclusion

In summary, my internship experience with the *Osdag team* was highly rewarding. This internship provided me with valuable experiences that broadened my technical expertise and improved my problem-solving abilities. The skills I acquired in software installation, containerization, front-end and back-end development, documentation, and version control will undoubtedly benefit my future endeavors in the field of software engineering.

Appendix A

Appendices

A.1 Repository Links

- Windows Installer Repository
- Ubuntu Installer Repository
- osdag-web Repository
- Windows Installer
- Ubuntu Installer