



# Summer Fellowship Report

On

**Osdag on Cloud**

Submitted by

**Suraj Ranajit Bhosale**

Under the guidance of

**Prof. Sidhartha Ghosh**

Department of Civil Engineering  
IIT Bombay

and under the mentorship of

**Nagesh Karmali**

Project Manager, IIT Bombay

**Mr. Danish Ansari**

Project Software Engineer, IIT Bombay

August 29, 2023

# Acknowledgment

I would like to express my heartfelt gratitude to the FOSSEE team for providing a remarkable platform that allowed me to enhance my skills and contribute to projects in collaboration with IIT Bombay. This opportunity has been invaluable in my journey of growth and learning.

I extend my sincere thanks to every individual who played a pivotal role in guiding and mentoring me throughout this journey. Your insights and support have been instrumental in shaping my understanding and capabilities.

I am deeply appreciative of my fellow friends and team members. Together, we formed a cohesive unit and had the privilege of working alongside dedicated mentors. Their unwavering guidance and mentorship empowered us to navigate through various challenges and learn immensely from each experience.

I would like to extend a special thanks to Nagesh Karmali, who served as a Project Research Associate. From the selection process to the culmination of the fellowship, his continuous guidance proved to be invaluable. I am also grateful to Danish Ansari, the Assistant Project Manager, who provided assistance and insights at every step of the project.

I am honored to have had the opportunity to work under the guidance of Prof. Siddharth Ghosh, who generously shared his expertise and time to mentor us. His support has been a driving force behind our progress.

In conclusion, I extend my heartfelt thanks to FOSSEE and the OSDAG Community for guiding me on this transformative journey. Working with such kind-hearted individuals has been a privilege, and I am grateful for the support and encouragement that I have received. Thank You.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Osdag on Cloud Internship . . . . .	4
1.2	What is Osdag on Cloud? . . . . .	4
1.3	What Technology We Used to Develop . . . . .	4
<b>2</b>	<b>Project Involvement and Contributions</b>	<b>6</b>
2.0.1	My Contributions . . . . .	6
<b>3</b>	<b>Version of Development</b>	<b>8</b>
3.0.1	Package.json . . . . .	8
3.0.2	Dependencies and Their Uses . . . . .	8
<b>4</b>	<b>Flow Tree and Structure</b>	<b>11</b>
<b>5</b>	<b>Design Preferences UI</b>	<b>14</b>
5.1	Working . . . . .	16
<b>6</b>	<b>Authentication</b>	<b>24</b>
6.1	UI and Functionality . . . . .	24
6.1.1	Signup . . . . .	24
6.1.2	Login . . . . .	25
6.1.3	Forget Password: OTP Verification . . . . .	26
6.1.4	Forget Password: Add New Password . . . . .	27
6.2	API's and Contexts . . . . .	27
6.2.1	API's . . . . .	27
6.3	Working in Code . . . . .	29
6.3.1	Important Imports . . . . .	29
6.3.2	SignUp . . . . .	30
6.3.3	Login . . . . .	32
6.3.4	Guest User . . . . .	35
<b>7</b>	<b>Websites for UI Integration</b>	<b>36</b>
7.1	React - Official Documentation . . . . .	36
7.2	React Router . . . . .	36
7.3	Material-UI . . . . .	37
7.4	Ant Design . . . . .	37
7.5	React Bootstrap . . . . .	38

7.6	Semantic UI React . . . . .	38
7.7	React Query . . . . .	39
7.8	Chakra UI . . . . .	39
7.9	Evergreen . . . . .	40
7.10	React Spring . . . . .	40
7.11	Storybook . . . . .	41
7.12	React Icons . . . . .	41
7.13	React Virtualized . . . . .	42
<b>8</b>	<b>Important Documents</b>	<b>43</b>

# Chapter 1

## Introduction

### 1.1 Osdag on Cloud Internship

The FOSSEE Summer Fellowship is provided under the FOSSEE project. FOSSEE project promotes the use of FOSS (Free/Libre and Open Source Software) tools to improve quality of education in our country. FOSSEE encourages the use of FOSS tools through various activities to ensure availability of competent free software equivalent to commercial (paid) softwares.

The FOSSEE project is a part of the National Mission on Education through Infrastructure and Communication Technology (ICT), Ministry of Human Resources and Development, Government of India. Osdag on Cloud is one such open source software which comes under the FOSSEE project. Osdag on Cloud internship is provided through FOSSEE project. And the selection was based on a screening task followed by a task demonstration-interview.

### 1.2 What is Osdag on Cloud?

Osdag on Cloud is a web version of the Osdag Desktop application which is Free/Libre and Open Source Software being developed for design of steel structures. The project uses ReactJS in the frontend, Django in the backend, Postgres and SQLite as a database and FreeCAD software for generating the CAD models. 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.

### 1.3 What Technology We Used to Develop

Osdag was developed to cater to both educational purposes and industry professionals. Its primary goal is to provide valuable insights into the subject matter for students, while also serving as a useful tool for professionals. The user interface

has been meticulously designed to offer flexibility and attractiveness. Additionally, video tutorials are available to facilitate users in getting started with the application.

The technologies employed in the development of Osdag include:

- **Front-End with React.js:** React.js, a prominent JavaScript library, was utilized to create a responsive and dynamic front-end for the application. This ensures an engaging user experience and seamless interaction.
- **Back-End with Django:** Django, a powerful Python web framework, forms the backbone of the application's back-end. It manages user authentication, database models, and API endpoints, ensuring efficient data handling.
- **Database with PostgreSQL:** PostgreSQL, a robust open-source relational database management system, was chosen to store and manage various aspects of the application, including user data and content.
- **React Libraries:** Apart from the core React.js library, additional React libraries such as React Quill and React Router were integrated to enhance the application's capabilities. These provide rich text editing and client-side routing functionalities.
- **Styling Libraries:** Libraries like Material-UI and Ant Design were utilized to leverage pre-styled components, contributing to a polished and user-friendly interface.

The integration of these technologies ensures that Osdag delivers a comprehensive and efficient solution, meeting the requirements of both educational and professional users.

# Chapter 2

## Project Involvement and Contributions

### 2.0.1 My Contributions

- **Main Window and Launch Screen Window:**

Developed the user interface for the main window and launch screen window of the project.

- **Sidebar Development:**

Created and designed the sidebar for easy navigation and access to various sections.

Implemented the functionality of the sidebar to ensure smooth interaction.

- **Module Sections:**

Designed and implemented 12 different sections/modules, enhancing the project's usability.

- **Finplate Design and Implementation:**

Designed the UI for the finplate section.

Integrated popups for user input and feedback.

Implemented API integration related to the finplate module.

- **UI Alignment and Enhancement:**

Aligned UI elements in accordance with design preferences for a consistent look and feel.

Incorporated minor UI enhancements such as dropdowns to improve user experience.

- **Project Flow Structuring:**

Structured and organized the project flow to ensure logical progression and ease of use.

- **Document Creation:**

Created several documents detailing UI/UX guidelines, design patterns, and functional specifications.

- **Authentication UI:**

Implemented the complete UI for user authentication, including login and registration.

- **Frontend Integration:**

Managed the integration of frontend components, ensuring seamless communication with backend services.

- **User Account UI:**

Designed and developed the user account UI, enabling users to manage their profiles effectively.

- **API Implementation and Testing:**

Implemented and thoroughly tested various APIs and endpoints to guarantee functionality and reliability.

- **Summary:**

Throughout my involvement in the project, I have primarily focused on UI development and the successful implementation of various functionalities. This includes designing UI components, integrating APIs, aligning with design preferences, and enhancing the overall user experience. I have also been responsible for structuring the project flow, creating documentation, and ensuring the smooth integration of frontend elements with backend services.



# Chapter 3

## Version of Development

### 3.0.1 Package.json

The `package.json` file contains a list of dependencies used in the project. These dependencies contribute to various functionalities and features within the application.

```
{
  "dependencies": {
    "@mui/icons-material": "^5.14.3",
    "@mui/material": "^5.13.4",
    "@react-pdf-viewer/core": "^3.12.0",
    "@react-three/drei": "^9.74.16",
    "@react-three/fiber": "^8.13.3",
    "@reduxjs/toolkit": "^1.9.5",
    "antd": "^5.5.0",
    "axios": "^0.24.0",
    "base-64": "^1.0.0",
    "crypto-browserify": "^3.12.0",
    "js-file-download": "^0.4.12",
    "jwt-decode": "^3.1.2",
    "pdfjs-dist": "^2.11.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-redux": "^8.0.7",
    "react-router-dom": "^6.14.2",
    "react-toastify": "^8.0.0",
    "three": "^0.153.0"
  }
}
```

### 3.0.2 Dependencies and Their Uses

Here are some of the key dependencies used in the Osdag project along with their purposes and links for more information:

- **@mui/icons-material** and **@mui/material**: These packages are part of the Material-UI library, which provides pre-styled UI components for a polished design. [<https://mui.com/>]
- **@react-pdf-viewer/core**: This package enables rendering and viewing PDF documents within the application. [<https://react-pdf-viewer.dev/>]
- **@react-three/drei** and **@react-three/fiber**: These packages are used for integrating 3D graphics and effects into the application using the Three.js library. [<https://github.com/pmndrs/react-three-fiber>]
- **@reduxjs/toolkit**: Redux Toolkit simplifies state management by providing utilities for managing state, actions, and reducers. [<https://redux-toolkit.js.org/>]
- **antd**: Ant Design is a UI library that offers a variety of components for building user interfaces. [<https://ant.design/>]
- **axios**: Axios is used for making HTTP requests to APIs, facilitating data fetching and interaction with the back-end. [<https://axios-http.com/>]
- **react-router-dom**: This package enables client-side routing, allowing navigation between different components without a full page reload. [<https://reactrouter.com/>]
- **react-toastify**: React Toastify provides customizable toast notifications to display user feedback. [<https://fkhadra.github.io/react-toastify/>]
- **three**: Three.js is a library for creating 3D graphics and visualizations in the browser. [<https://threejs.org/>]

Please make sure to keep the versions of these dependencies up-to-date to ensure compatibility and take advantage of the latest features and improvements.

```
npm install @mui/icons-material@^5.14.3
npm install @mui/material@^5.13.4
npm install @react-pdf-viewer/core@^3.12.0
npm install @react-three/drei@^9.74.16
npm install @react-three/fiber@^8.13.3
npm install @reduxjs/toolkit@^1.9.5
npm install antd@^5.5.0
npm install axios@^0.24.0
npm install base-64@^1.0.0
npm install crypto-browserify@^3.12.0
npm install js-file-download@^0.4.12
npm install jwt-decode@^3.1.2
npm install pdfjs-dist@^2.11.0
npm install react@^18.2.0
npm install react-dom@^18.2.0
npm install react-redux@^8.0.7
npm install react-router-dom@^6.14.2
npm install react-toastify@^8.0.0
npm install three@^0.153.0
```

Figure 3.1: npm commands

# Chapter 4

## Flow Tree and Structure

The project's directory structure is as follows:

```
APP_CRASH
cad
design_report
design_type
documentation
drawing_2D
file_storage
freecad_utils
gui
osdag
osdagclient
osdag_api
osdag_web
ResourceFiles
static
texlive
themes
utilities
utils
.gitignore
.travis.yml
4-Postgres-Installation.sh
bc_endplate_test.py
cad_test.py
% ... (other file entries)
```

This representation provides a visual structure of the project's organization. Here's a brief explanation of each folder and its contents:

**APP\_CRASH:** The directory for crash reports and error logs.

**cad:** Storage for CAD calculation files.

**design\_report:** Contains logic and code in Python to create design reports.

**design\_type:** Specifies design types, such as beam, column, connection, etc. Along with Python files.

- beam\_column
- compression\_member
- connection
- flexural\_member
- frame\_2D
- frame\_3D
- group\_design
- plate\_girder
- tension\_member
- truss
- design\_type.py
- main.py
- member.py

**documentation:** Installation guidelines and documentation by Atharv Pingle.

**drawing\_2D:** Storage for 2D drawings.

**file\_storage:** Contains subfolders for CAD modules and design reports, storing .brep and .aux files, respectively.

- Cad Modules: Store .brep files of CAD modules.
- Design\_report: Store reports in .aux format.

**freecad\_utils:** Utility files for FreeCAD.

**gui:** GUI related files.

**osdag:** Subfolder **web\_api** contains APIs.

**osdagclient:** Main folder for the frontend, including subfolders **assets**, **components**, and **context**.

**osdag\_api:** Contains API endpoints for various functionalities.

**osdag\_web:** Contains **secret\_key.py** for OSDAG web.

**ResourceFiles:** Contains database files and design examples in .osi format.

**static**, **texlive**, **themes**, **utilities**, **utils:** Miscellaneous folders for static assets, LaTeX distribution, themes, utilities, and helper scripts.

**.gitignore, .travis.yml, 4-Postgres-Installation.sh, ...:** Various project configuration and script files.

**bc\_endplate\_test.py, cad\_test.py, ...:** Python scripts and test files.

# Chapter 5

## Design Preferences UI

In this chapter, we will explore the various tabs and features of the Design Preferences User Interface (UI) for beam-column structures. The UI allows users to customize mechanical properties and design parameters to generate accurate calculations and models. Let's take a closer look at each tab:

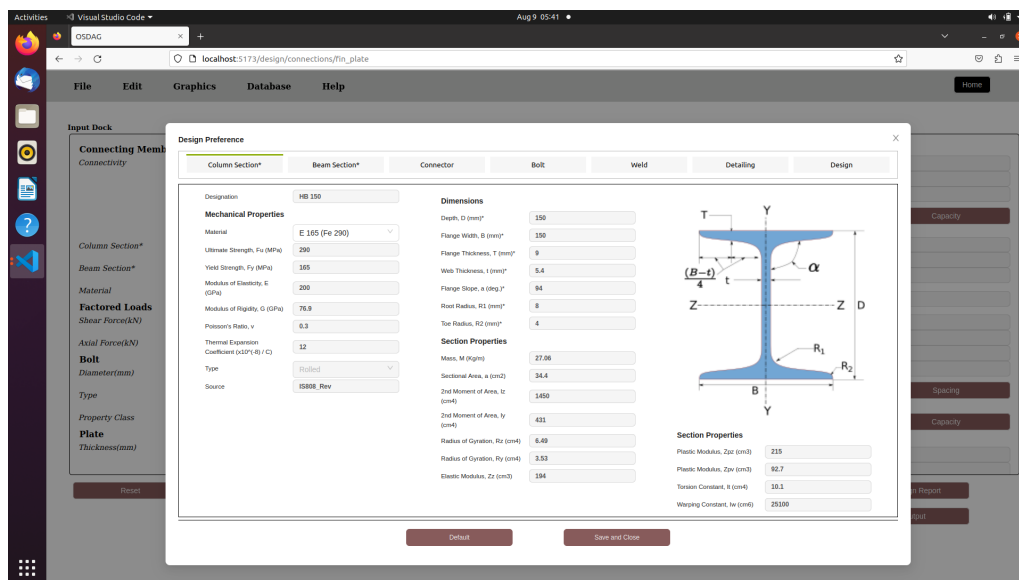


Figure 5.1: Design Preferences UI

### • Column Section:

- This tab is dedicated to defining and customizing properties of column sections, such as shape, dimensions, and material.
- Users can input parameters like section type (rectangular, circular, etc.), dimensions (width, depth, diameter, etc.), and select or create custom materials.
- Customizable mechanical properties include Young's Modulus, Poisson's ratio, yield strength, and ultimate strength.

- **Beam Section:**

- Similar to the Column Section, this tab focuses on beam properties and customization.
- Users can define beam cross-sectional properties like shape, dimensions, and material.
- Input parameters may include section type (I-beam, T-beam, etc.), dimensions (flange width, web thickness, etc.), and material specifications.
- Customizable mechanical properties include elastic modulus, shear modulus, yield strength, and more.

- **Connector:**

- This tab deals with connections between various structural members, such as beams and columns.
- Users can define connection types (welded, bolted, etc.) and customize relevant parameters.
- Customization options may include connector dimensions, types of fasteners, and their spacing.
- Mechanical properties could include shear strength, bearing strength, and other relevant connection properties.

- **Bolt:**

- In this tab, users can specify properties related to bolts used in connections.
- Parameters may cover bolt diameter, length, grade, and material.
- Mechanical properties of bolts like tensile strength, shear strength, and preload can be customized.

- **Weld:**

- This section allows users to define and customize weld properties used in connections.
- Parameters might include weld type, size, and material.
- Customizable mechanical properties include weld strength and efficiency.

- **Detailing:**

- The detailing tab focuses on providing additional information for design and construction drawings.
- Users can input preferences for annotations, dimensions, and symbols used in the generated drawings.

- **Design:**

- This tab is where the actual structural design calculations are performed.



- Users can input design loads, boundary conditions, and other relevant design parameters.
- The software uses the provided inputs from previous sections to calculate and generate structural design results, such as required member sizes, reinforcement details, and connection forces.

## 5.1 Working

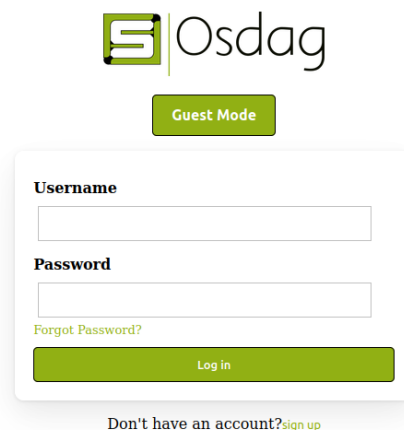
- **Technology Stack:**

OSDAG-web is built using a combination of modern web development technologies. React.js is employed for creating the interactive and dynamic frontend user interface. Django, a powerful Python web framework, serves as the backend to manage data, business logic, and user authentication. PostgreSQL is the chosen database system to store and retrieve application data securely.

- **Folder Structure and Project Launch:**

The project's folder structure follows best practices for maintainability. The `osdagclient` folder contains the frontend components, including JavaScript files, stylesheets, and assets. To launch the project, the necessary dependencies are configured, and the server is started, initiating the rendering of the initial page – the `LoginPage.jsx`.

- **Authentication:**



The screenshot shows the OSDAG web application's login interface. At the top center is the OSDAG logo, consisting of a green square with a white 'S' and the word 'Osdag' in a grey sans-serif font. Below the logo is a green button with the text 'Guest Mode'. The main part of the page is a white login form with a subtle shadow. Inside the form, there are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Below the password field is a green link that says 'Forgot Password?'. At the bottom of the form is a wide green button labeled 'Log in'. Below the entire form, centered, is the text 'Don't have an account?' followed by a green link that says 'sign up'.

Figure 5.2: Login

LoginPage.jsx is where the user authentication process begins. The authentication code implemented here ensures secure user access to the application. It handles user account creation (signup) and existing user logins. Upon successful login, users are granted access to the main page.

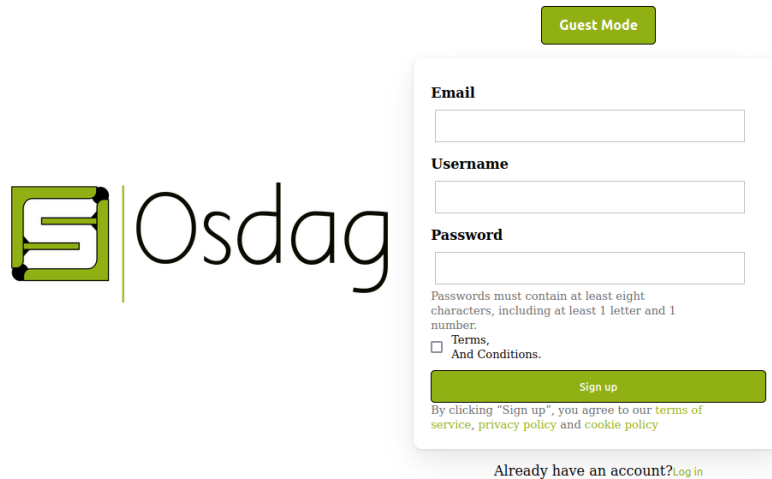
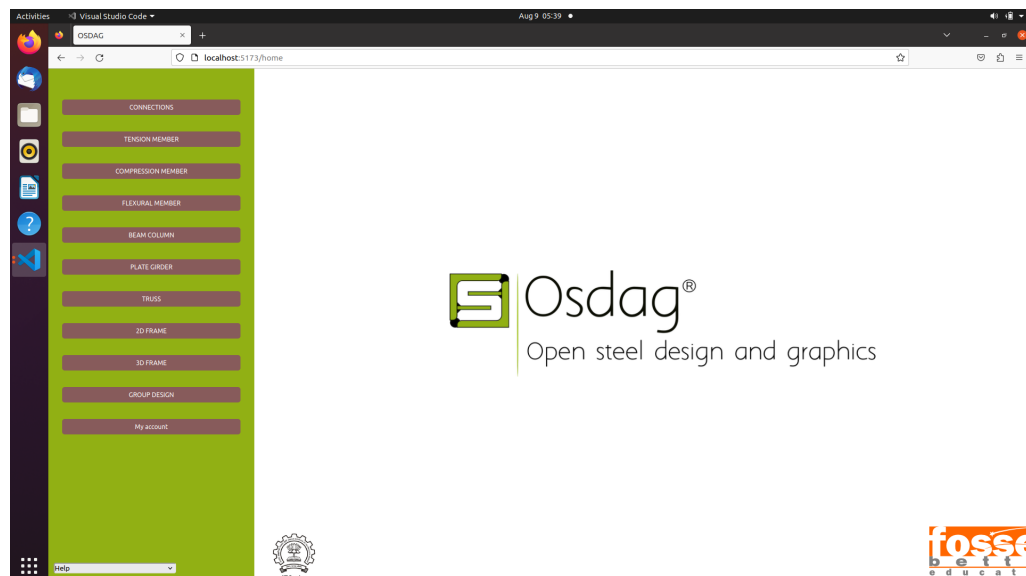


Figure 5.3: Signup

- **Main Page:** The main page serves as the central hub for the application's functionality and features.



- **Modules Section:**

Under the "All Modules" section, various modules related to structural engineering tasks are available. These modules are designed to help engineers

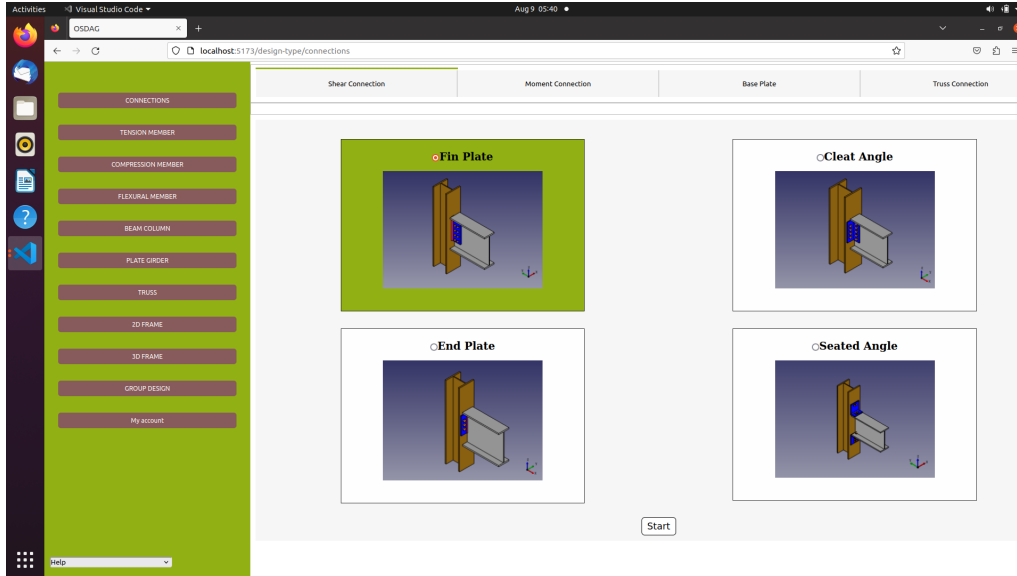


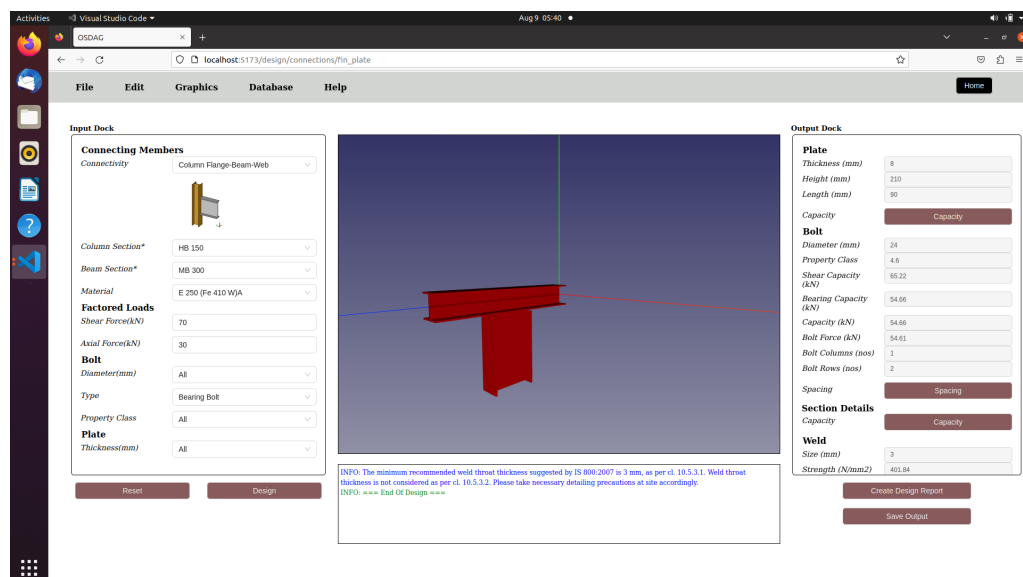
Figure 5.4: All Modules

and users perform different analyses and calculations related to their projects. Each module provides a dedicated workspace for specific tasks.

- **Connections Module:**

The Connections module focuses on designing connections between structural components. It offers several sub-modules, each catering to a specific type of connection. One such sub-module is the Shear Connection.

- **Fin Plate Model:** Clicking on the "Fin Plate" model initiates a detailed design process, encompassing the following sections:



- **Input Dock:**

The Input Dock is the starting point for generating the CAD model. Users provide parameters such as Connectivity type, Column and Beam

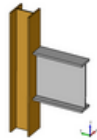
Sections. The validation process ensures that only valid inputs are accepted, preventing errors downstream.

#### Input Dock

### Connecting Members

*Connectivity*

Column Flange-Beam-Web



*Column Section\**

HB 150

*Beam Section\**

MB 300

*Material*

E 250 (Fe 410 W)A

### Factored Loads

*Shear Force(kN)*

70

*Axial Force(kN)*

30

### Bolt

*Diameter(mm)*

All

*Type*

Bearing Bolt

*Property Class*

All

### Plate

*Thickness(mm)*

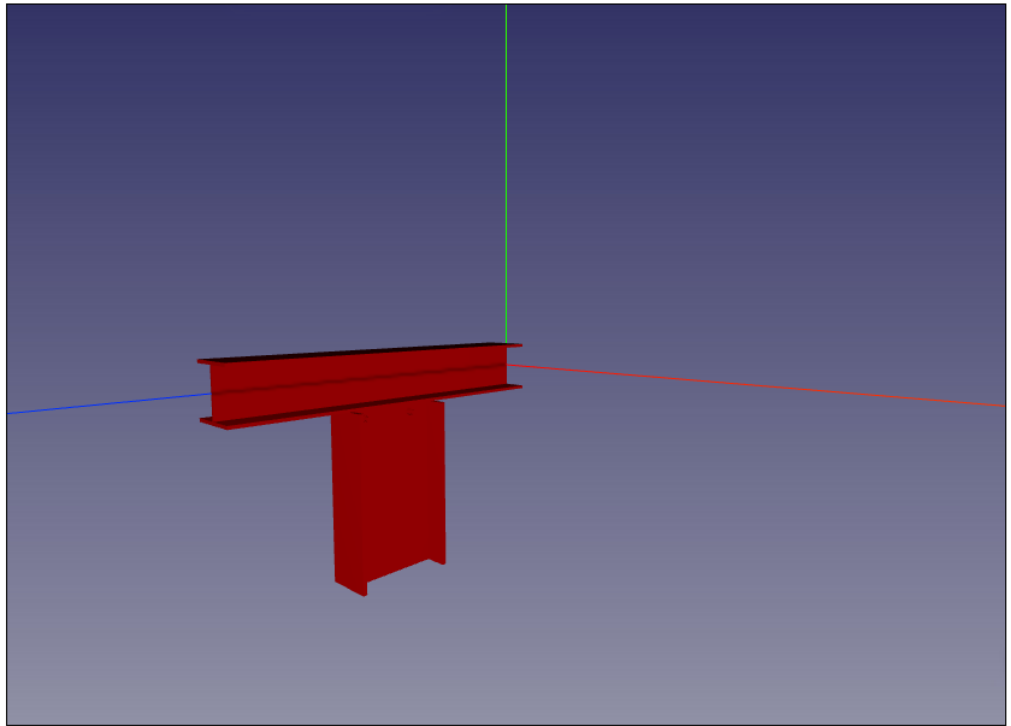
All

Reset

Design

#### – CAD Model View and Log View:

In the CAD Model View, users can interact with a three-dimensional representation of the designed connection. This visualization is powered by Three.js, a JavaScript library for 3D graphics. The Log View, located below the CAD model, displays real-time feedback on the validity of inputs and the status of the CAD generation process. Errors, warnings, and successful generation messages are shown here.



INFO: The minimum recommended weld throat thickness suggested by IS 800:2007 is 3 mm, as per cl. 10.5.3.1. Weld throat thickness is not considered as per cl. 10.5.3.2. Please take necessary detailing precautions at site accordingly.

INFO: === End Of Design ===

- **Output Dock:** The Output Dock presents users with calculated outputs based on the provided inputs. This section details the properties of the plate (Thickness, Height, Length) and bolt (Diameter, Property Class, Shear Capacity). These outputs are crucial for engineers to evaluate the feasibility and safety of the design.

**Output Dock**

<b>Plate</b>	
Thickness (mm)	8
Height (mm)	210
Length (mm)	90
Capacity	Capacity
<b>Bolt</b>	
Diameter (mm)	24
Property Class	4.6
Shear Capacity (kN)	65.22
Bearing Capacity (kN)	54.66
Capacity (kN)	54.66
Bolt Force (kN)	54.61
Bolt Columns (nos)	1
Bolt Rows (nos)	2
Spacing	Spacing
<b>Section Details</b>	
Capacity	Capacity
<b>Weld</b>	
Size (mm)	3
Strength (N/mm2)	401.84

Create Design Report

Save Output

– **Design Report and Test Cases:**

Within the Output Dock, users have options to generate a comprehensive design report and save output data for future reference. The design report includes applied test cases, aiding engineers in assessing whether the design meets safety and performance standards.

– **Drop-down Menu:**

The Drop-down Menu enhances user experience by organizing functionalities:

- \* Load Input: Users can load a saved `.osi` file, populating the Input Dock with the corresponding parameters.
- \* Download Input: Allows users to download the current input as an `.osi` file for later use.
- \* Save Input: Enables users to save the current input in their dashboard for easy retrieval.
- \* Save Log Message: Provides a way to save log status and information for documentation purposes.
- \* Create Design Report: Initiates the process of generating a detailed design report.

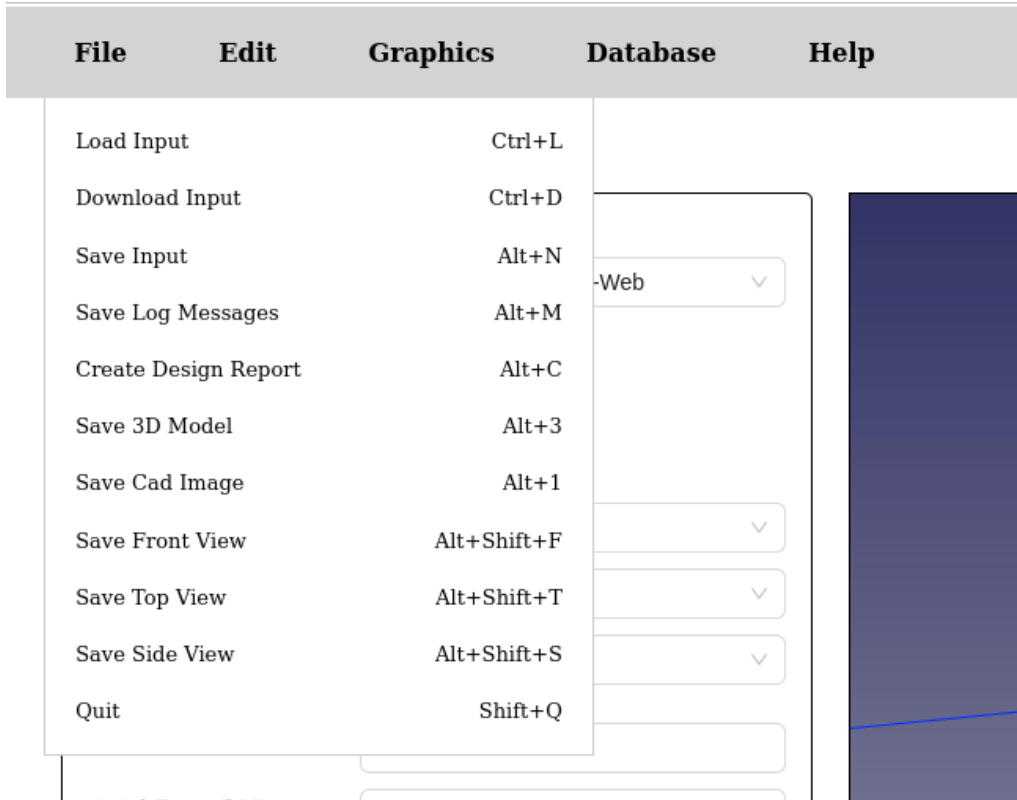


Figure 5.5: Drop-down menu

- \* Home Button: Users can quickly navigate back to the homepage from any section.

- **User Account:**

The "My Account" section empowers users to manage their profile and interact with their saved data.

- **Dashboard:**

Users can view a dashboard displaying their saved .osi files. These files represent various design scenarios and configurations. Each entry provides options for downloading and viewing.

- **Logout Button:**

The logout button ensures users can securely log out of their accounts. It clears cookies and local storage data associated with the user session, enhancing privacy and security.

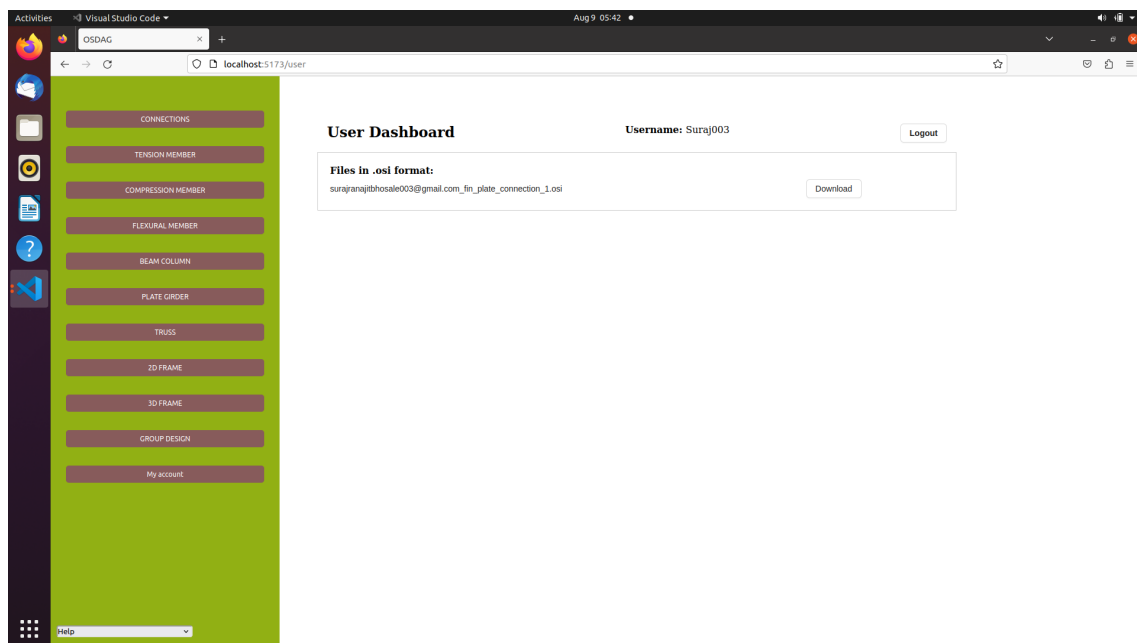


Figure 5.6: User Account

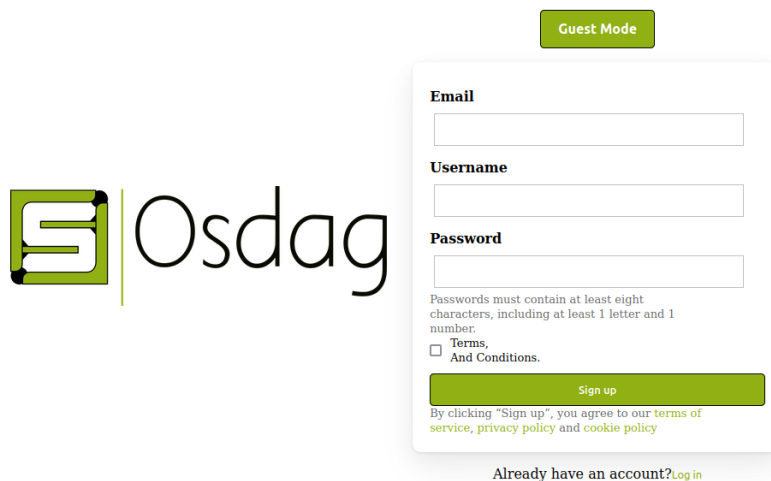


# Chapter 6

## Authentication

### 6.1 UI and Functionality

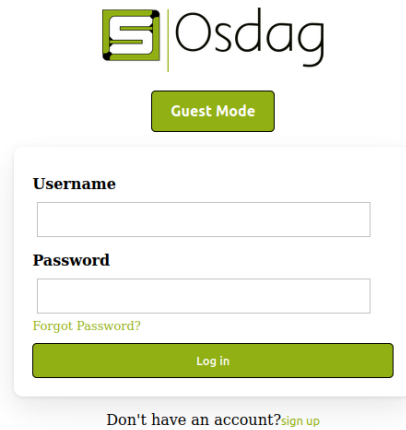
#### 6.1.1 Signup



The image shows a web interface for signing up. On the left is the Osdag logo, which consists of a stylized 'S' inside a green square followed by the word 'Osdag'. To the right is a white signup form with a green 'Guest Mode' button at the top. The form contains three input fields labeled 'Email', 'Username', and 'Password'. Below the 'Password' field is a note: 'Passwords must contain at least eight characters, including at least 1 letter and 1 number.' There is a checkbox labeled 'Terms, And Conditions.' and a green 'Sign up' button. Below the button, it says 'By clicking "Sign up", you agree to our terms of service, privacy policy and cookie policy'. At the bottom of the form, it says 'Already have an account? Log in'.

When a user wants to sign up, they are presented with a signup form containing three input fields: Email, Username, and Password. Upon submitting the form, the data is stored in a PostgreSQL database for future authentication and access.

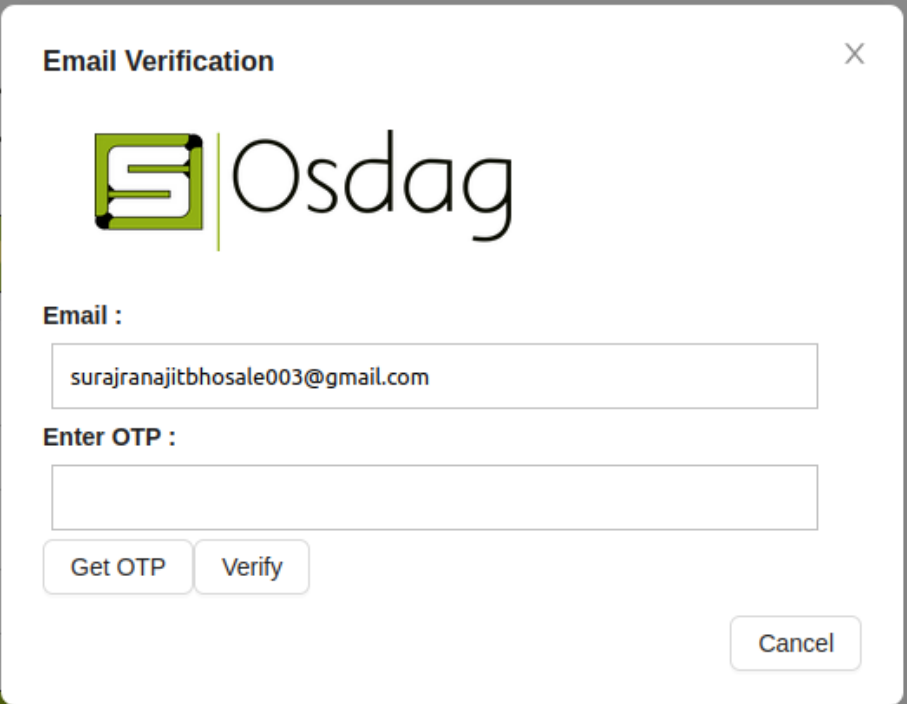
## 6.1.2 Login



The screenshot shows the login interface for 'Osdag'. At the top center is the logo, which consists of a green square with a white 'S' inside, followed by the word 'Osdag' in a black sans-serif font. Below the logo is a green rectangular button with the text 'Guest Mode' in white. Underneath this is a white login form with a subtle drop shadow. The form contains two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Below the password field is a link that says 'Forgot Password?' in green text. At the bottom of the form is a wide green button with the text 'Log in' in white. Below the entire form, centered, is the text 'Don't have an account?' followed by a green link that says 'sign up'.

The login screen provides two input fields: Username and Password. The system uses the username (considered as a primary key) to search for the user's data in the database. Additionally, the system includes a "Forgot Password" functionality, allowing users to reset their passwords.

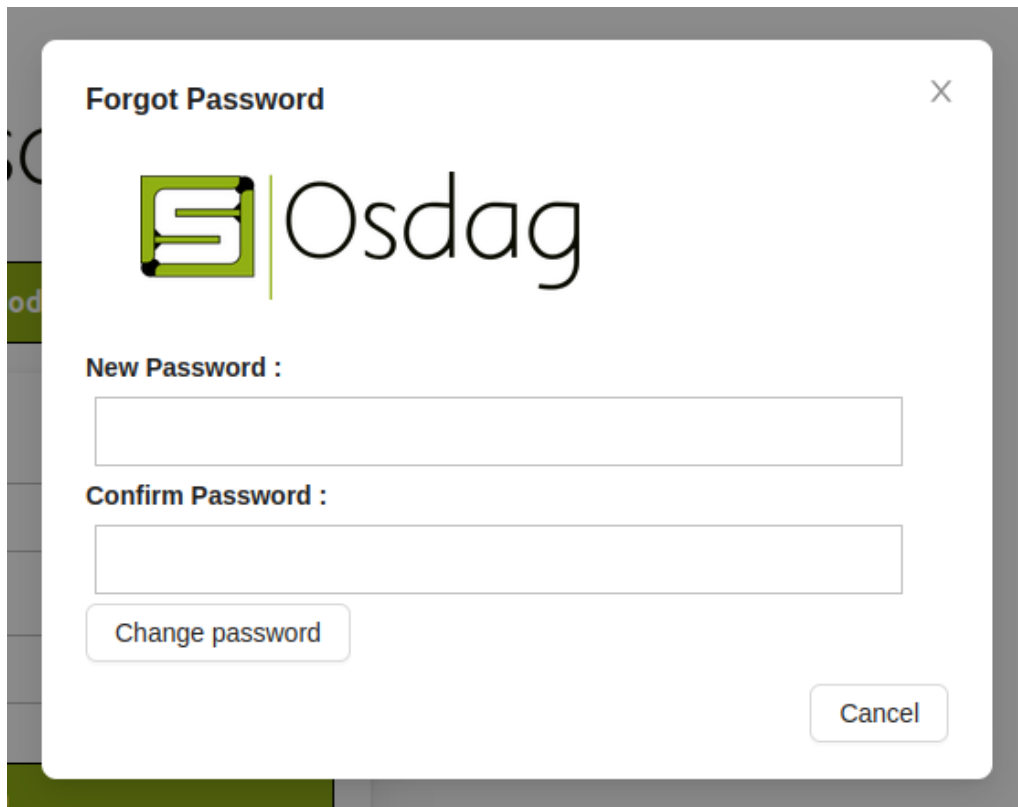
### 6.1.3 Forget Password: OTP Verification



The image shows a modal dialog box titled "Email Verification" with a close button (X) in the top right corner. The dialog features the Osdag logo, which consists of a green square icon with a white stylized 'S' followed by the word "Osdag" in a sans-serif font. Below the logo, there is a label "Email :" followed by a text input field containing the email address "surajranajitbhosale003@gmail.com". Underneath this is a label "Enter OTP :" followed by an empty text input field. At the bottom left of the dialog are two buttons: "Get OTP" and "Verify". At the bottom right is a "Cancel" button. The dialog is set against a dark gray background with some blurred text from the underlying application visible on the left.

In case a user forgets their password, they can request a password reset by entering their registered email address and clicking on the "Send OTP" button. An OTP (One-Time Password) is sent to their email address. After receiving the OTP, the user enters it in the verification interface to confirm their identity.

### 6.1.4 Forget Password: Add New Password



Once the user successfully verifies their OTP, they are directed to the "Add New Password" screen. Here, the user is required to enter and confirm their new password. After clicking the "Save" button, the new password is securely stored in the database, allowing the user to log in with the updated credentials.

## 6.2 API's and Contexts

### 6.2.1 API's

#### 1. User Signup

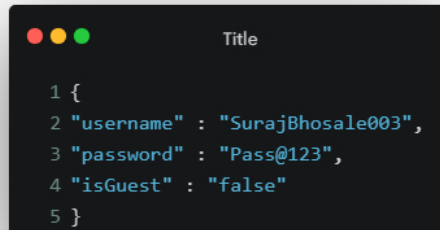


**Endpoint:** `http://localhost:8000/user/signup/` **Method:** POST

This endpoint allows you to create a new user account using the HTTP POST method. You need to send encrypted data in the request body, including the

username, password, and email. No special authentication is required for this request.

## 2. User Login

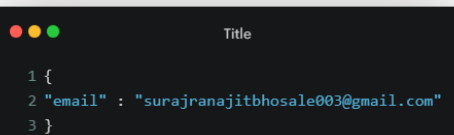


```
1 {  
2  "username" : "SurajBhosale003",  
3  "password" : "Pass@123",  
4  "isGuest" : "false"  
5 }
```

**Endpoint:** `http://localhost:8000/user/login/` **Method:** POST

This API endpoint is likely used for user login. To log in, you need to send a JSON request with the username, password, and a flag indicating whether the user is a guest.

## 3. OTP Verification

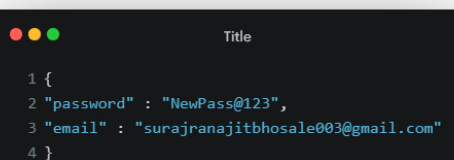


```
1 {  
2  "email" : "surajranajitbhosale003@gmail.com"  
3 }
```

**Endpoint:** `http://localhost:8000/user/checkemail/` **Method:** POST

This API endpoint represents a process where you provide an email address, and in response, you receive a message and a one-time password (OTP) via email. You then ask the user to enter the OTP for verification, comparing it with the received OTP to confirm the authenticity of the email address.

## 4. Confirm Forget Password



```
1 {  
2  "password" : "NewPass@123",  
3  "email" : "surajranajitbhosale003@gmail.com"  
4 }
```

**Endpoint:** `http://localhost:8000/user/forgetpassword/` **Method:** POST

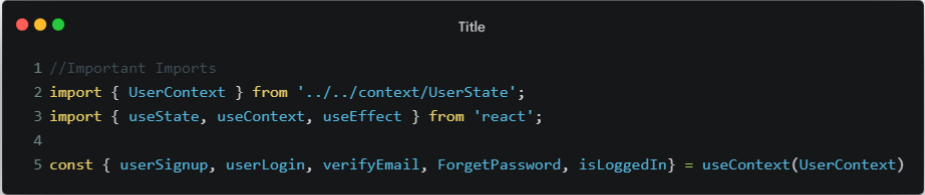
After OTP verification, users are directed to set a new password. Once they confirm the new password, the system updates the user's password in the

database with the provided email address. This process ensures secure password changes and updates the database accordingly.

## 6.3 Working in Code

### 6.3.1 Important Imports

#### 1. Imports



```
1 //Important Imports
2 import { useContext } from '../context/UserState';
3 import { useState, useEffect } from 'react';
4
5 const { userSignup, userLogin, verifyEmail, forgetPassword, isLoggedIn } = useContext(UserContext)
```

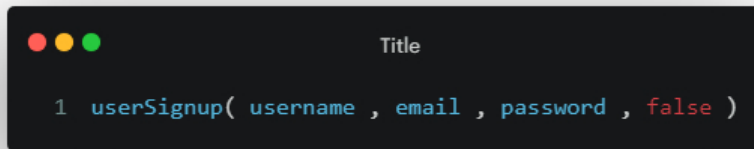
In the provided code snippet, we are working within a React application. The code focuses on managing user-related functionality using a `UserContext` from the `'../context/UserState'` module. This context likely encompasses user data and actions. Additionally, we are utilizing three crucial React hooks: `useState`, `useContext`, and `useEffect`.

To streamline the process, the `useContext` hook enables us to extract specific functions and data from the `UserContext`. This includes:

- `userSignup`: A function responsible for user registration.
- `userLogin`: A function facilitating user login.
- `verifyEmail`: A function used to verify the user's email address.
- `forgetPassword`: A function that initiates the password reset process.
- `isLoggedIn`: A boolean flag indicating the user's login status.

This setup empowers us to seamlessly integrate user authentication and management into our component. By harnessing these functions and data from the `UserContext`, we can proficiently handle user actions such as registration, login, email verification, password reset, and ascertain whether a user is currently logged in or not.

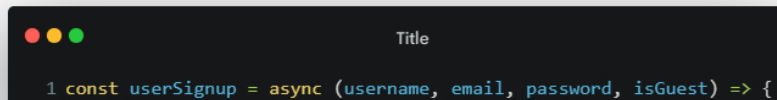
### 6.3.2 SignUp



```
1 userSignup( username , email , password , false )
```

In react This SignUp process triggers the registration process for a new user. It supplies the chosen username, email, password, and marks the account as unverified initially.

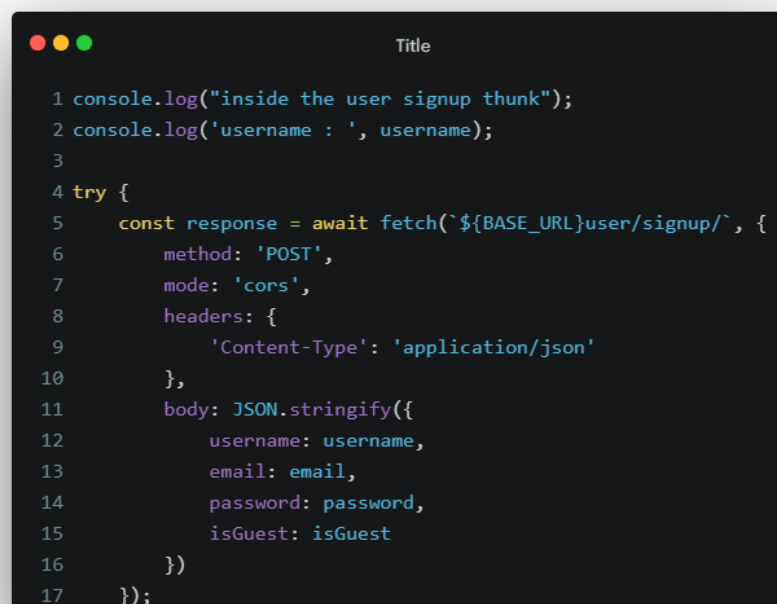
- **Function Signature:**



```
1 const userSignup = async (username, email, password, isGuest) => {
```

The function takes four parameters: username, email, password, and isGuest. These parameters are presumably used to gather user registration information.

- **Logging and Fetching:**



```
1 console.log("inside the user signup thunk");
2 console.log('username : ', username);
3
4 try {
5   const response = await fetch(`${BASE_URL}user/signup/`, {
6     method: 'POST',
7     mode: 'cors',
8     headers: {
9       'Content-Type': 'application/json'
10    },
11    body: JSON.stringify({
12      username: username,
13      email: email,
14      password: password,
15      isGuest: isGuest
16    })
17  });
```

Here, the function logs some information for debugging purposes. It then uses the fetch function to send a POST request to the specified URL (user/signup/). The request includes the provided username, email, password, and isGuest parameters in the request body as a JSON payload.

- **Processing Response:**



```
1  const jsonResponse = await response?.json();
2  console.log('jsonResponse : ', jsonResponse);
3
4  if (response.status === 201) {
5    // User registration successful
6    console.log('user successfully created');
7
8    // Call the thunk for creating the JWT token
9    createJWTToken(username, password);
10
11    // Call the reducer action to set the Login variable
12    dispatch({ type: 'SET_SIGNUP_STATUS', payload: { isLoggedIn: false, message: "User Successfully Signed up" } });
13
14    console.log('isLoggedIn in signup thunk : ', state.isLoggedIn);
15  } else {
16    // User registration failed
17    console.log('response.status is not 201, failed to create a new user');
18    dispatch({ type: 'SET_SIGNUP_STATUS', payload: { isLoggedIn: false, message: "Error in creating the User Account, please try again" } });
19  }
20 } catch (err) {
21   // Server error or exception
22   console.log('there is an error in user signup : ', err);
23   dispatch({ type: 'SET_SIGNUP_STATUS', payload: { isLoggedIn: false, message: "Server Error in creating User Account, please try again" } });
24 }
```

After the fetch request, the code awaits the response and parses it as JSON. If the response status is 201 (indicating successful user creation), the function proceeds to call another function (createJWTToken) to create a JWT token, dispatches a reducer action to update the signup status with a success message, and logs the isLoggedIn state.

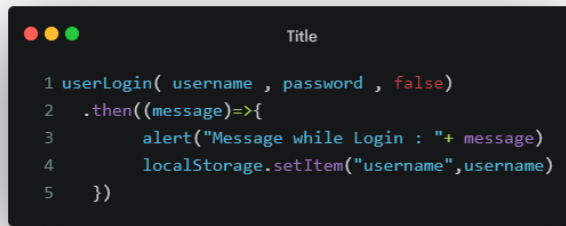
If the response status is not 201, the function dispatches a reducer action with an error message.

If an error occurs during the entire process, such as a network error or server exception, the catch block handles it by dispatching a reducer action with an appropriate error message.

This involves Redux or a similar state management system (as implied by the use of dispatch). It seems to handle user registration, potentially creating a user account, generating a JWT token, and updating the application state accordingly.



### 6.3.3 Login



```
1 userLogin( username , password , false)
2 .then((message)=>{
3     alert("Message while Login : "+ message)
4     localStorage.setItem("username",username)
5 })
```

The `userLogin` function is invoked with three arguments: `username`, `password`, and `false`. This function returns a Promise. When the Promise is fulfilled, it triggers a callback function that performs the following actions:

- **Alert Display:**

An alert is displayed with a message derived from the message received from the Promise resolution, providing feedback about the login process.

- **Handling Guest User:**

If the value of the third argument (`false`) indicates that the user is a guest, no token is created, and the user's information is not stored in the database.

- **Local Storage:**

Regardless of whether the user is a guest or not, the username is stored in the browser's local storage, allowing it to be stored on the user's device for potential future use.

The code handles the login process, provides an alert message, and handles whether to create a token and store the user's information in the database based on whether the user is a guest or not.

## Section 1: Function Declaration and Debugging Logs

```
1 const userLogin = async (username, password, isGst, JWTLogin) => {
2   // Debugging logs
3   console.log('in userlogin Context - inside user login');
4   console.log('in userlogin Context - username : ', username);
5   console.log('in userlogin Context - isGuest : ', isGst);
6
7   // Check if JWTLogin is true
8   if (JWTLogin === true) {
9     dispatch({ type: 'SET_LOGGING_STATUS', payload: { isLoggedIn: true, message: "Login Successful" }
10  });
11   return;
12 }
13
14 try {
15   // Make a POST request to the login endpoint
16   const response = await fetch(`${BASE_URL}user/login/`, {
17     method: 'POST',
18     mode: 'cors',
19     headers: {
20       'Content-Type': 'application/json', // Set the Content-Type header to JSON
21     },
22     body: JSON.stringify({
23       username: username,
24       password: password,
25       isGuest: isGst
26     })
27   });
28   // Parse the response JSON
29   const jsonResponse = await response?.json();
```

In this section:

The `userLogin` function is declared, taking four parameters: `username`, `password`, `isGst`, and `JWTLogin`. Debugging logs are printed to the console, displaying information about the function context, username, and whether the user is a guest (`isGst`). If `JWTLogin` is true, it means a JSON Web Token (JWT) login is being attempted. In this case, a successful login status is set, and the function returns early.

## Section 2: Handling Successful Login Response

```
1   if (response.status === 200) {
2     console.log('user logged in successfully in userLogin Context ');
3
4     // Create a new JWT token and store user information if not a guest
5     if (isGst === false) {
6       createJWTToken(username, password);
7       localStorage.setItem('userType', "user");
8       localStorage.setItem('username', username);
9       localStorage.setItem('email', jsonResponse.email);
10      localStorage.setItem('allInputValueFilesLength', jsonResponse.allInputValueFilesLength);
11    } else {
12      localStorage.setItem('userType', "guest");
13    }
14  }
```

If the HTTP response status is 200 (successful), it means the user has logged in successfully. If the user is not a guest (i.e., `isGst` is false), a new JWT token is created using the `createJWTToken` function (not shown in the provided code). User information is stored in the browser's local storage, including the user type,

username, email, and other data. If the user is a guest (i.e., `isGst` is true), only the user type ("guest") is stored in local storage.

### Section 3: Setting Login Status and Handling Error Messages

```
1      // Set the login status to true
2      dispatch({ type: 'SET_LOGGING_STATUS', payload: { isLoggedin: true, message:
jsonResponse.message } });
3
4      return jsonResponse.message;
5    } else {
6      // Handle different login error cases
7      if (jsonResponse.message === "The User Account does not exists") {
8        dispatch({ type: 'SET_LOGGING_STATUS', payload: { isLoggedin: false, message: "The User
Account does not exist" } });
9      } else if (jsonResponse.message === "Invalid credentials") {
10       dispatch({ type: 'SET_LOGGING_STATUS', payload: { isLoggedin: false, message: "Invalid
Credentials, please try again" } });
11     } else {
12       dispatch({ type: 'SET_LOGGING_STATUS', payload: { isLoggedin: false, message: "Error
while logging" } });
13     }
14
15     return jsonResponse.message;
16   }
17 }
```

If the login is successful, the login status is set to true, and a success message is stored in the payload. If the response status is not 200, the code handles different error cases based on the message received from the server. The login status is set to false, and an appropriate error message is stored in the payload.

### Section 4: Error Handling and Dispatching Server Error Message

```
1   } catch (err) {
2     console.log('error in logging in');
3     dispatch({ type: 'SET_LOGGING_STATUS', payload: { isLoggedin: false, message: "Server error
occurred while logging in, please try again" } });
4
5     return "Server error occurred while logging in, please try again";
6   }
7 }
8 }
```

Here, any errors that occur during the login process are caught. If an error occurs, the login status is set to false, and a server error message is stored in the payload.

Please note that the code relies on external functions like `dispatch` and `createJWTToken`, and the behavior might vary based on their implementations.

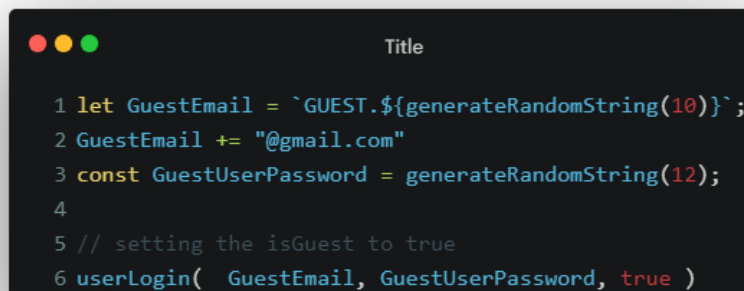
### 6.3.4 Guest User

#### generateRandomString



```
1 const generateRandomString = (length) => {
2   const charset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
3   let result = '';
4   for (let i = 0; i < length; i++) {
5     const randomIndex = Math.floor(Math.random() * charset.length);
6     result += charset[randomIndex];
7   }
8   return result;
9 };
```

The `generateRandomString` function creates a string by randomly selecting characters from a predefined set of lowercase letters, uppercase letters, and digits. The length of the string is determined by the `length` parameter provided when calling the function.



```
1 let GuestEmail = `GUEST.${generateRandomString(10)}`;
2 GuestEmail += "@gmail.com"
3 const GuestUserPassword = generateRandomString(12);
4
5 // setting the isGuest to true
6 userLogin( GuestEmail, GuestUserPassword, true );
```

- **GuestEmail = GUEST.\${generateRandomString(10)};** This line generates a random string of 10 characters using the `generateRandomString` function and then constructs a guest email by appending it to the string "GUEST." and adding "@gmail.com" at the end. This creates a random email address like "GUEST.xxxxxxxxxx@gmail.com".
- **const GuestUserPassword = generateRandomString(12);** This line generates a random string of 12 characters using the `generateRandomString` function, creating a random password for the guest user.
- **userLogin(GuestEmail, GuestUserPassword, true);** This line uses the `userLogin` function to attempt a guest user login. The `GuestEmail` and `GuestUserPassword` are passed as arguments, along with `true` to indicate that the user is a guest.

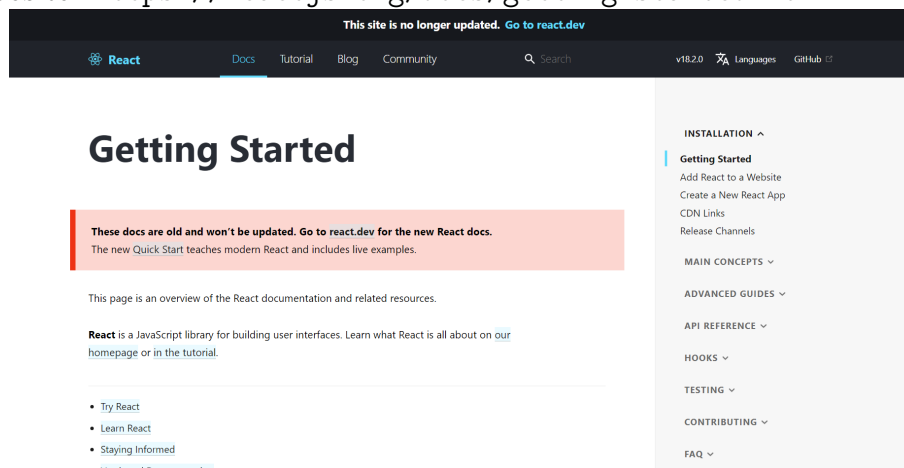
It generates random email and password credentials for a guest user and then attempts to log in the guest user using the `userLogin` function. The `generateRandomString` function is used to create the random email and password.

# Chapter 7

## Websites for UI Integration

### 7.1 React - Official Documentation

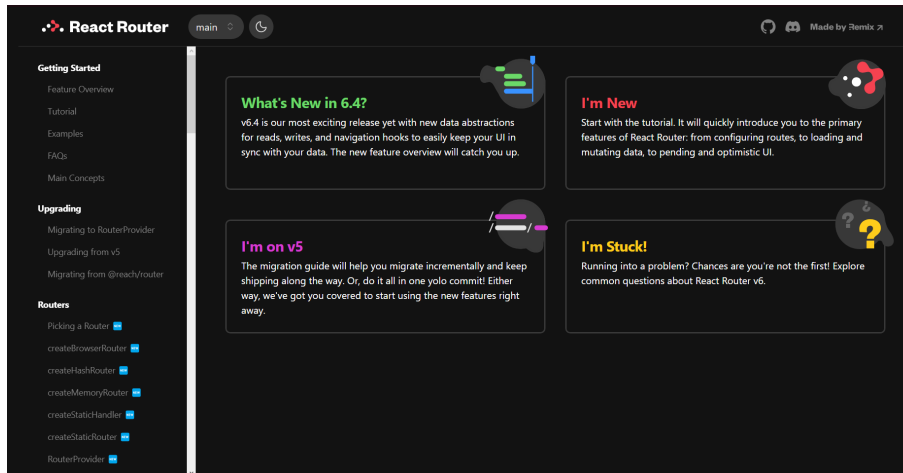
Website: <https://reactjs.org/docs/getting-started.html>



The official documentation for React.js serves as an essential resource for learning about React's fundamental concepts, such as components, props, state, lifecycle, hooks, and more. It provides comprehensive guides and examples to help developers build efficient and maintainable user interfaces using React.

### 7.2 React Router

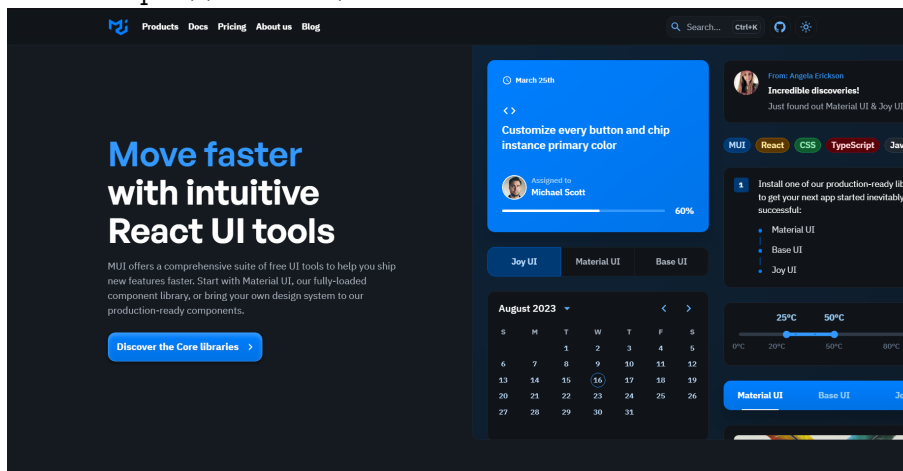
Website: <https://reactrouter.com/>



React Router offers thorough documentation and guides for implementing routing and navigation within React applications. It enables developers to create single-page applications with multiple views and dynamic URL routing, allowing for seamless navigation between different sections of the app.

## 7.3 Material-UI

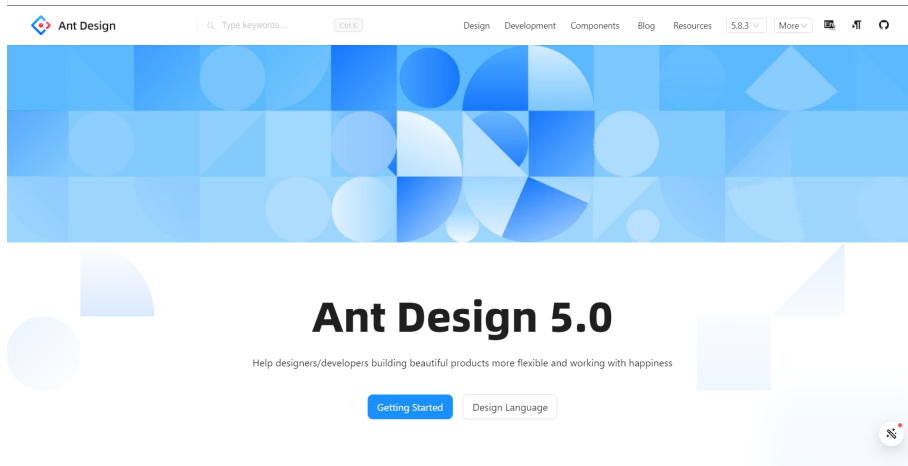
Website: <https://mui.com/>



Material-UI is a widely-used React UI framework that provides a comprehensive set of pre-designed components following Google's Material Design guidelines. It offers a cohesive and visually appealing design system for building modern and responsive user interfaces.

## 7.4 Ant Design

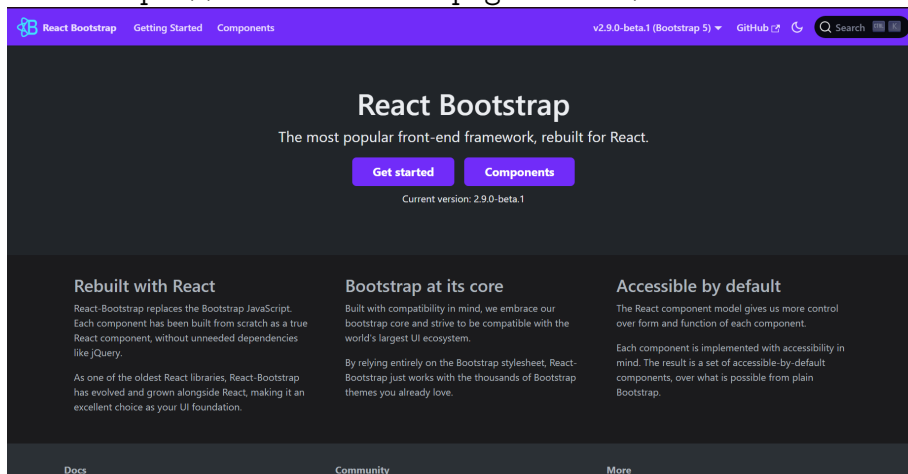
Website: <https://ant.design/>



Ant Design offers a collection of high-quality and customizable React components designed with a focus on elegance and usability. It comes with a set of design principles and guidelines that help developers create professional and intuitive user interfaces.

## 7.5 React Bootstrap

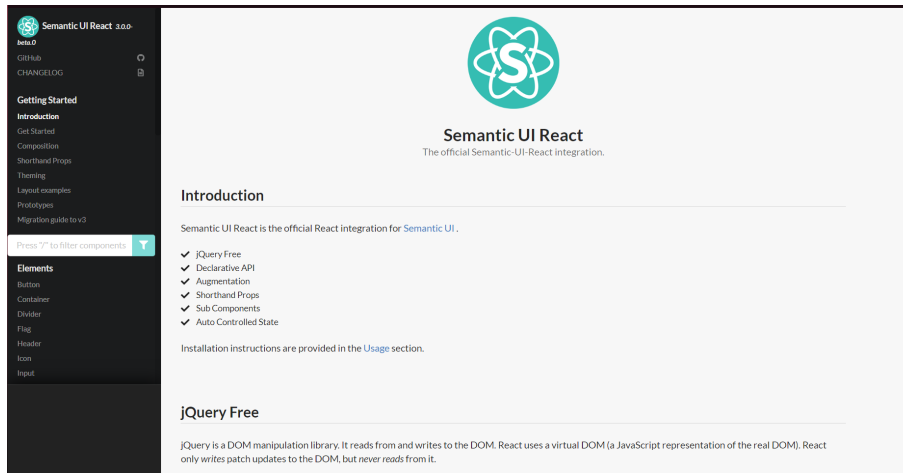
Website: <https://react-bootstrap.github.io/>



React Bootstrap brings the popular Bootstrap framework's components to the world of React. It provides a wide range of ready-to-use components and utilities, making it easy to create responsive and consistent UIs.

## 7.6 Semantic UI React

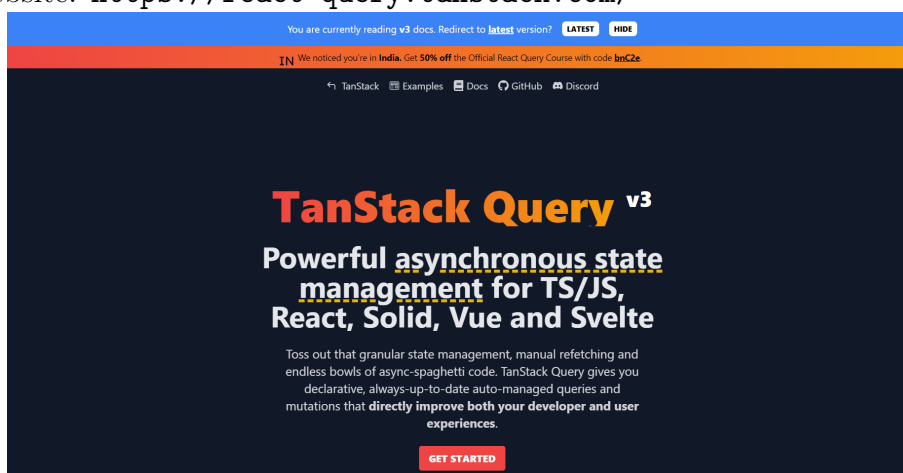
Website: <https://react.semantic-ui.com/>



Semantic UI React provides a suite of React components that adhere to the Semantic UI design language. It allows developers to create expressive and semantic user interfaces using a set of well-designed building blocks.

## 7.7 React Query

Website: <https://react-query.tanstack.com/>

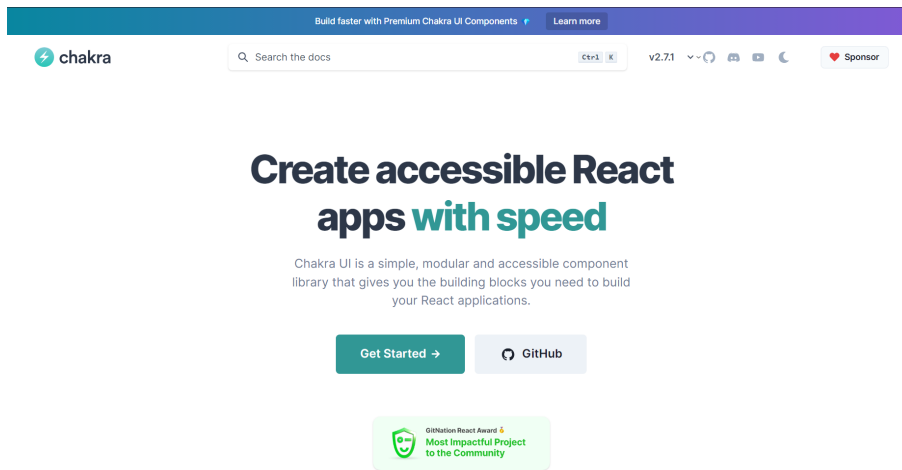


React Query simplifies data fetching, caching, synchronization, and state management in React applications. It offers tools to manage complex data requirements, such as querying APIs, managing mutations, and handling pagination.

## 7.8 Chakra UI

Website: <https://chakra-ui.com/>

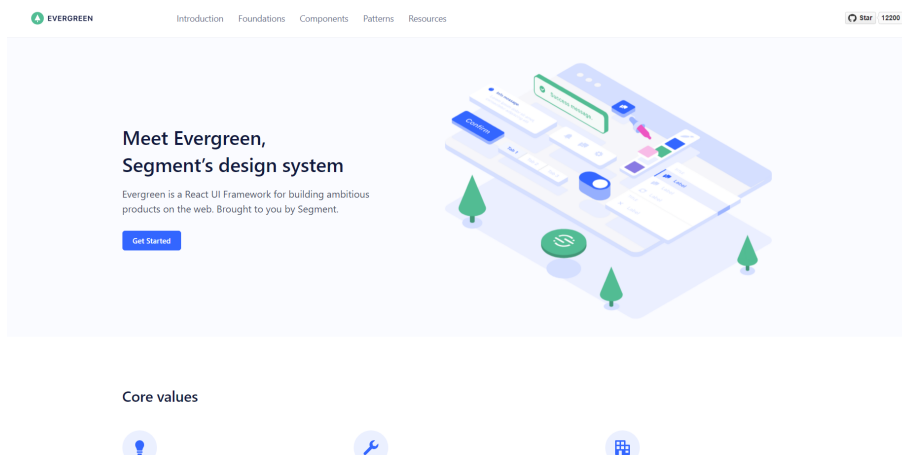




Chakra UI is a modular and accessible component library that aims to streamline the process of building user interfaces in React applications. It provides a set of responsive and customizable components that adhere to design and accessibility best practices.

## 7.9 Evergreen

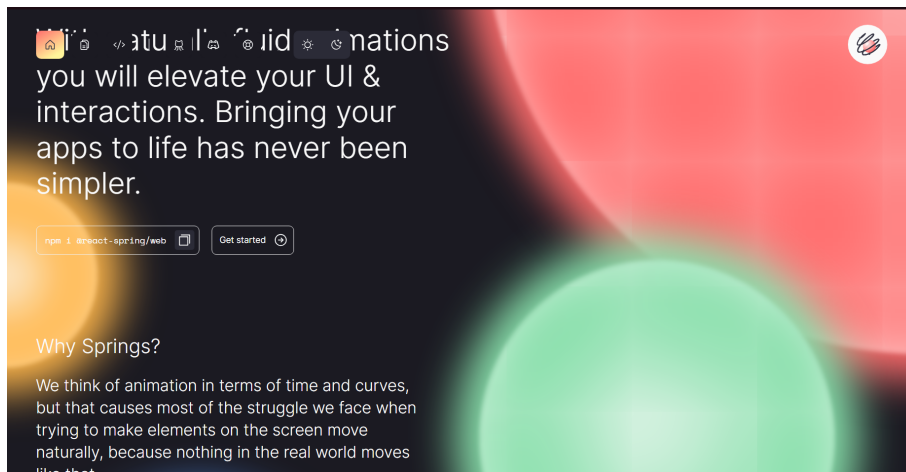
Website: <https://evergreen.segment.com/>



Evergreen is a UI component library for React designed to help developers create consistent and functional user interfaces. It offers a set of components that work seamlessly together to build design systems and user interfaces.

## 7.10 React Spring

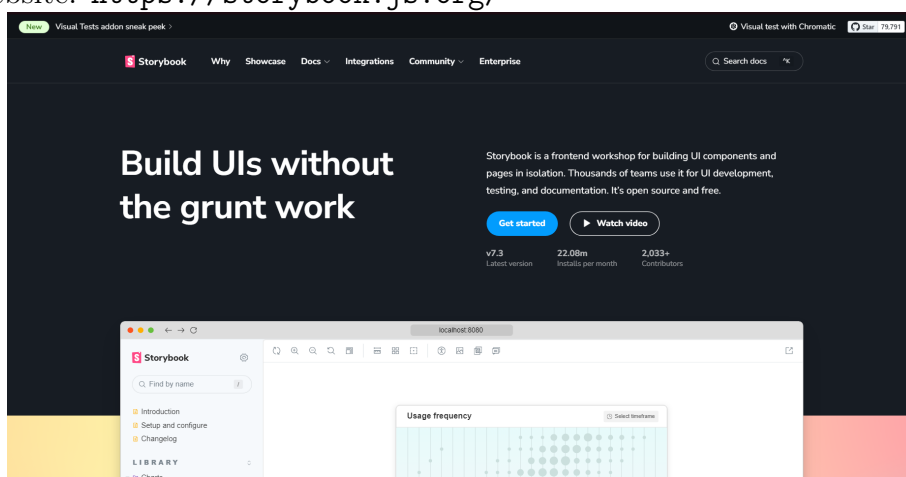
Website: <https://react-spring.io/>



React Spring is a library for creating fluid animations and transitions in React applications. It enables developers to add smooth and dynamic animations to UI elements, enhancing the user experience.

## 7.11 Storybook

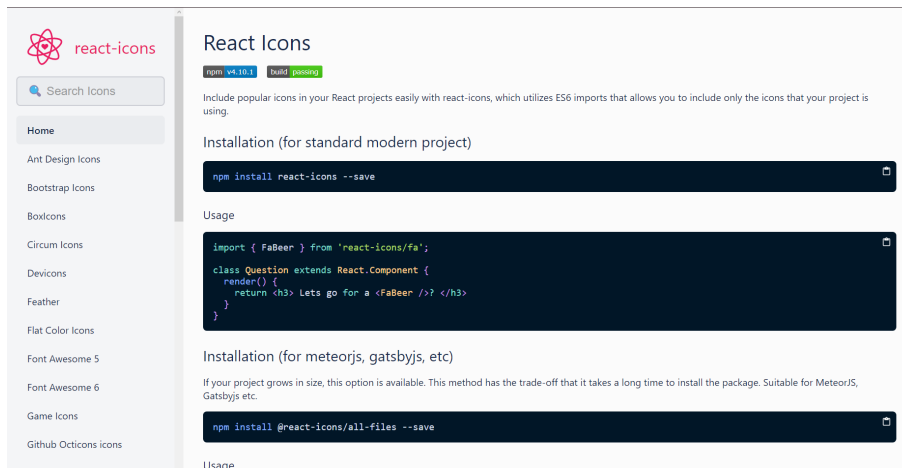
Website: <https://storybook.js.org/>



Storybook is a development environment and UI component explorer for React applications. It allows developers to isolate and develop UI components in isolation, facilitating testing, documentation, and showcasing of components.

## 7.12 React Icons

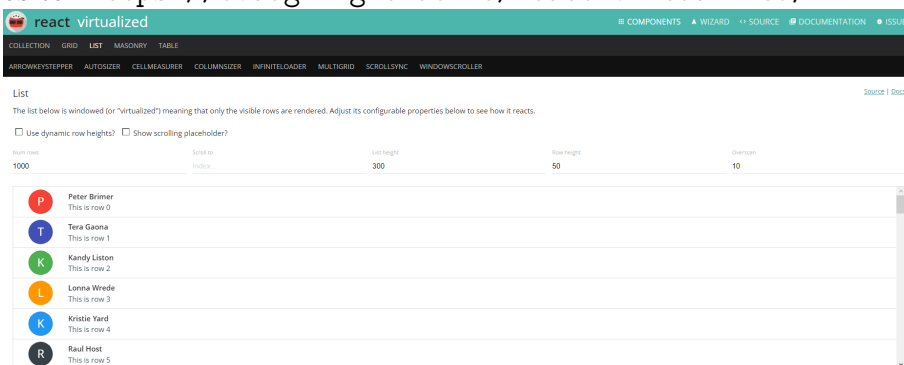
Website: <https://react-icons.github.io/react-icons/>



React Icons provides a collection of popular icon libraries as ready-to-use React components. It simplifies the process of integrating icons into React applications, enhancing visual aesthetics and user experience.

## 7.13 React Virtualized

Website: <https://bvaughn.github.io/react-virtualized/>



React Virtualized offers a set of components optimized for efficiently rendering large lists and tabular data. It provides solutions for optimizing performance and memory consumption when dealing with extensive data sets.

# Chapter 8

## Important Documents

1. Link to Structure of Osdag on Cloud
2. Flow of Osdag on Cloud
3. Osdag on Cloud Repository

## References

1. [React](<https://reactjs.org/>)
2. [React Router DOM](<https://reactrouter.com/web/guides/quick-start>)
3. [Ant Design](<https://ant.design/>)
4. [Material-UI](<https://mui.com/>)
5. [jwt-decode](<https://www.npmjs.com/package/jwt-decode>)
7. [YouTube](<https://www.youtube.com/>)