Summer Fellowship Report

On

Osdag on Cloud

Submitted by

Atharva Ratnadeep Pingale

Under the guidance of

Prof. Sidhartha Ghosh
Department of Civil Engineering
IIT Bombay

and under the mentorship of

Mr. Nagesh Karmali
Project Manager, IIT Bombay

Mr. Danish Ansari
Project Software Engineer, IIT Bombay

August 28, 2023

Acknowledgment

I would like to thank FOSSEE for providing me a platform to work on something I am very interested in. I am thankful to everyone who thought of having and involved in selection process based on screening tasks. I am grateful to be a part of team which promotes open source software.

I thank all the Osdag on Cloud members, who are wonderful mentors and great team. I thank Dr. Kannan M. Moudgalya, Prof. Sidhartha Ghosh, Mr. Nagesh Karmali (Project Manager), Mr. Danish Ansari (Assistant Project Manager) and team, who made us feel welcome and planned all the tasks meticulously during this period. Your collective expertise and willingness to share your knowledge have significantly contributed to my professional growth and understanding of the field. I also extend my appreciation to my fellow interns and colleagues for creating an environment that fosters collaboration and learning. Your camaraderie and willingness to share insights and experiences have made this fellowship experience truly enriching.

As I reflect on this internship, I realize the immense impact it has had on my professional development and future prospects. The practical skills I've acquired and the connections I've made are invaluable assets that I will carry with me throughout my career. Once again, I am truly thankful for the opportunities I've been given and for the people who have played a role in making this fellowship a transformative and rewarding experience.

Contents

Chapter 1

Introduction

1.1 FOSSEE Summer Fellowship

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 About Osdag

Osdag is Free/Libre and Open Source Software being developed for design of steel structures. Its 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. As Osdag is currently funded by MHRD, Osdag team is developing software in such a way that it can be used by the students during their academics and to give them a better insight look in the subject.

1.3 Challenges with the Osdag software

In the Osdag software, there were some major issues which made it less user-friendly. Some major issues were - Installing the software packages on a system that already had some of the packages gave an installation error, Installation takes some space on the system, in order to use the Osdag software on has to install it usng hte installer and of the user didn't like it then had to remove it, this costed more effort for the user. Updating the Osdag software is difficult as compared to having a rolling update for a browser version. Some 3rd party antivirus softwares were identifying some installer files as 'containing malware'.

1.4 What is Osdag on Cloud?

Osdag on Cloud is a browser version of the Osdag software. Osdag on Cloud is developed to overcome the challenges which the classic Osdag software faced and provide a better version of it. It aims to provide hands-on design experience for college students, and thus creating tomorrow's designers familiar and confident with steel design, provide practical design experience for (new) practising engineers Work as a teaching tool helping college teachers. 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.

1.5 Who can use Osdag on Cloud?

Osdag on Cloud is created both for educational purpose and industry professionals. Osdag on Cloud is a FOSS and its team is developing software in such a way that it can be used by the students during their academics and to give them a better insight look in the subject. Osdag on Cloud can be used by anyone starting from novice to professionals.

Chapter 2

Installation and Setup of Osdag on Cloud

The installation steps are given below and also in the `documentation/installation.md` file of 'Osdag-web' Repository.

Before we install anything on our machine : Check if your Ubuntu machine USERNAME is present in the sudoers file or not.

```
$ sudo -l
```

If the output gives you the username of your Ubuntu machine, then skip this step and proceed to '2.1 Software Requirements' section.
If the output is : **Sorry, user USERNAME may not run sudo on VIRTUAL_MACHINE/UBUNTU_MACHINE**
Then, you will have to add your Ubuntu username into the sudoers file

- Obtain the USERNAME

  ```
  $ whoami
  ```

- Get into the root and open the file

  ```
  $ su root
  ```

  ```
  $ nano /etc/sudoers
  ```

- Then under the title **User Priviledge specification** , insert the below line :

  ```
  $ USERNAME ALL=(ALL:ALL) ALL
  ```

  Replace the text USERNAME with the username that you have obtained by the command whoami.
  Save the file (ctrl+o) and exit (ctrl+x)

- Exit the root terminal with ctrl+d

- Close the terminal and reopen it (ctrl+alt+t)

2.1 Software Requirements

1. Ubuntu LTS 20.04 / 22.04
2. Git : Install Git on Ubuntu. Open the terminal (ctrl+alt+t) and run the below commands:
   - Update the Repository

     ```
     $ sudo apt update
     ```

   - Install Git

     ```
     $ sudo apt install git
     ```

3. IDE : (**OPTIONAL**) Preferably VSCode. Install VSCode with :

   ```
   $ sudo snap install --classic code
   ```

4. Node v16.20.0 : Install Node from NVM by running these commands in the Terminal
   - Install curl before with the command :

     ```
     $ sudo apt install curl
     ```

   - Node installation commands :

     ```
     $ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/
     install.sh
     ```

     ```
     $ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/
     install.sh | bash
     ```

     ```
     $ source  /.bashrc
     ```

     ```
     $ nvm install v16.20.0
     ```

5. Postgres : Install Postgres by running the following commands :

   ```
   $ sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt
   ```

   ```
   $ (lsb_release - cs)- pgdg main" > /etc/apt/sources.list.d/pgdg.list'
   ```

   ```
   $ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc
   |sudo apt-key add -
   ```

```
$ sudo apt-get update
```

```
$ sudo apt-get -y install postgresql
```

6. Freecad : Install freecad with the following commands :

   - Move to the root directory of the Ubuntu machine

     ```
     $ cd /
     ```

     ```
     $ sudo apt-get update
     ```

   - Install snapd package manager

     ```
     $ sudo apt-get install snapd
     ```

   - Install freecad from snap

     ```
     $ sudo snap install freecad
     ```

2.2 Installation steps

The Osdag on Cloud project uses 'Conda' environment which contains all the dependencies. To first download these, visit the link : Osdag Download and download the **Installer** [**Release: 2021-02-15**] for Ubuntu. Install both the Installer - Linux and the Installation instructions for Ubuntu files :

1. Follow the instructions in the **Installation instructions** file that you have downloaded. After completing that return back to this installation guide.

2. Install texlive-latex-extra packages. Open the terminal (ctrl+alt+t) and run the following command :

   ```
   $ sudo apt-get update
   ```

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

3. Now you have successfully installed Osdag, texLive and miniconda on your machine. Navigate to 'Desktop'

4. The next step is to clone the Osdag-Web repository on github. There are 2 ways to download the repository :

   - If you already have **git** installed on your machine, then open a new terminal in Desktop (ctrl+alt+t) and run the following command :

     ```
     $ git clone https://github.com/SurajBhosale003/Osdag-web.git
     ```

- If you don't have git installed, then visit the Repository link : https://github.com/SurajBhosale003/Osdag-web , click on **Code** tab and download the zip file After downloading the zip file, open the terminal

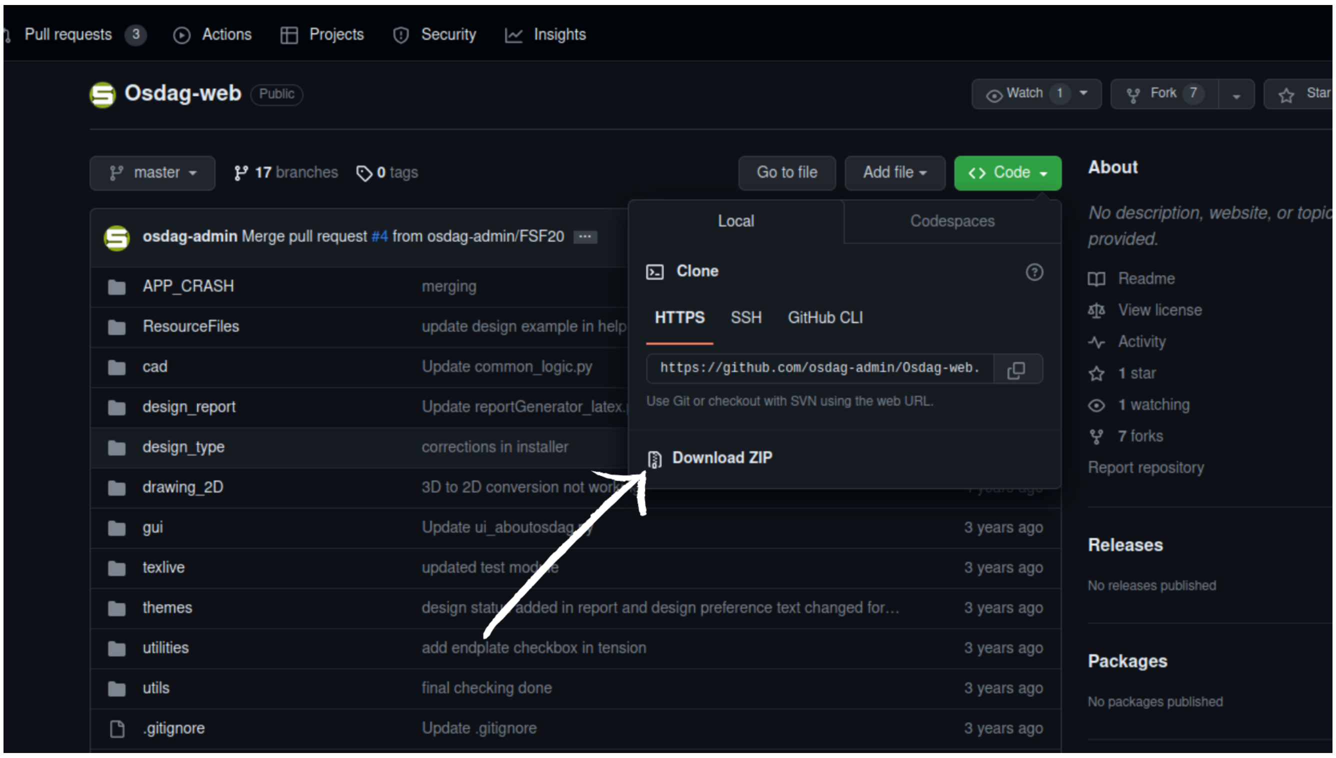


Figure 2.1: Osdag zip file download

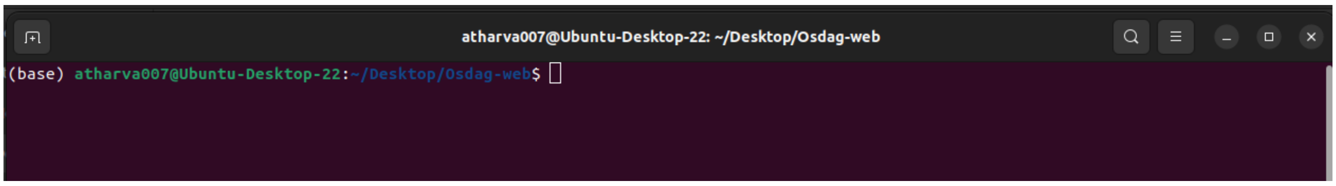
and unzip the file :

```
$ tar -xvf Osdag-Web-master.zip
```

Move the unzipped Osdag-Web-master folder to **Desktop** or wherever you want and rename it to **Osdag-web**

5. Open the Osdag-web folder and open a new terminal there. Make sure you have the conda environment activated. You can know this if there is (base) written at the start of the terminal line. If you don't see this, activate the conda environment using :

```
$ conda activate
```



```
(base) atharva007@Ubuntu-Desktop-22:~/Desktop/Osdag-web$
```

Figure 2.2: Root directory

6. Create Database and Role in Postgres and Configure it, open the Terminal (ctrl+alt+t):

   - Enter into the Postgres Terminal

     ```
     $ sudo -u postgres psql
     ```

   - Create a new role

     ```
     $ CREATE ROLE osdagdeveloper PASSWORD 'password' SUPERUSER
     CREATEDB CREATEROLE INHERIT REPLICATION LOGIN;
     ```

   - Create the database

     ```
     $ CREATE DATABASE "postgres_Intg_osdag" WITH OWNER osdagdeveloper;
     ```

   - Exit from the Postgres terminal

     ```
     $ \q
     ```

7. Open a terminal (ctrl+alt+t) and follow the below steps

   - Enter into the Osdag-web folder which you have cloned

     ```
     $ cd Desktop/Osdag-web
     ```

   - Switch to develop branch

     ```
     $ git checkout develop
     ```

   - Install requirements.txt packages

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

   - Configure the Postgres database

     ```
     $ python populate_database.py
     ```

     ```
     $ python update_sequences.py
     ```

     ```
     $ python manage.py migrate
     ```

   - Install the node dependencies

     ```
     $ cd osdagclient
     ```

     ```
     $ npm install
     ```

     ```
     $ cd ..
     ```

- Start the Django server

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

- Open another terminal, navigate to root of Osdag-web folder and run the following commands :

```
$ cd osdagclient
```

```
$ npm run dev
```

8. Now your Server and Client are running. Navigate to http://localhost:5173/ on your Browser. Now you can use the application.

Chapter 3

My Fellowship work

Since the commencement of my Fellowship I have contributed to the project Osdag on Cloud. The initial few days of the Fellowship were spent in understanding the process flow and working of the Osdag Software. I have then developed various features and functionalities for Osdag on Cloud. Below are my contributions in brief :

1. **Input Values API (Backend) :** The Input values API is a singular APIView which handles the communication of all the input values mentioned in the input dock like the - material, connectivity type, property class, thickness list, bolt diameter, axial force etc. The input values are read from the Database and are sent to the client in the response.

2. **Create Design Report API (Backend) :** The Create Design Report APIView consists of 3 API Views which handle : storing the company logo in the server, creating the LaTEX design report and obtaining (downloading) the Design report.

3. **User Authentication, Authorization and activities (Backend) :** I have fully developed the User activities part in the project. I have developed APIViews in the backend which perform operations like - user login, user signup, user logout, setting the refresh token, JWT token creation, verifying mail, forgot password feature, saving input files, sending all the input files associated to the user to the client.

4. **Module State :** The Module state file which is fully developed by me except the design preferences calls. This file is responsible for making API calls when a user is working on a selected module (ex - Fin Plate Connection). All the thunks in the file perform operations like - obtaining the input values, setting the session, API call for CAD design, generating the design report and its related APIs.

5. **User State :** I have fully developed all the Thunks in the UserState. All the thunks in the file are responsible for user login, user signup, verifying email, forgot password feature, saving the input value file, obtaining the input values files of a user, generating and setting the 'Access' and 'Refresh' token using JWT.

6. **Postgres Database :** I have configured the Postgres Database for the project, developed the Django models, the serializers for the models, the shell scripts for populating the Postgres database and updating sequences, developed the Design model, have developed the UserAccount model, developed the installation documentation for the project (`documentation/installation.md`)
7. **Architecture :** I have designed the Osdag on Cloud Architecture. This is a blueprint of the application. It defines how components of a software system are assembled, their relationship and communication between them.

Chapter 4

Input Values - Backend

The inputData_view.py file in the Django server file system is an API file which communicates between the client (Browser) and the server. This file is located in relative directory : `osdag/web_api/inputData_view.py`. This file contains the InputData APIView class which handles GET requests only. When an API call is made to obtain the input values (a GET request), the get() method of the InputData APIView class is invoked. Depending on the type of request, the program control is navigated using if-else statements. There are 3 types of connectivity data which this APIView serves. These are :

1. Column Flange-Beam-Web
2. Column Web-Beam-Web
3. Beam-Beam

Before the request is served, the API checks if the cookie in the request is present or not. It checks for the cookie named `fin_plate_connection_session`. The request gets served only if this cookie is present. This cookie is generated and set by the `session_api.py` file inside the same directory as `inputData_view.py`

4.1 Column Flange-Beam-Web and Column-Web-Beam-Web

For both Column Flange-Beam Web and Column-Web-Beam-Web types of connectivity, the response is served in the same way. These requests require 3 values :

1. 'Designation' column from 'Columns' table
2. 'Designation' column from 'Beams' table
3. 'Grade' column from 'Material' table

The data for these columns is obtained from the Postgres database and sent to the client as a JSON DRF Response.

4.2 Beam-Beam

For Beam-Beam type of connectivity, the response is served differently. This request requires 2 values :

1. 'Designation' column from 'Beams' table
2. 'Grade' column from 'Material' table

The data for these columns is obtained from the Postgres database and sent to the client as a JSON DRF Response.

4.3 Customized inputs

There are 3 input values that can be set to 'customized' on the Browser. These are :

1. Propertyclass The API will send all the values from 'boltFyFuList' table as a python list embedded in a JSON object. This response is sent as DRF Response.
2. Thickness When the thickness property is set to customized, the API will send a 'PLATE_THICKNESS_SAIL' list embedded in an object. This object will be sent as a JSON DRF Response.
3. BoltDiameter When the BoltDiameter property is set to customized, all the records of the column 'Bolt_diameter' from the table 'Bolt' is obtained from the Postgres database. This list is sent to the client as a JSON DRF Response.

Chapter 5

Create Design Report - Backend

5.1 API for Company Logo

The company logo feature is present when the user hits the 'Create Design' button and a popup appears. The user then uploads a company logo in the input field. When an image is uploaded, before pressing the 'Ok' button to start report generation, an API is called which stores the company logo in the server file system. This image will be used to display in the report. An APIView called **CompanyLogoView** in `osdag/web_api/design_report_csv_view.py` is called. This APIView accepts only POST request. When the program control enters this APIView, it first checks for a cookie_id named `fin_plate_connection_session`. The file is obtained from the request, a unique filename is generated using uuid4. It then checks if the directory `file_storage/company_logo` exists or not, if the directory does not exist, then the directory is created. The company logo image file is then stored in the `file_storage/company_logo/` directory. After successful write operation, a response is sent back to the client with the 'logofullPath' which contains the absolute path of the image.

5.2 Generating the Design Report

When the user sends all the details for generating the design report, the APIView **CreateDesignReport** is called. This APIView accepts POST request only. The 'metadata' from the request body is obtained. Then the program control checks for the cookie_id called `fin_plate_connection_session`. Then the input_values which was saved in 'Design' model is obtained, along with the design_status and logs from the 'Design' model. Then it checks if the metadata is None, then it creates an object for the dummy

5.3 Obtaining the Design Report PDF

After the design report is created, it gets saved in `file_storage/design_reports` directory. After which another request is made to the server to obtain the design report PDF file. This task is handled by the GetPDF APIView which is present in `osdag/web_api/design_report_csv_view.py` file. This API View obtains the report_id from the request data (POST request). The filename is the `report_id.pdf` file which is searched in that directory. The latex files is converted to the PDF file with the same filename as the report_id and is sent back to the client.

Chapter 6

User Authentication, Authorization and activities - Backend

6.1 User Signup

When the user on the browser creates a new account by submitting the username, email and password, a request is made to the **Signup APIView** in the `osdag/web_api/user_view.py file`. This APIView obtains the username, email, password and the isGuest attribute which defines if the user is a guest or not. All the request.data is passed to the UserAccount_serializer. If the serializer is valid, then a new user account is created in the Django's default User model. Creating this user in the Django's by default User model will be useful in providing authentication, defining permission_classes and more advanced things that Django provides. So, the user account gets created in the UserAccount model and the Django's User model.

6.2 User Login

When a user who has an account logs provides the username and password to the Browser. A request is made to the **LoginView APIView**. This APIView obtains the isGuest attribute, if the isGuest attribute is false, then it proceeds to authenticate the user with the useranme and password. To authenticate the user, the record with both the matching values username and password is obtained. If found, then the user gets logged in, else receives an error message. On successful login, the length of the list of all the inputValueFiles associated with that user is sent to the client along with the user's email.

6.3 Guest Mode

While logging in, if the user clicks on the Guest mode in the Browser, a request is made to the **LoginView APIView**. This time, the isGuest attribute is set to true. For guest mode, a dummy user is created in Django's User model with a dummy username, email and password. Then, a successful login message is sent back to the client.

6.4 Email Verification

When the user is on the login page, clicks Forget password, provides email and hits the 'Get OTP' button, a request is made to the **CheckEmailView APIView** in the `osdag/web_api/user_view.py` file. The email is obtained from the request data (POST request), it then checks if the email is present in the database or not. If it is present, then it proceeds with generating a random OTP. The email and the OTP is passed to the **send_mail()** function which is located in mailing.py file in osdag_web directory. The send_mail function will obtain the HOST, PORT, FROM_EMAIL and PASSWORD from the `osdag_web/utils.py` file. The credentials for osdag on cloud's email and password can be set here. Then an email is sent to the user's email using the SMTP protocol. After which, the program control returns back to CheckEmailView APIView and a response with the OTP is sent back to the client.

6.5 Forget Password

After the OTP is sent to the user's account by **CheckEmailView APIView**, the user will provide the new password in the browser. After hitting save, an API call will be made to the **ForgetPasswordView APIView**. In this APIView, the password and email is obtained from the request data. The user's email is searched in the Django User model. If a user is found in the User model, then the password is updated for the user. Then a user with the email is searched in the UserAccount model. If the user is found, then the password for that user gets updated as well. After this, a success message is sent back to the client.

6.6 API for Saving the input values

When the user us working on the module and hits the 'Save Input' button in the drop-down in the File top-bar. An API call is made to the **SaveInputFileView APIView**. This APIView obtains the content and email from the request data (POST request). The user with the email is searhed in the UserAccount model. Upon finding the user, the userObject is obtained. Then the length of allInputValueFiles array is obtained and the length+1 value is added to the filename string. Currently the module is fin_plate_connection, so the filename will have the name of the `module_fileIndex` osi file. The contents obtained from the request is written in the file. The file is stored in the `file_storage/input_values_files directory`. Upon successfully creating and writing the file. The absolute path of that file is appended in the **allInputValueFiles** array associated to that particular user. The updated userObject is saved. After this, the response with allInputValueFilesLength and the filename is sent back to the client.

6.7 API for obtaining all the input files of a user

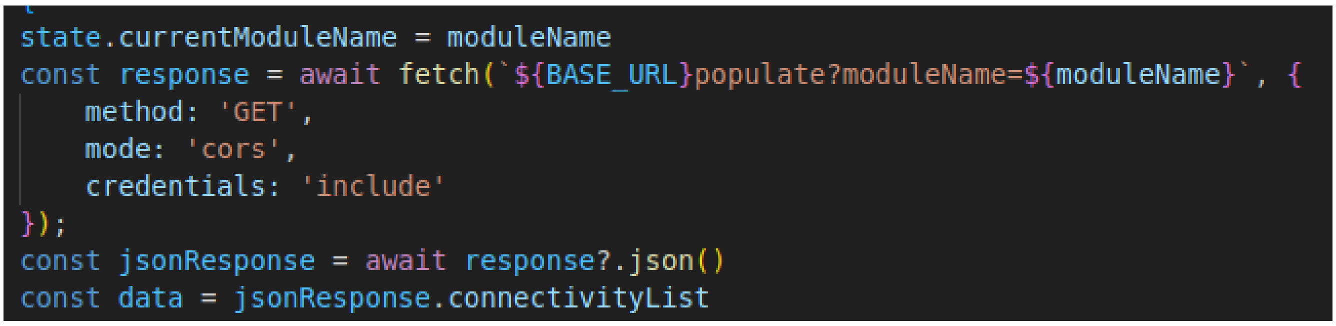
When a logged in User goes to the My Account page. An API call is made to the **ObtainInputFileView APIView**. This APIView obains the email and the fileIndex from the request data. The email is searched in the UserAccount model. If found, a userObject is obtained from which the file with the **allInputValueFile**[**fileIndex**] from the array is obtained. This filename is the absolute path which we had appended while saving the input in the **SaveInputFileView APIView**. The file with the obtained absolute path is found in the Server's File system. Upon finding the file, it is sent to the client as a FileResponse. Even tough there can be multiple files created by the user, this APIView will serve files one at a time to the client. This means if there are 10 files, then 10 API calls will be made to obtain 10 files.

Chapter 7

ModuleState Thunks and Reducer

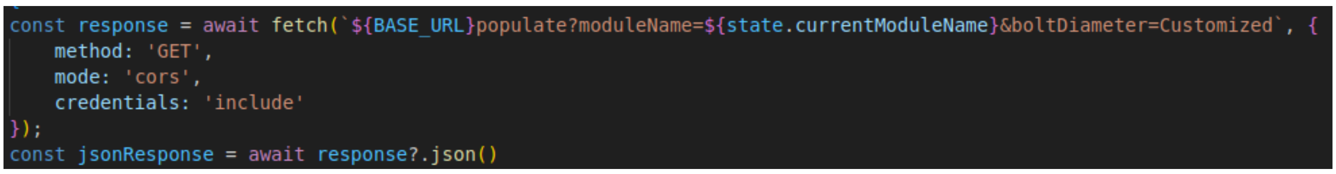
7.1 Input values fetching

There are many fields in the input dock. These fields are obtained by individual requests. Like - the connectivityList (Column Web-Beam Web, Beam-Beam etc) is obtained by the **getConnectivityList** where the module name is passed to the API endpoint and the connectivity List is obtained, all the boltDiameter values are obtained from **getBoltDiameterList**, all thickness values are obtained from **getThicknessList**, all propertyClass values are obtained from **getPropertyClassList**, all material values are obtained from **getColumnBeamMaterialList**.



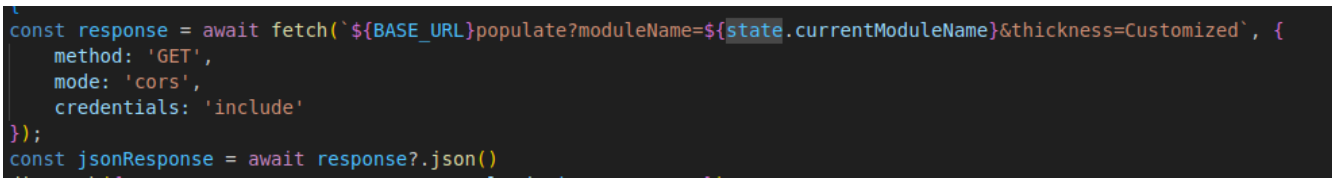
```
state.currentModuleName = moduleName
const response = await fetch(`${BASE_URL}populate?moduleName=${moduleName}`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include'
});
const jsonResponse = await response?.json()
const data = jsonResponse.connectivityList
```

Figure 7.1: getConnectivityList Thunk



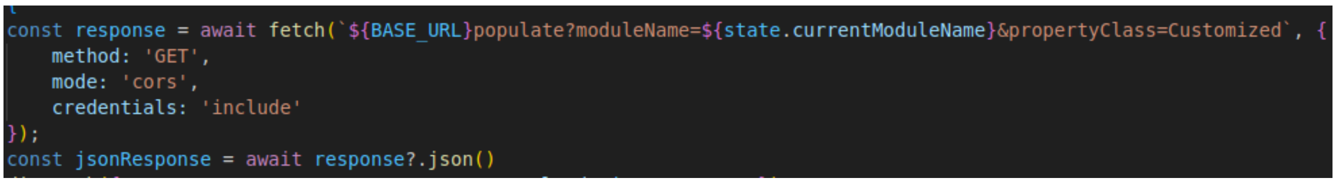
```
const response = await fetch(`${BASE_URL}populate?moduleName=${state.currentModuleName}&boltDiameter=Customized`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include'
});
const jsonResponse = await response?.json()
```

Figure 7.2: getBoltDiameter Thunk



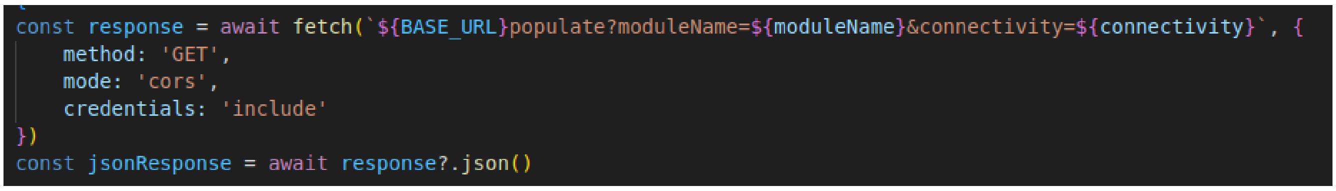
```
const response = await fetch(`${BASE_URL}populate?moduleName=${state.currentModuleName}&thickness=Customized`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include'
});
const jsonResponse = await response?.json()
```

Figure 7.3: getThicknessList Thunk



```
const response = await fetch(`${BASE_URL}populate?moduleName=${state.currentModuleName}&propertyClass=Customized`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include'
});
const jsonResponse = await response?.json()
```

Figure 7.4: getPropertyClassList Thunk



```
const response = await fetch(`${BASE_URL}populate?moduleName=${moduleName}&connectivity=${connectivity}`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include'
})
const jsonResponse = await response?.json()
```

Figure 7.5: getColumnBeamMaterialList Thunk

7.2 Design Report fetching

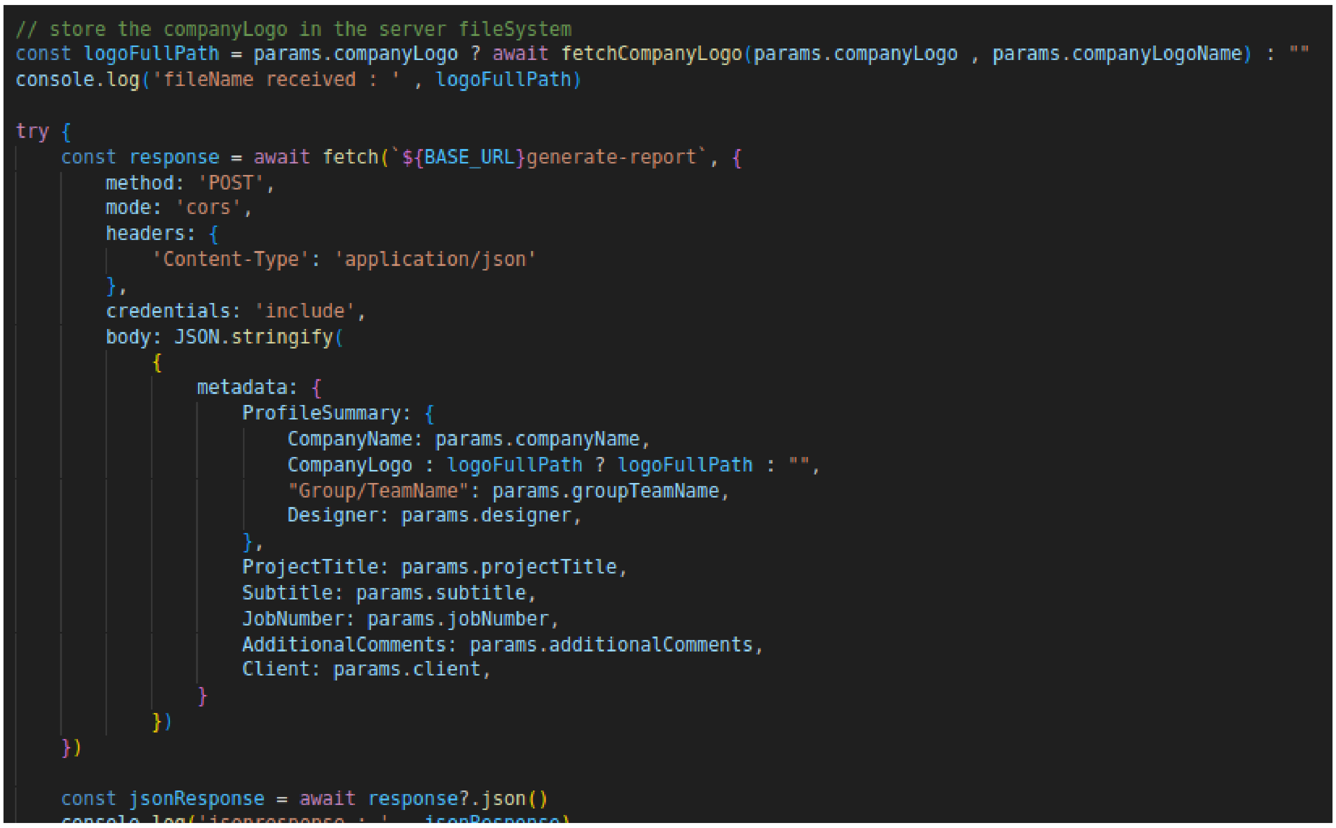
When the create Design Report button is clicked, the **createDesignReport Thunk** thunk is called which sends the request to the server. In the request body, the companyLogo is required, so another request to the server is made by the **companyLogo Thunk**. If the companyLogo is provided, then the companyLogo is sent, else nothing.

7.3 CAD model fetching

To obtain the CAD model .obj file, the request is made to the server by the **createCADModel Thunk**. This thunk will not have anything in the response as the CAD .obj file is stored in the `osdagclient/public` folder by the server.

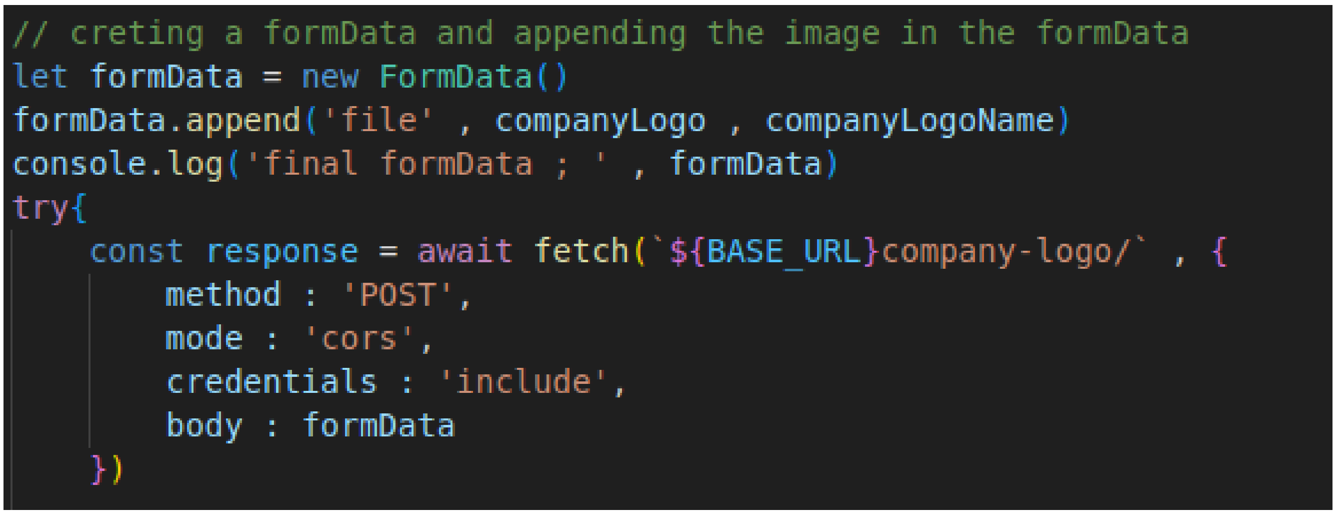
7.4 Design Report PDF fetching

To obtain the design report PDF file, after the **createDesignReport Thunk** creates the design report, another thunk makes a call to the server to obtain the PDF file. This is done by the **getPDF Thunk**.



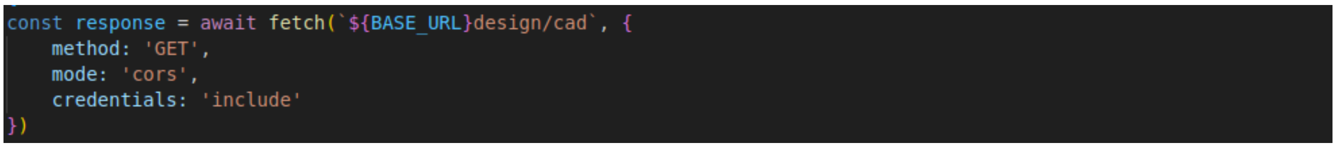
```
// store the companyLogo in the server fileSystem
const logoFullPath = params.companyLogo ? await fetchCompanyLogo(params.companyLogo , params.companyLogoName) : ""
console.log('fileName received : ' , logoFullPath)

try {
    const response = await fetch(`${BASE_URL}generate-report`, {
        method: 'POST',
        mode: 'cors',
        headers: {
            'Content-Type': 'application/json'
        },
        credentials: 'include',
        body: JSON.stringify(
            {
                metadata: {
                    ProfileSummary: {
                        CompanyName: params.companyName,
                        CompanyLogo : logoFullPath ? logoFullPath : "",
                        "Group/TeamName": params.groupTeamName,
                        Designer: params.designer,
                    },
                    ProjectTitle: params.projectTitle,
                    Subtitle: params.subtitle,
                    JobNumber: params.jobNumber,
                    AdditionalComments: params.additionalComments,
                    Client: params.client,
                }
            })
    })

    const jsonResponse = await response?.json()
    console.log('jsonresponse : ' , jsonResponse)
```

Figure 7.6: createDesignReport Thunk



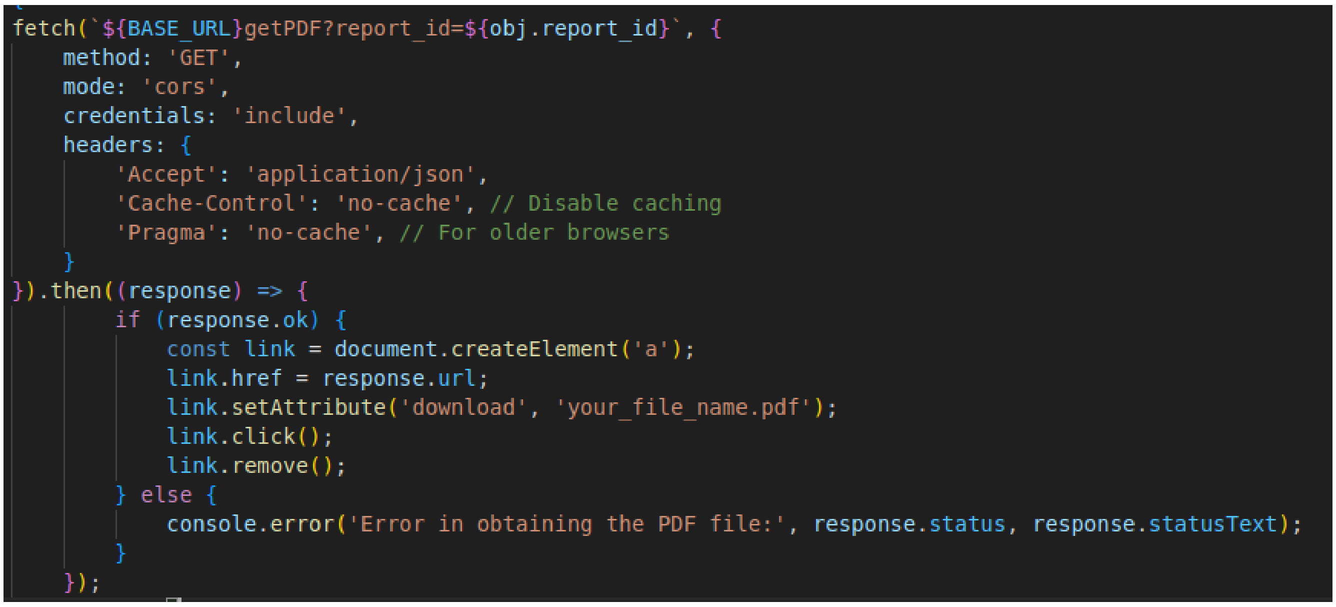
```
// creting a formData and appending the image in the formData
let formData = new FormData()
formData.append('file' , companyLogo , companyLogoName)
console.log('final formData ; ' , formData)
try{
    const response = await fetch(`${BASE_URL}company-logo/` , {
        method : 'POST',
        mode : 'cors',
        credentials : 'include',
        body : formData
    })
```

Figure 7.7: CompanyLogo Thunk



```
const response = await fetch(`${BASE_URL}design/cad`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include'
})
```

Figure 7.8: createCADModel Thunk

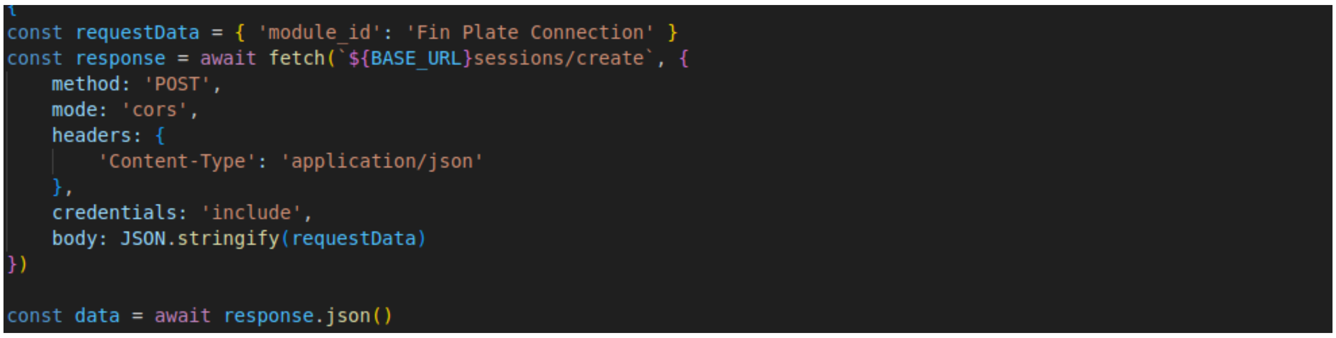


```
fetch(`${BASE_URL}getPDF?report_id=${obj.report_id}`, {
    method: 'GET',
    mode: 'cors',
    credentials: 'include',
    headers: {
        'Accept': 'application/json',
        'Cache-Control': 'no-cache', // Disable caching
        'Pragma': 'no-cache', // For older browsers
    }
}).then((response) => {
        if (response.ok) {
            const link = document.createElement('a');
            link.href = response.url;
            link.setAttribute('download', 'your_file_name.pdf');
            link.click();
            link.remove();
        } else {
            console.error('Error in obtaining the PDF file:', response.status, response.statusText);
        }
    });
```

Figure 7.9: getPDF Thunk

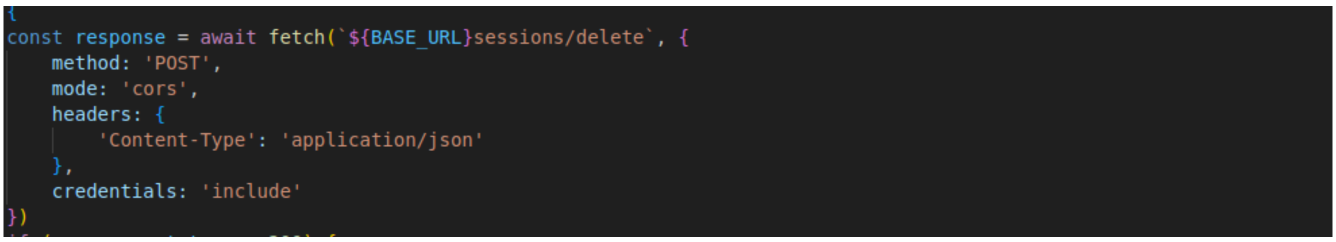
7.5 Create and Delete Design Session

To create a session, a cookie with a `key=fin_plate_connection_session` is set by the server. To make this happen, a request is made by the **createSession Thunk** which makes the call to create the session. And similarly, to delete the current session, the **deleteSession Thunk** makes a call to the server to delete the current session.



```
const requestData = { 'module_id': 'Fin Plate Connection' }
const response = await fetch(`${BASE_URL}sessions/create`, {
    method: 'POST',
    mode: 'cors',
    headers: {
        'Content-Type': 'application/json'
    },
    credentials: 'include',
    body: JSON.stringify(requestData)
})

const data = await response.json()
```

Figure 7.10: createSession Thunk

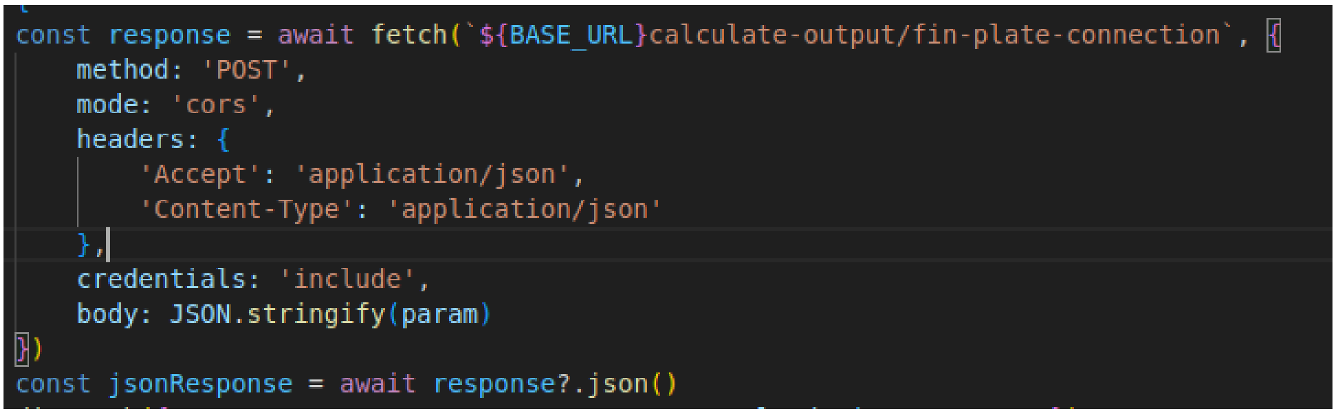


```
{
const response = await fetch(`${BASE_URL}sessions/delete`, {
    method: 'POST',
    mode: 'cors',
    headers: {
        'Content-Type': 'application/json'
    },
    credentials: 'include'
})
```

Figure 7.11: deleteSession Thunk

7.6 Create Design

When the user hits the 'Design' button. A request is made by the **createDesign Thunk**. This thunk will obtain all the input values which are passed to it via parameters and will make a request to the server to generate the output.



```
const response = await fetch(`${BASE_URL}calculate-output/fin-plate-connection`, {
    method: 'POST',
    mode: 'cors',
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
    credentials: 'include',
    body: JSON.stringify(param)
})
const jsonResponse = await response?.json()
```

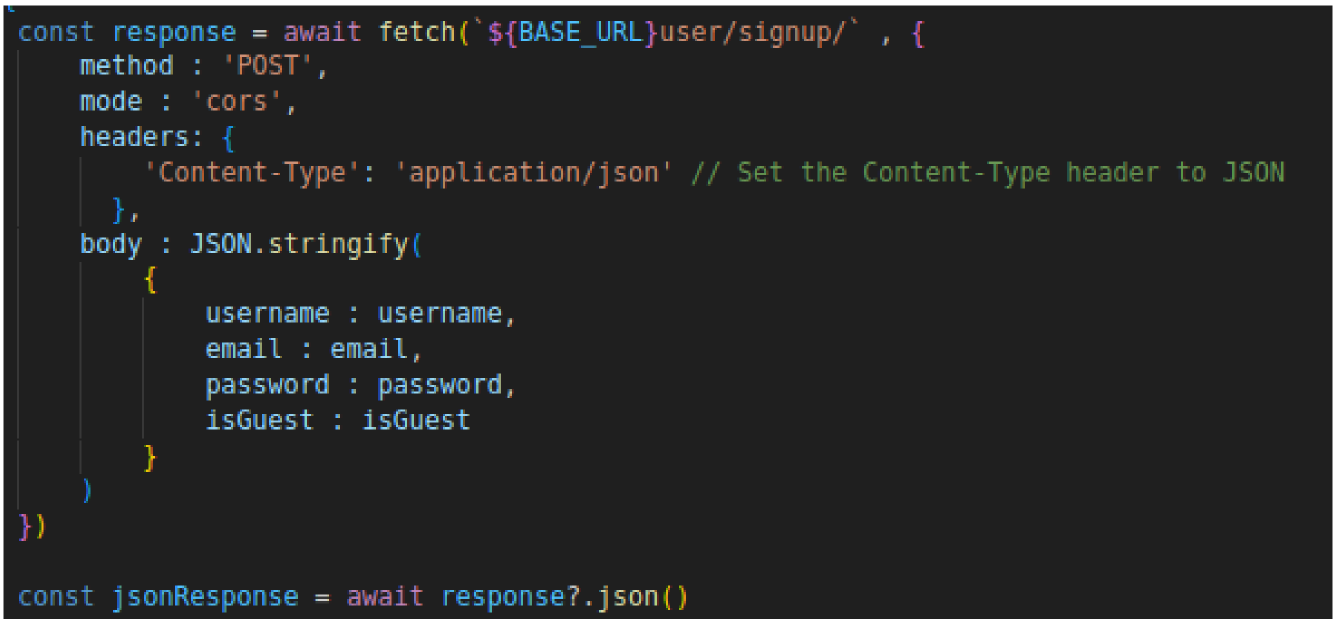
Figure 7.12: createDesign Thunk

Chapter 8

UserState Thunks and Reducer

8.1 User Signup fetching

When a user creates an account. The credentials - username, email and password are sent to the **userSignup Thunk** which sends a request to the server.



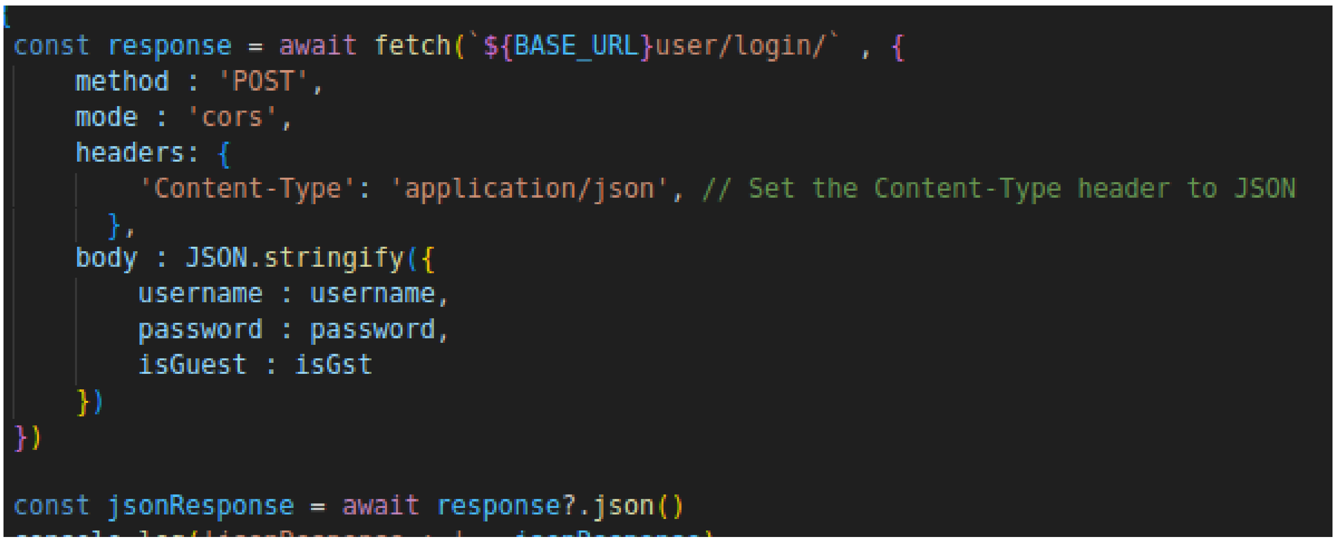
```
const response = await fetch(`${BASE_URL}user/signup/` , {
    method : 'POST',
    mode : 'cors',
    headers: {
        'Content-Type': 'application/json' // Set the Content-Type header to JSON
      },
    body : JSON.stringify(
        {
            username : username,
            email : email,
            password : password,
            isGuest : isGuest
        }
    )
})

const jsonResponse = await response?.json()
```

Figure 8.1: userSignup Thunk

8.2 User Login fetching

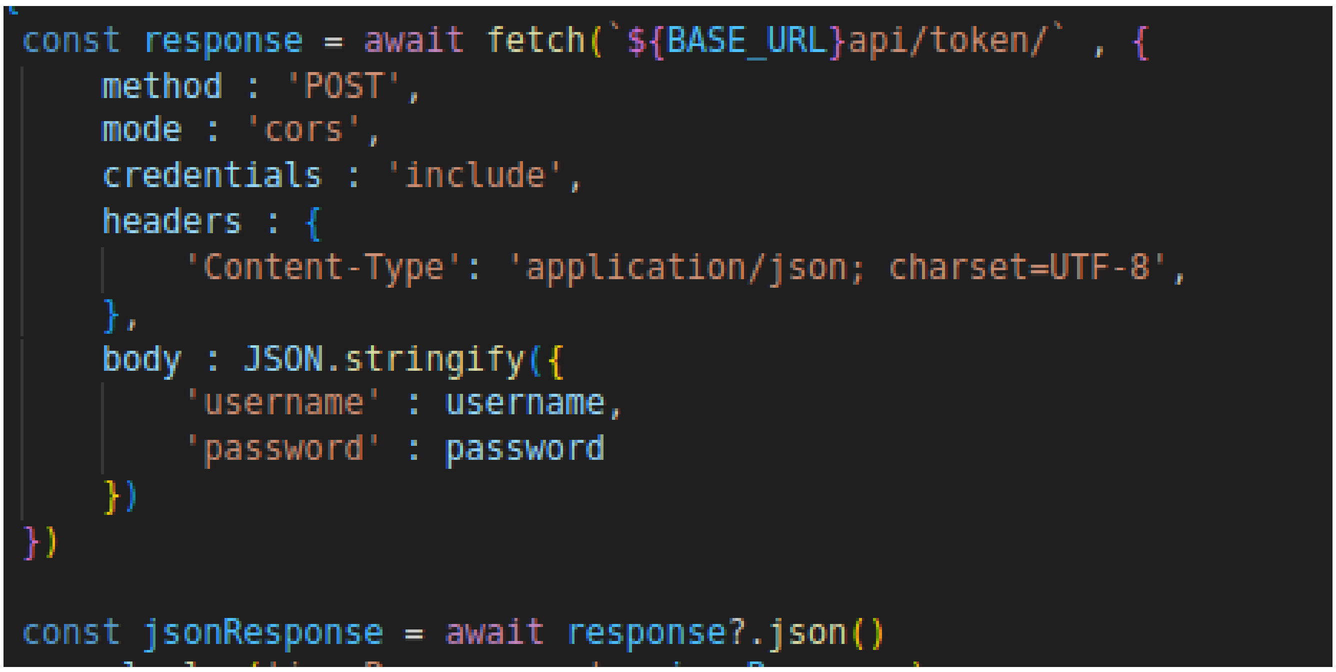
When a user logs into an existing account, the credentials username, password are passed to the **userLogin Thunk**. Here, a request is made to the server for authentication followed by the calling of **createJWTToken Thunk** which is responsible to generate 'refresh_token' and 'access_token' based upon the username and password required for authorization. In these 2 API calls, some data is obtained as a response which is stored in the localStorage. The refresh_token is however stored into the

Cookies. Do set the refresh_token in the Cookies, after the tokens have been created by the createJWTToken Thunk, another thunk is called to set the refresh token to the cookie. This is done by the **setRefreshTokenCookie Thunk**.



```
const response = await fetch(`${BASE_URL}user/login/` , {
    method : 'POST',
    mode : 'cors',
    headers: {
        'Content-Type': 'application/json', // Set the Content-Type header to JSON
    },
    body : JSON.stringify({
        username : username,
        password : password,
        isGuest : isGst
    })
})

const jsonResponse = await response?.json()
```

Figure 8.2: userLogin Thunk

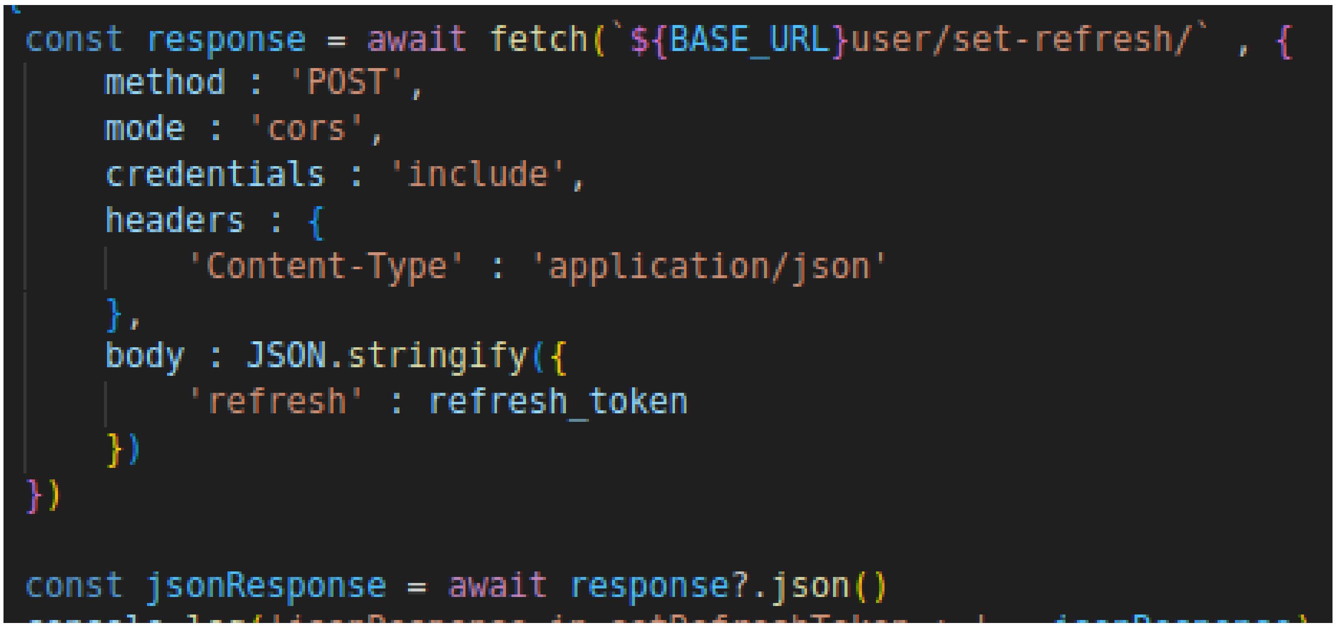


```
const response = await fetch(`${BASE_URL}api/token/` , {
    method : 'POST',
    mode : 'cors',
    credentials : 'include',
    headers : {
        'Content-Type': 'application/json; charset=UTF-8',
    },
    body : JSON.stringify({
        'username' : username,
        'password' : password
    })
})

const jsonResponse = await response?.json()
```

Figure 8.3: createJWTToken Thunk

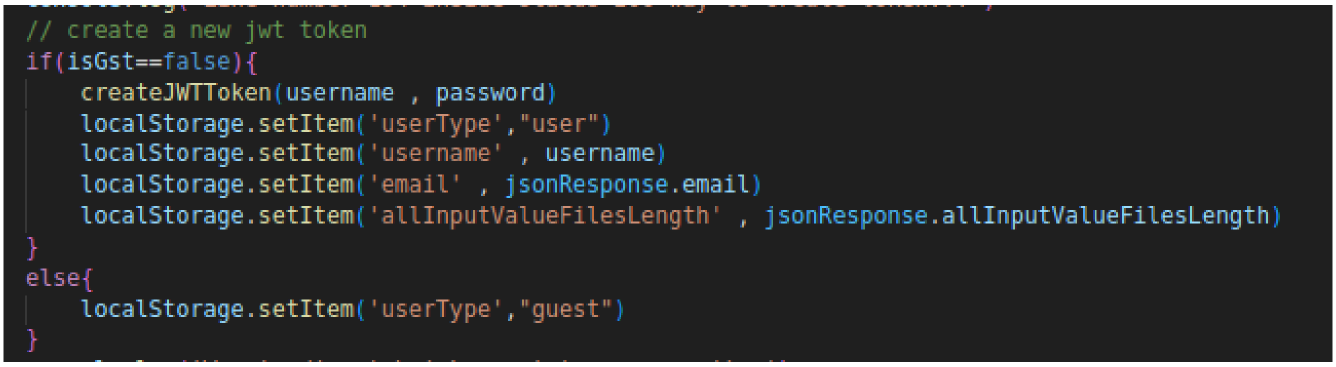
8.3 Email Verification

For email verification, **verifyEmail Thunk** is called, which makes a request to the server, providing the email in the request. The response obtained is the OTP which is used by the client to validate the user's input OTP.



```
const response = await fetch(`${BASE_URL}user/set-refresh/` , {
    method : 'POST',
    mode : 'cors',
    credentials : 'include',
    headers : {
        'Content-Type' : 'application/json'
    },
    body : JSON.stringify({
        'refresh' : refresh_token
    })
})

const jsonResponse = await response?.json()
```

Figure 8.4: setRefreshTokenCookie Thunk



```
// create a new jwt token
if(isGst==false){
    createJWTToken(username , password)
    localStorage.setItem('userType',"user")
    localStorage.setItem('username' , username)
    localStorage.setItem('email' , jsonResponse.email)
    localStorage.setItem('allInputValueFilesLength' , jsonResponse.allInputValueFilesLength)
}
else{
    localStorage.setItem('userType',"guest")
}
```

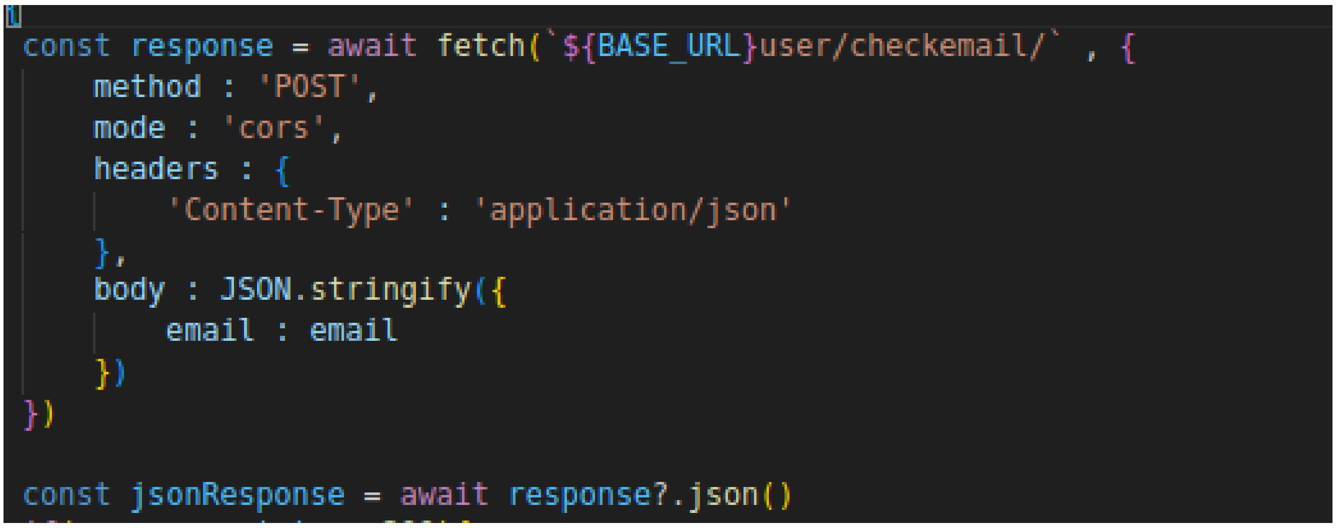
Figure 8.5: LocalStorage set values

8.4 Forget Password

The forget Password feature calls the **ForgetPassowrd Thunk**, which makes a request. It sends email and the new password to the server.

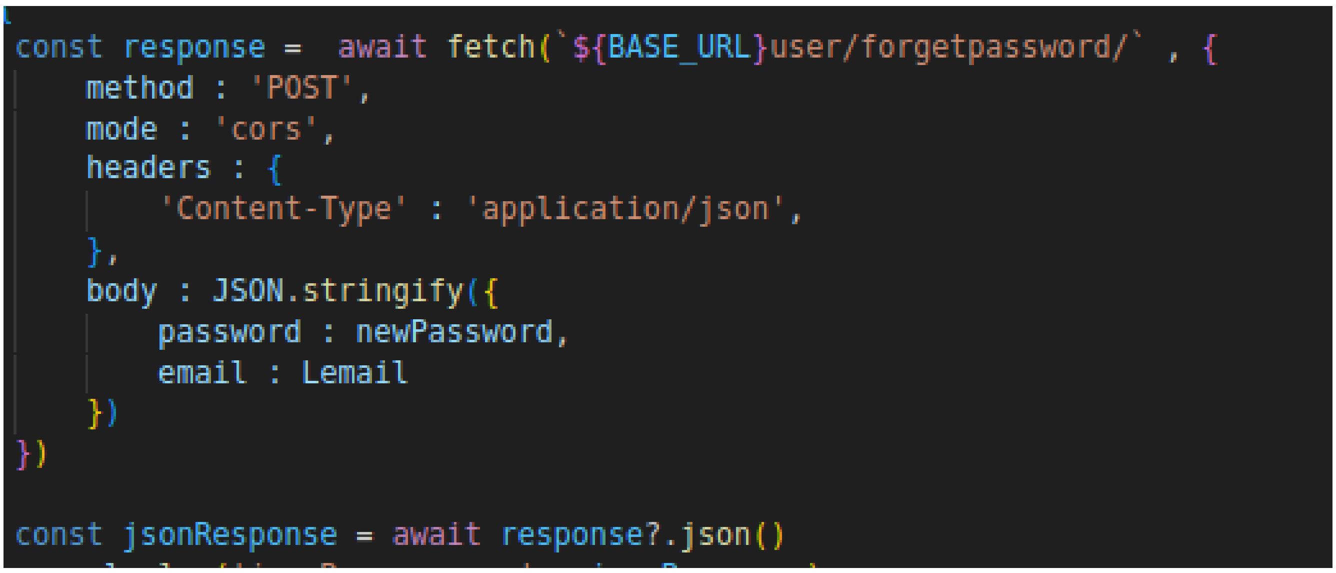
8.5 Saving the Input Files

To save the input file of the current module. When the user clicks on the Save Input button in the dropdown. A **SaveInputValues Thunk** makes a call to the back-end providing the all the input values in its request.body field. The request is a POST request with credentials included. The email is also sent in the request.body to help the Django server to create and name the file. Upon successful creation, a status OK is received which then triggers the dispatch() function which updates the redux variables. One of the variables is the save input message which is displayed on the top bar on the browser - that the input file has been saved as FILENAME. Upon



```
const response = await fetch(`${BASE_URL}user/checkemail/` , {
    method : 'POST',
    mode : 'cors',
    headers : {
        'Content-Type' : 'application/json'
    },
    body : JSON.stringify({
        email : email
    })
})

const jsonResponse = await response?.json()
```

Figure 8.6: verifyEmail Thunk



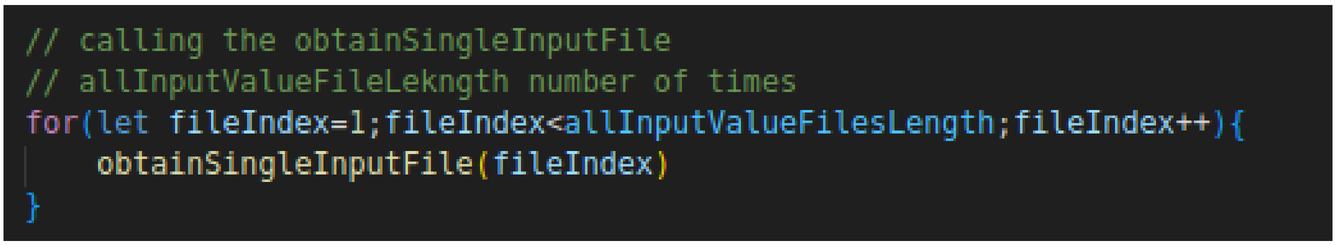
```
const response =  await fetch(`${BASE_URL}user/forgetpassword/` , {
    method : 'POST',
    mode : 'cors',
    headers : {
        'Content-Type' : 'application/json',
    },
    body : JSON.stringify({
        password : newPassword,
        email : Lemail
    })
})

const jsonResponse = await response?.json()
```

Figure 8.7: ForgetPassword Thunk

saving the input values of the module again, the contents of the file is sent to the Django server which then creates a new file and a new message is displayed on the top-bar in the browser with the new FILENAME.

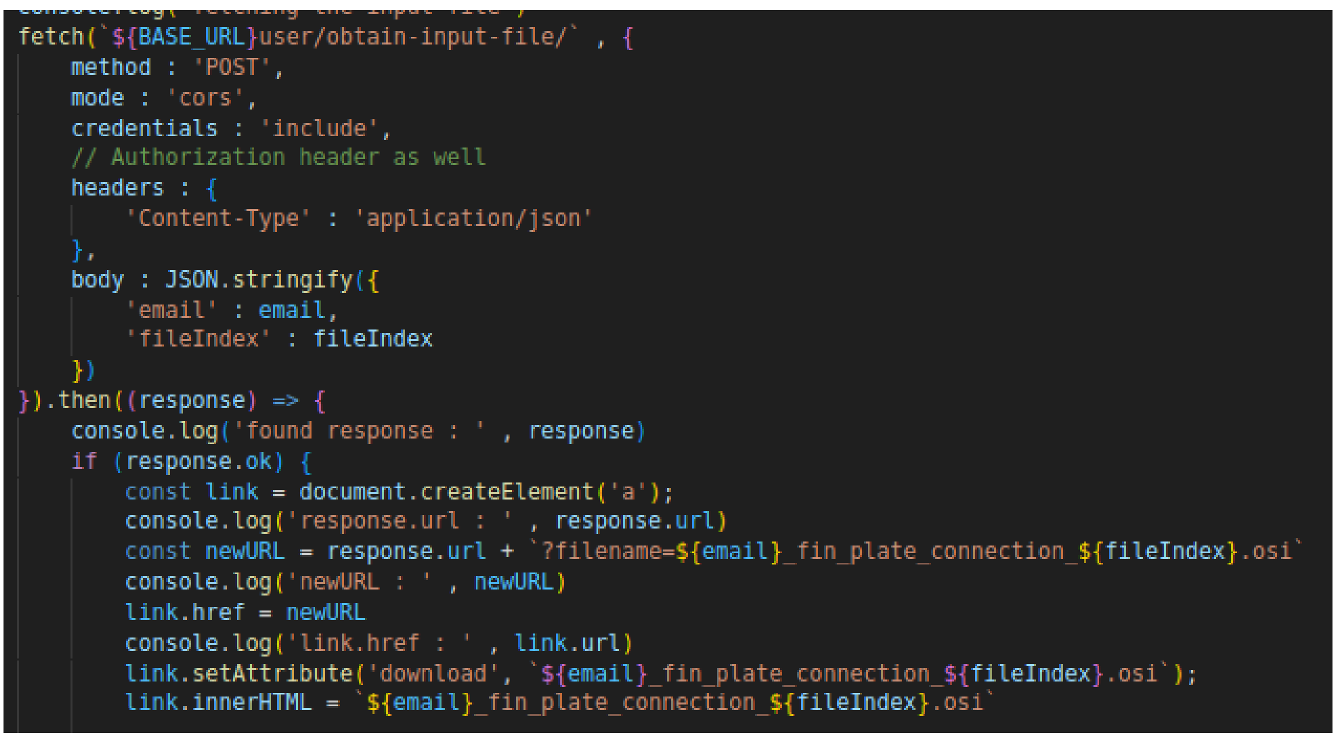
8.6 Obtaining the input files

To obtain multiple input files associated to the user, the **obtainAllInputValue-Files Thunk** is called which obtains the number of inputFiles associated to the user. This number of inputFiles is obtained from the localStorage which we had obtained when the user logged in or when the user saves the input. This value (number of inputFiles) gets updated in the localStorage whenever the user saves the input file. Now that the number of input files is known, a for loop is executed which calls the **obtainSingleInputFile Thunk**. This thunk will obtain one file at a time. in this way, after the for loop is executed, all the input files are obtained. When the user clicks on the download button of any .osi input file, the download link will sent (a GET request) to the server which will provide the file and download it in the browser.



```
// calling the obtainSingleInputFile
// allInputValueFileLekngth number of times
for(let fileIndex=1;fileIndex<allInputValueFilesLength;fileIndex++){
    obtainSingleInputFile(fileIndex)
}
```

Figure 8.8: obtainAllInputValueFiles Thunk



```
fetch(`${BASE_URL}user/obtain-input-file/` , {
    method : 'POST',
    mode : 'cors',
    credentials : 'include',
    // Authorization header as well
    headers : {
        'Content-Type' : 'application/json'
    },
    body : JSON.stringify({
        'email' : email,
        'fileIndex' : fileIndex
    })
}).then((response) => {
    console.log('found response : ' , response)
    if (response.ok) {
        const link = document.createElement('a');
        console.log('response.url : ' , response.url)
        const newURL = response.url + `?filename=${email}_fin_plate_connection_${fileIndex}.osi`
        console.log('newURL : ' , newURL)
        link.href = newURL
        console.log('link.href : ' , link.url)
        link.setAttribute('download', `${email}_fin_plate_connection_${fileIndex}.osi`);
        link.innerHTML = `${email}_fin_plate_connection_${fileIndex}.osi`
```

Figure 8.9: obtainSingleInputFile Thunk

Chapter 9

Postgres Database

9.1 Script for generating Postgres Database

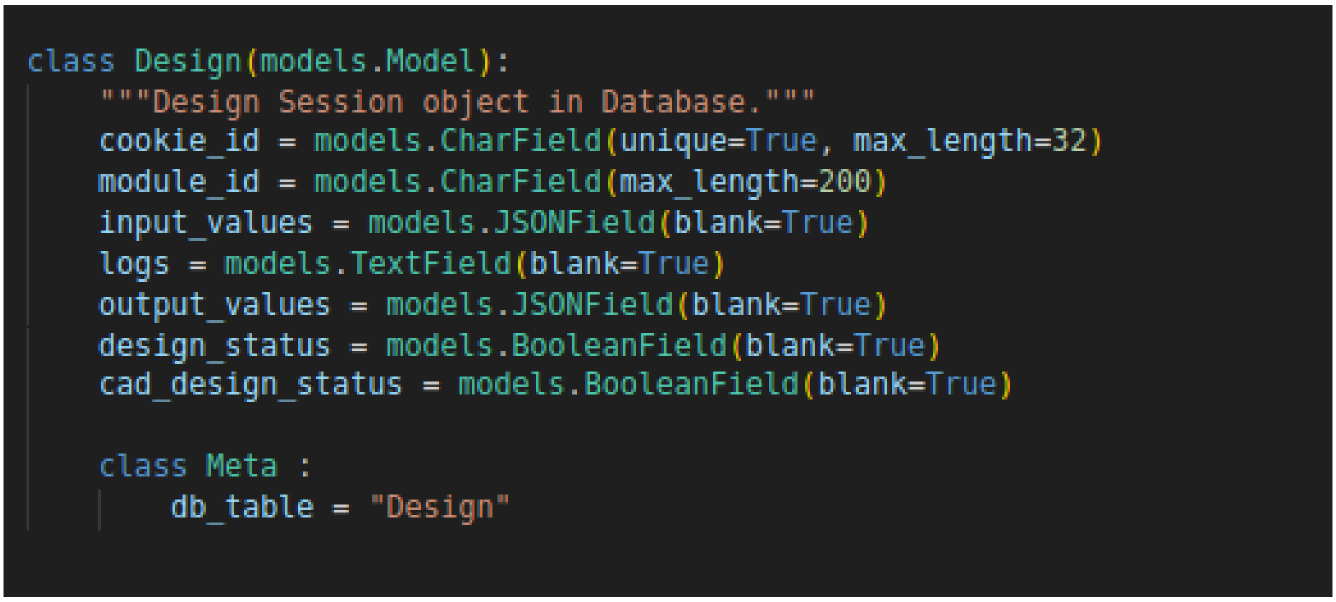
A new script is developed for creating a Postgres database and populating the database. This script is located in the `ResourceFiles/Database/postgres_Intg_osdag.sql` file. This script will populate the database with the name 'postgres_Intg_osdag'. We have created this database while installing and configuring Postgres in Chapter 1. This action can be executed by running the command : `python populate_database.py` file in the server terminal in the root of the project. The 'populate_database.py' file will execute the Postgres script and give a success message upon completion.

9.2 Script for updating Sequences

A new script is developed for updating the sequences of all the tables in the `postgres_Intg_osdag` database. The reason was because, by default the Django's Postgres driver is unaware about the updation in the sequences in the Postgres database. The updation in the sequence is caused when a new record is inserted into the table. Say - When the database is populated. The boltDiameter contains (11 records - guess). So the sequence should have the next value in the record which is 12. But since Django is unaware about the updation in the sequences, the script present in the file `ResourceFiles Database update_sequences.py` updates the sequences in the Django's table. This will make insertion and deletion in any of the tables of the database hassle free. This script can be run by running the command : `python update_sequences.py` in the server terminal in the root of the project. This python file will execute the update_sequences.sql script and give a success message upon successful completion. We have already performed this action while configuring Postgres database in Chapter 1.

9.3 Design model

The Design model stores session module related information like - cookie_id, module_id, input_values, logs, output_values, design_status, cad_design_status. All these information is used for session activities, creating the Design reports, checking output values and CAD model generation status. All the models in the `osdag/models.py` file have been developed by me.

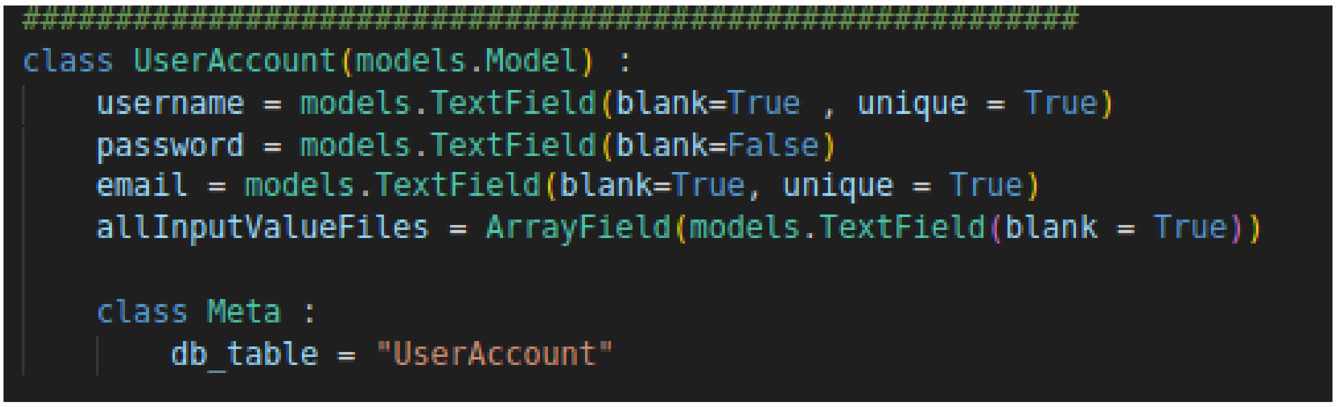


```
class Design(models.Model):
    """Design Session object in Database."""
    cookie_id = models.CharField(unique=True, max_length=32)
    module_id = models.CharField(max_length=200)
    input_values = models.JSONField(blank=True)
    logs = models.TextField(blank=True)
    output_values = models.JSONField(blank=True)
    design_status = models.BooleanField(blank=True)
    cad_design_status = models.BooleanField(blank=True)

    class Meta :
        db_table = "Design"
```

Figure 9.1: Design model

9.4 UserAccount model

The UserAccount model stores user credentials information like - username, password, email, allInputValueFiles. These information is associated to the user. IT is used for User authentication, authorization and any user based actions like creating the input files for design report. This model has a field called allInputValueFiles, which stores the data in the form of an ArrayField(). This ArrayField() is not part of the Django's provided data types for models, instead it is a data type which comes with the Postgres driver for Django. The ArrayField(models.TextField(blank=True)) means that it is an array having element as a TextField() from Django data type and the initial element (0th index) element is " (blank text field).



```
############################################################
class UserAccount(models.Model) :
    username = models.TextField(blank=True , unique = True)
    password = models.TextField(blank=False)
    email = models.TextField(blank=True, unique = True)
    allInputValueFiles = ArrayField(models.TextField(blank = True))

    class Meta :
        db_table = "UserAccount"
```

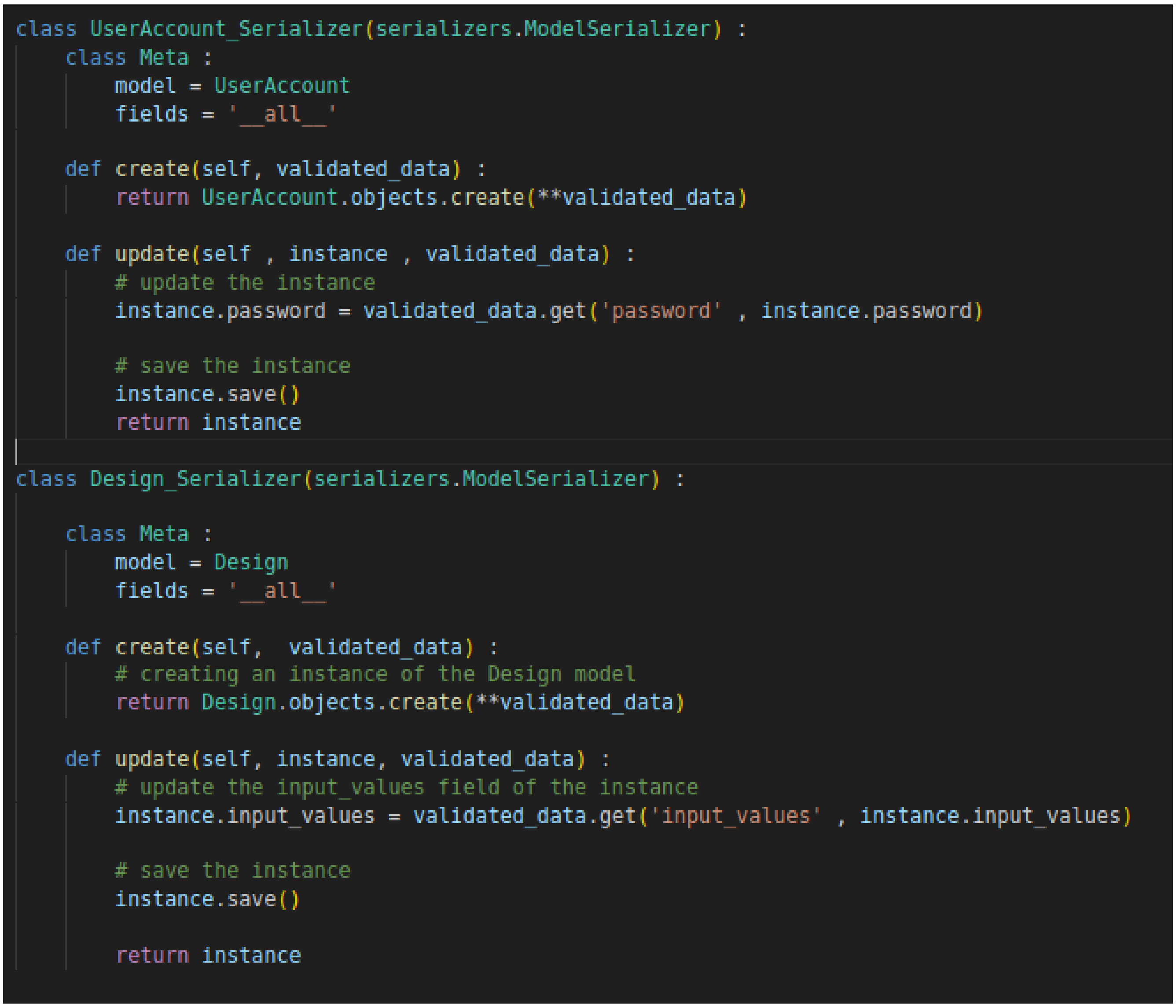
Figure 9.2: UserAccount model

Chapter 10

Serializers

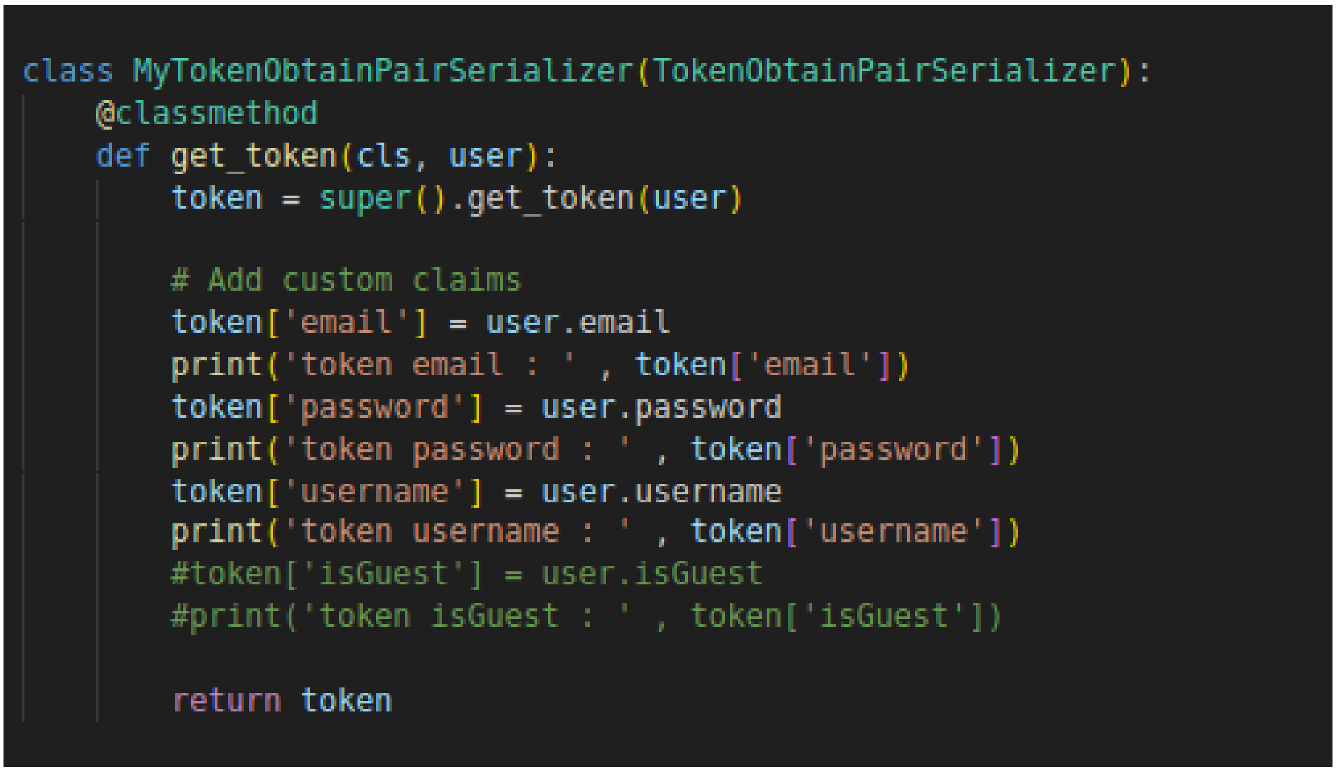
10.1 All Serializers

All the serializers are present in the `osdag/serializers.py` file for each and every model. These serializers will check if the data that needs to be saved in the model has a correct format or not. All the serializers are written by me. The UserAccount Serializer also updates the password of existing user. The MyTokenObtainPairSerializer allows to set custom token claims in the JWT token body. This means that the data in the JWT token can be customized, this data is encrypted and can be securely sent back and forth to the client. The client can then decode this token and read the information to perform any operation.



```python
class UserAccount_Serializer(serializers.ModelSerializer) :
    class Meta :
        model = UserAccount
        fields = '__all__'

    def create(self, validated_data) :
        return UserAccount.objects.create(**validated_data)

    def update(self , instance , validated_data) :
        # update the instance
        instance.password = validated_data.get('password' , instance.password)

        # save the instance
        instance.save()
        return instance

class Design_Serializer(serializers.ModelSerializer) :

    class Meta :
        model = Design
        fields = '__all__'

    def create(self,  validated_data) :
        # creating an instance of the Design model
        return Design.objects.create(**validated_data)

    def update(self, instance, validated_data) :
        # update the input_values field of the instance
        instance.input_values = validated_data.get('input_values' , instance.input_values)

        # save the instance
        instance.save()

        return instance
```

Figure 10.1: UserAccount and Design serializers



```python
class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
    @classmethod
    def get_token(cls, user):
        token = super().get_token(user)

        # Add custom claims
        token['email'] = user.email
        print('token email : ' , token['email'])
        token['password'] = user.password
        print('token password : ' , token['password'])
        token['username'] = user.username
        print('token username : ' , token['username'])
        #token['isGuest'] = user.isGuest
        #print('token isGuest : ' , token['isGuest'])

        return token
```

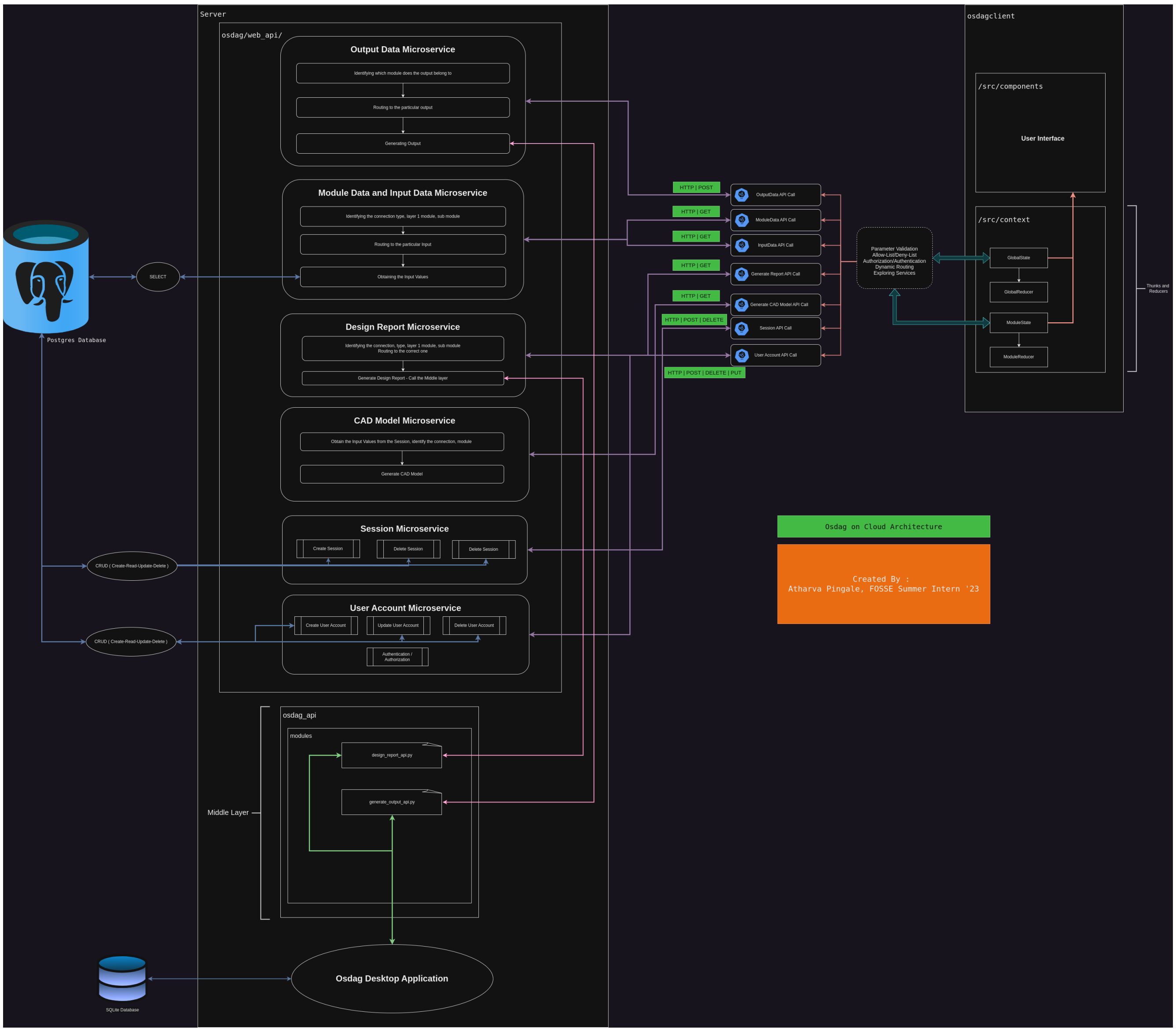
Figure 10.2: MyTokenobtain Serializer

Chapter 11

Architecture

11.1 Osdag on Cloud architecture

I have designed this architecture for Osdag on Cloud. I have designed this with the best of my knowledge and this can be used as a blueprint for further development. The architecture gives a really good idea about the flow of the project, how things work and what are the components/services that are being called. At this time, the architecture is still `Monolithic`. I have proposed an architecture that uses `Microservices` layers. The idea behind this is to break back-end service into micro-services to have better scalability, control and improve collective performance of the application. In future, things like database caching, HTTP caching, Server level caching, API gateway, load-balancing etc can be integrated if it is built on this architecture. This architecture is still tweakable and not complete yet. It will change as per the requirements. So, future interns/developers can freely extend this architecture or modify it for a better use case.



Server
osdag/web_api/
Output Data Microservice
Identifying which module does the output belong to
Routing to the particular output
Generating Output
Module Data and Input Data Microservice
Identifying the connection type, layer 1 module, sub module
Routing to the particular Input
Obtaining the Input Values
Design Report Microservice
Identifying the connection, type, layer 1 module, sub module
Routing to the correct one
Generate Design Report - Call the Middle layer
CAD Model Microservice
Obtain the Input Values from the Session, identify the connection, module
Generate CAD Model
Session Microservice
Create Session
Delete Session
Delete Session
User Account Microservice
Create User Account
Update User Account
Delete User Account
Authentication / Authorization
Postgres Database
SELECT
CRUD (Create-Read-Update-Delete)
CRUD (Create-Read-Update-Delete)
osdag_api
modules
design_report_api.py
generate_output_api.py
Middle Layer
Osdag Desktop Application
SQLite Database
HTTP | POST
HTTP | GET
HTTP | GET
HTTP | GET
HTTP | GET
HTTP | POST | DELETE
HTTP | POST | DELETE | PUT
OutputData API Call
ModuleData API Call
InputData API Call
Generate Report API Call
Generate CAD Model API Call
Session API Call
User Account API Call
Parameter Validation
Allow-List/Deny-List
Authorization/Authentication
Dynamic Routing
Exploring Services
osdagclient
/src/components
User Interface
/src/context
GlobalState
GlobalReducer
ModuleState
ModuleReducer
Thunks and Reducers
Osdag on Cloud Architecture
Created By :
Atharva Pingale, FOSSE Summer Intern '23

Figure 11.1: Osdag on Cloud Architecture

References

- https://github.com/osdag-admin/Osdag-web
- https://github.com/SurajBhosale003/Osdag-web
- https://osdag.fossee.in/resources/downloads
- https://www.django-rest-framework.org/
- https://docs.djangoproject.com/en/4.2/
- https://stoplight.io/api-design-guide
- https://react-redux.js.org/
- https://redux-toolkit.js.org/
- https://jwt.io/introduction
- https://docs.djangoproject.com/en/4.2/ref/databases/
- https://www.postgresql.org/docs/
- https://javascript.info/