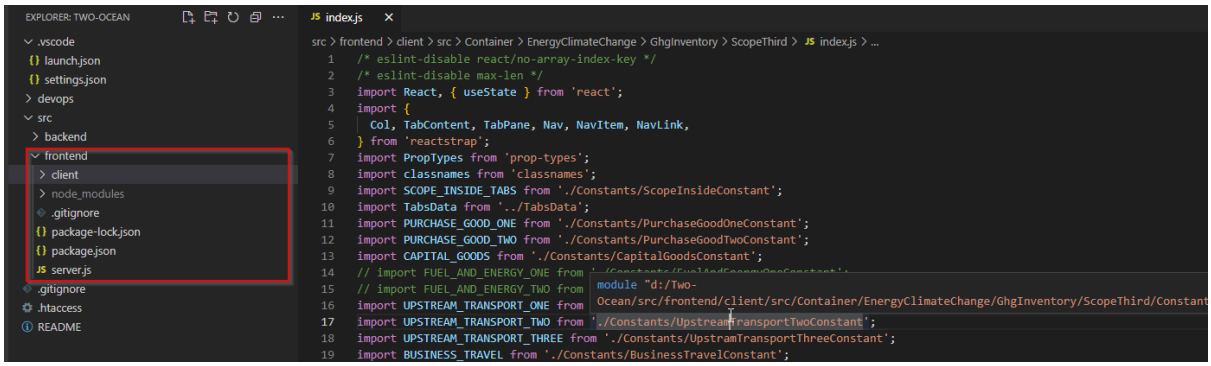
Front End Documentation

Created	@July 19, 2022 11:59 AM
Last Edited Time	@July 26, 2022 8:19 PM
Type	Technical Spec
Status	
Created By	
Last Edited By	
Stakeholders	
Property	

Let’s move to the project detail page ?
How to generate tables for each material issue ?
What is calculate method and how to use it ?

Overview of Folder Structure

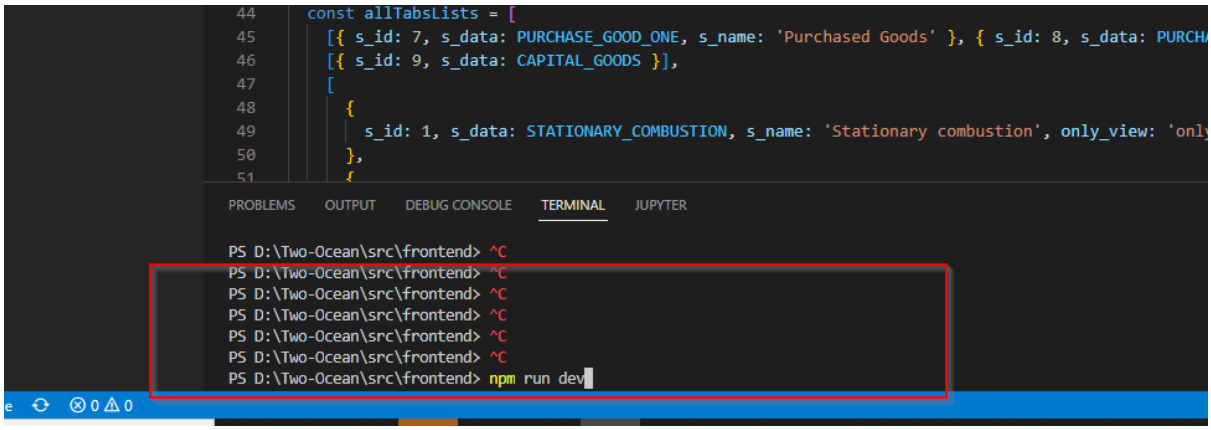
Here you can see a front-end folder, this folder is the base of the front-end code including node and react part. As you can see a **server.js** file, this file is basically used for API purpose and making the API URL hidden.



Also you can start your project from here by going through the terminal command:

“npm run dev”

You cannot start the project by adding the command “npm start” because of dependencies of the node.



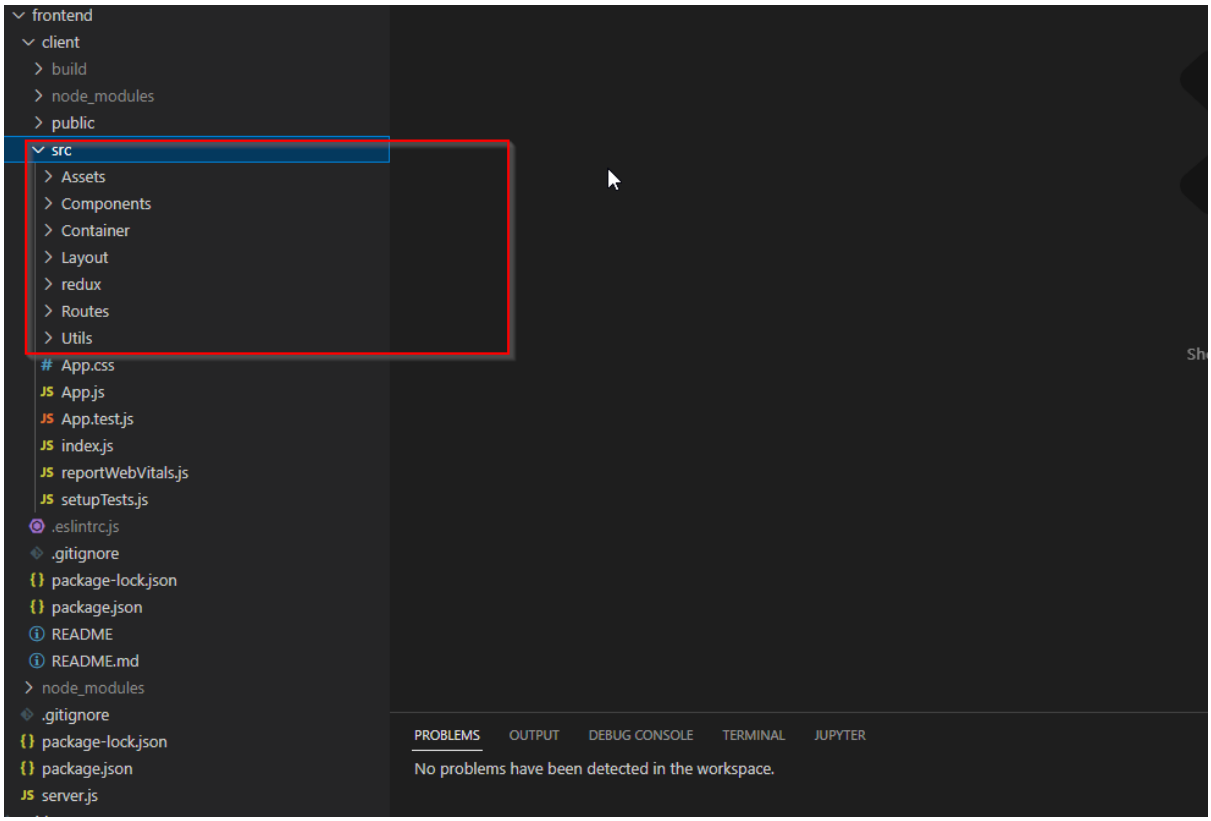
When you open a project on visual studio code you will find below listed folders under the **(src > frontend > client > src)** folder. Here you will find the code of entire dashboard including images.

but remember one thing project will not going to start from here it will be start from the frontend folder.

Frontend Folder is a primary folder which have all the dependent node API code and under the Frontend you will find client folder which is represent to the webpage view part.

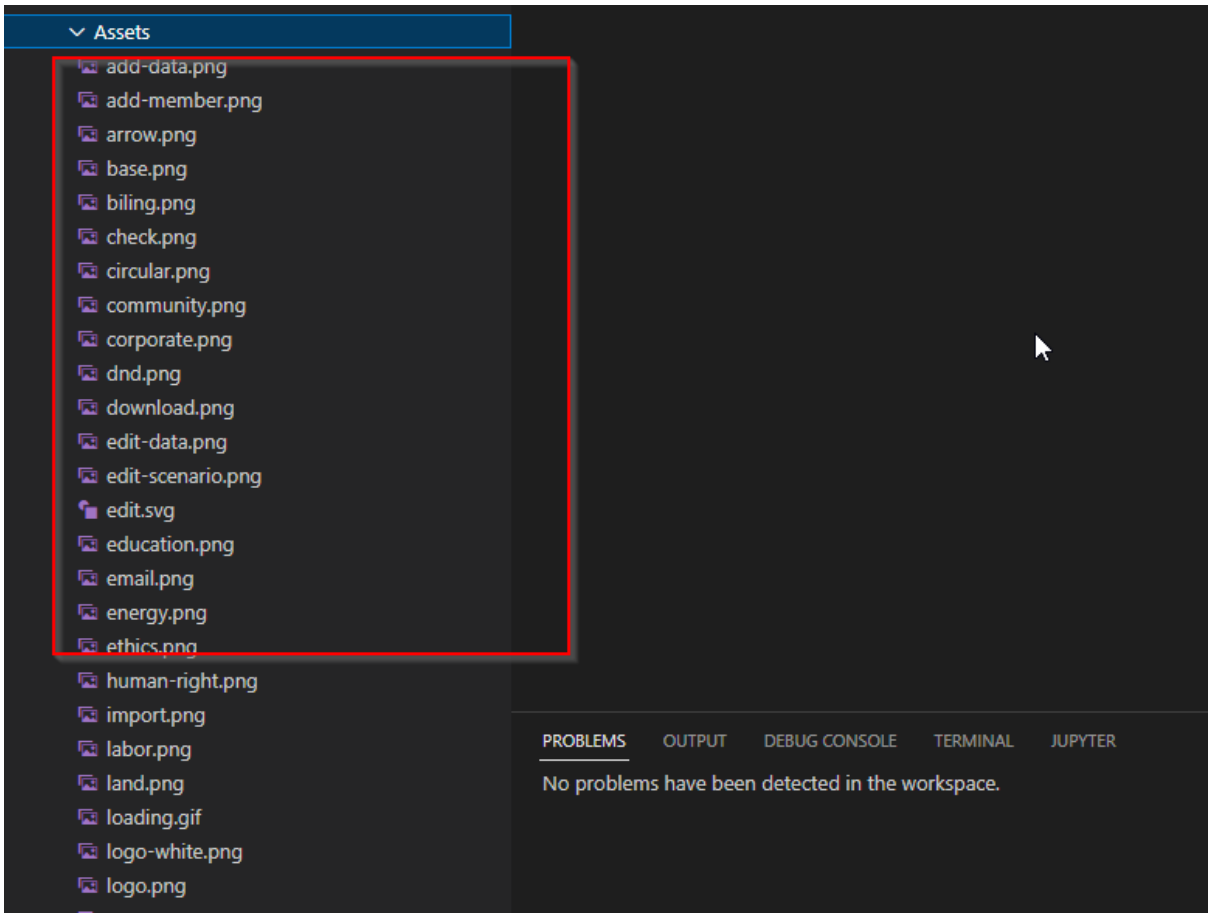
What are the use of these folders ?:

Let me explain you briefly one by one what are the uses of these folder



1) Assets

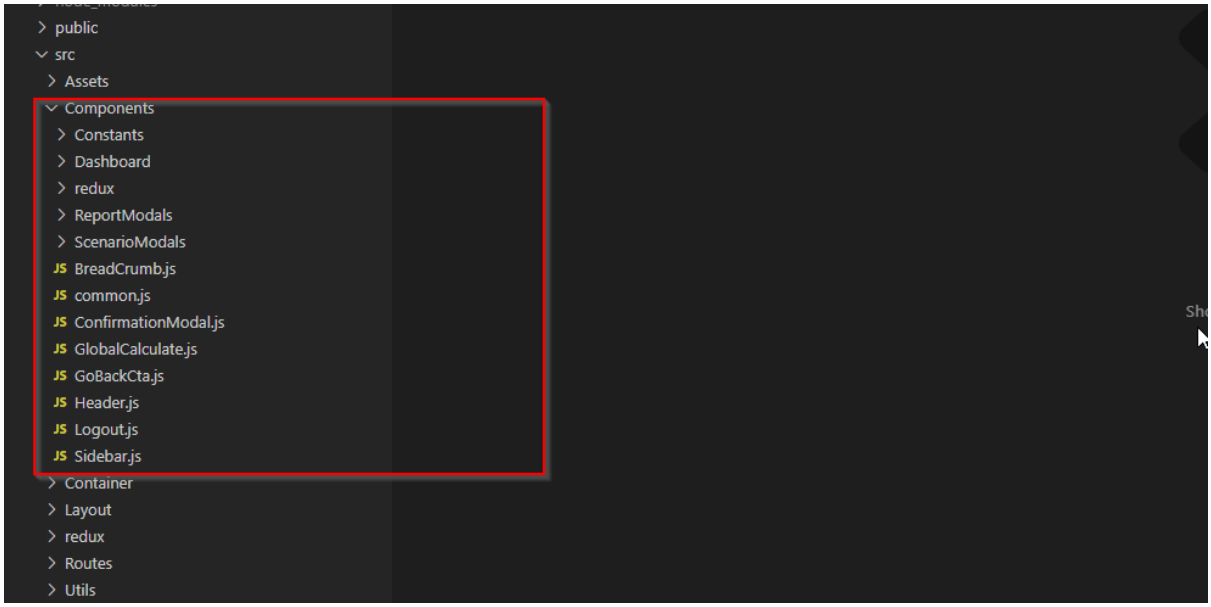
Assets folder is basically used for images for the whole dashboard. here you will find only static used images for the dashboard not dynamic appendid images.



2) Components

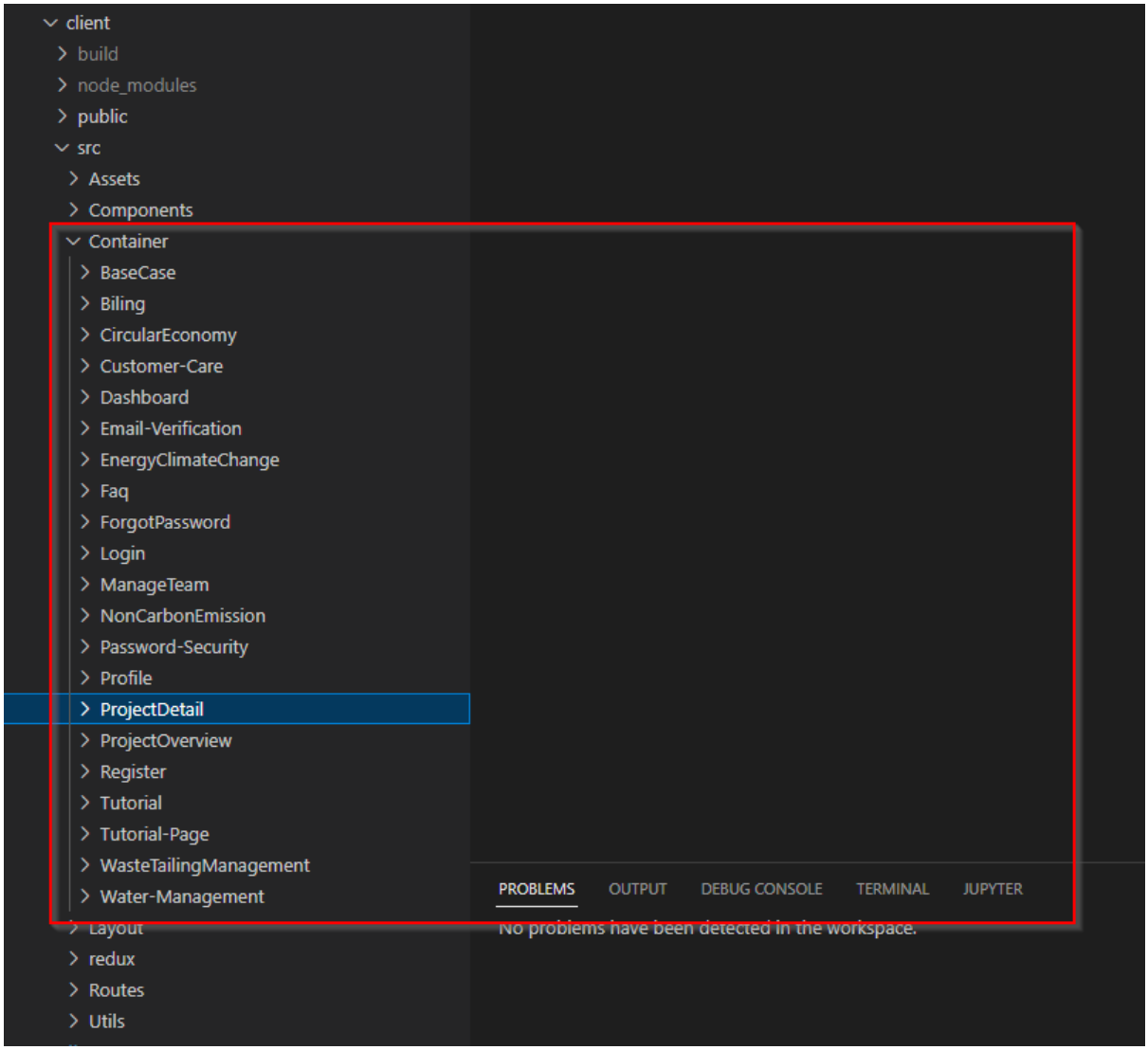
Components are those type of things which are going to be displayed of the entire dashboard

Example: header, footer, dashboard, breadcrumb, sidebar etc.



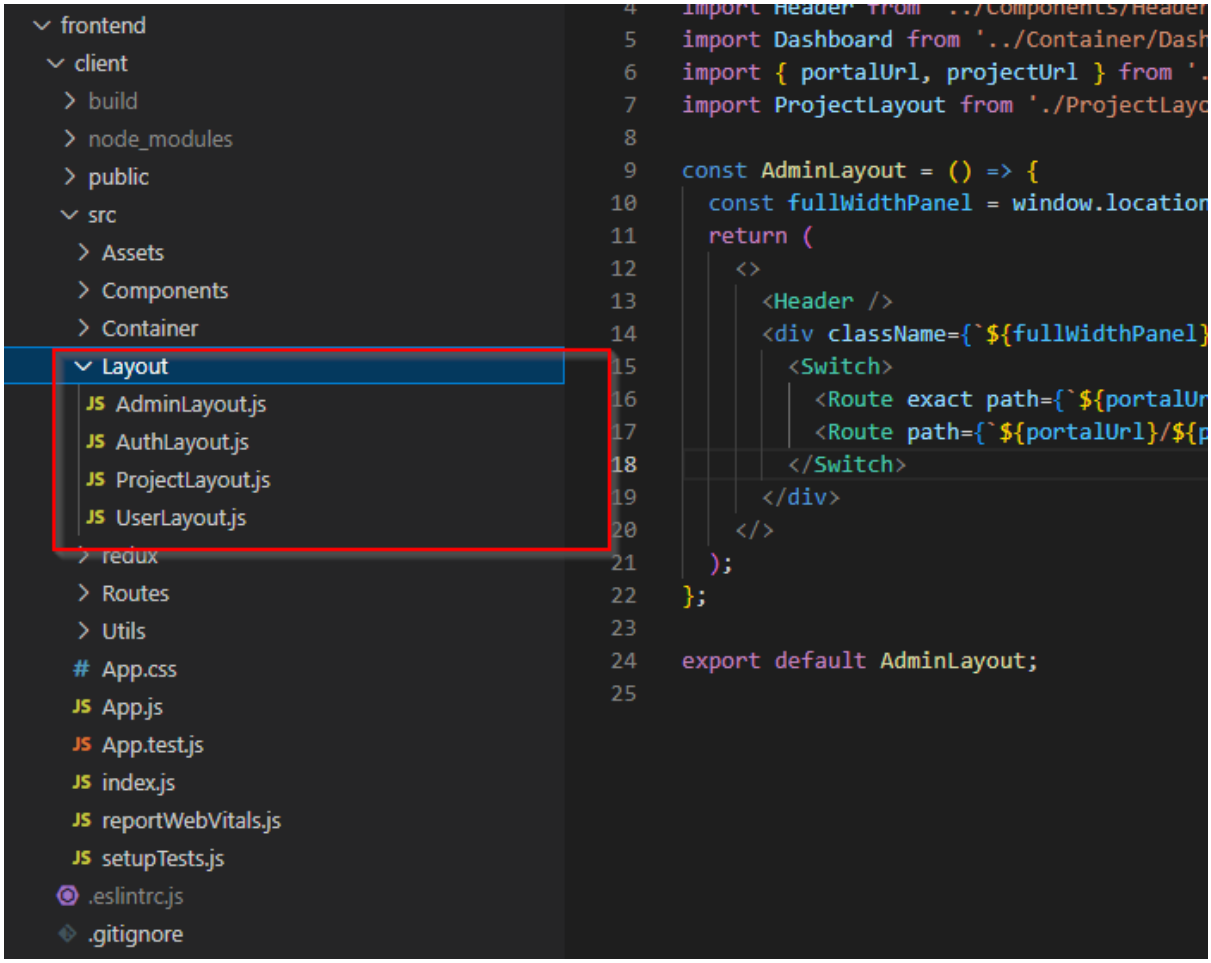
3) Container

Container are the main folder of the dashboard. here you will find all the pages of dashboard with the proper folder names according to the URL of the pages.



4) Layout

Layout is basically used for routing. Under the layout you will find the 4 types of layout structure.



5) Redux

Redux folder is used for creating a middleware for the project. as you can see there are two files one is **reducer** and second is **store.** They both file are backbone for the whole API data, they store the API data and distribute to the whole project, no matter where your folder or file are located you can easily collect the data form the store.

```
src
  Assets
  Components
  Container
  Layout
  redux
    reducer.js
    store.js
  Routes
  Utils
  App.css
  App.js
  App.test.js
  index.js
  reportWebVitals.js
  setupTests.js
.eslintrc.js
.gitignore
package-lock.json
package.json
README
```

```
  dashboardReducer,
  energyAndClimateReducer,
  componentReducer,
  waterReducer,
});

const rootReducer = (state, action) => {
  if (action.type === 'USER_LOGOUT') {
    return reducer(undefined, action);
  }

  return reducer(state, action);
};

export default rootReducer;
```

6) Routes

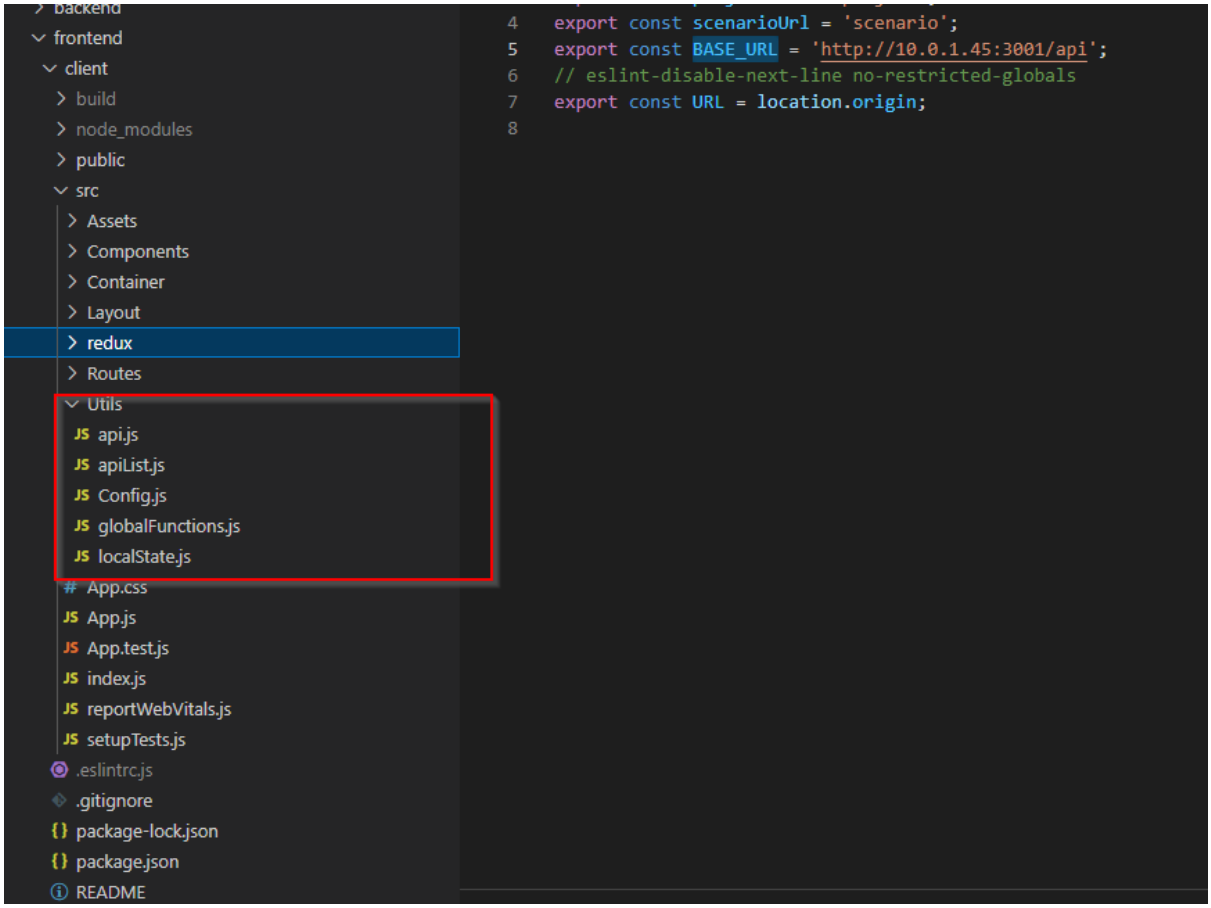
Basically routes are the combination of the layout part you will find only 2 routes under the routes folder

```
client
  build
  node_modules
  public
  src
    Assets
    Components
    Container
    Layout
    redux
    Routes
      Private.js
      Public.js
    Utils
    App.css
    App.js
    App.test.js
    index.js
    reportWebVitals.js
    setupTests.js
  .eslintrc.js
```

```
  render: Component, path, auth,
}) => <Route path={path} render={() => (auth ? <Component /> : <Redirect t

PrivateRoute.propTypes = {
  path: PropTypes.string.isRequired,
  render: PropTypes.oneOfType([
    PropTypes.shape({ render: PropTypes.func.isRequired }),
    PropTypes.func,
  ]).isRequired,
  auth: PropTypes.string,
};

PrivateRoute.defaultProps = {
  auth: null,
};
export default PrivateRoute;
```

7) Utils

Utils are the common functions which is going to be used for globally. Under the utils you will find the all API names, API instance, Base API URL, Global functions, local storage etc.



Hope you will understand the basic layout structure of the TOS

Move to the next page for better understanding the code and the API flow.

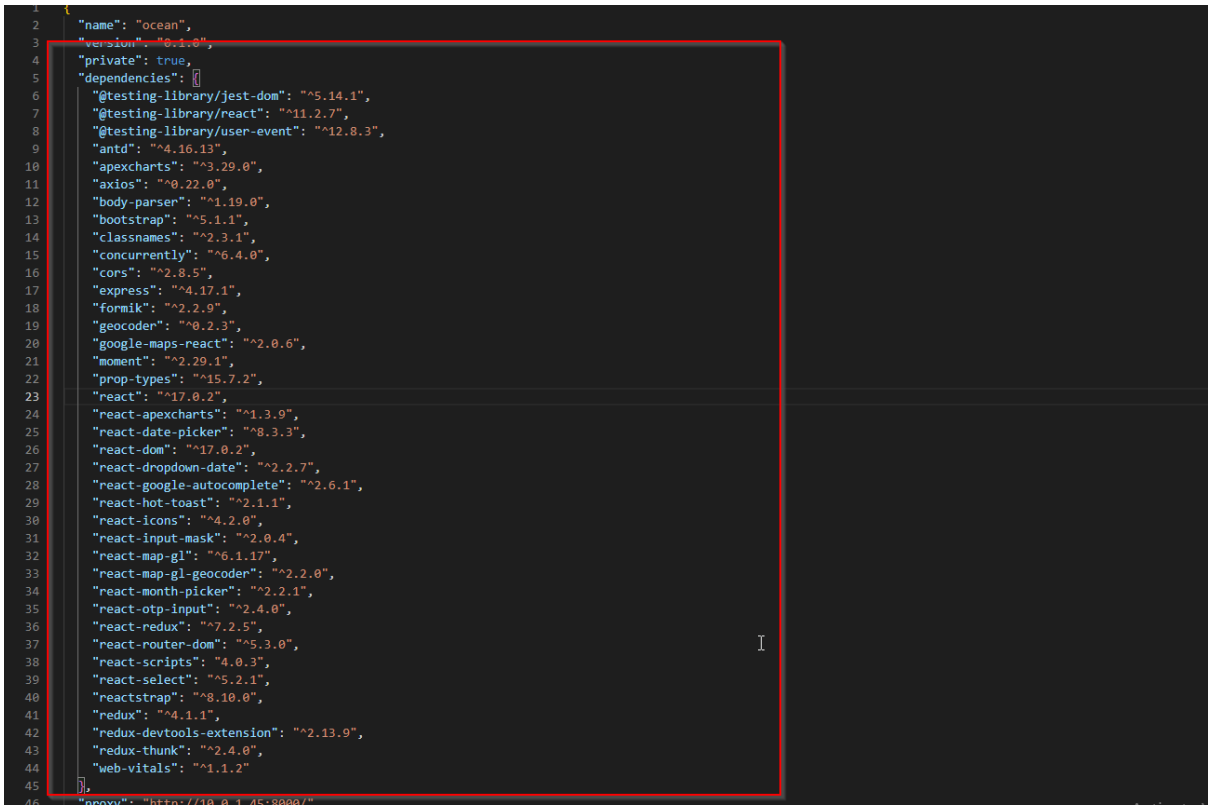
Quick Overview of Packages/Plugins

Before going to start on coding part Let me show you some packages and plugins that we have used in TOS Tool

Once you open your code on Vs-code, Under the Client folder you will see a **package.json** File. Here you can see all the packages and their versions that we have used for generating the TOS Tool.

Let me give you some major packages brief explanation.

Antd	Used for the Monthly date-picker and for range slider
ApexCharts	Used for the dashboard graphs charts
Reactstrap	Used for the designing purpose
React Hot Toast	Used for the notifications and success/error messages
Redux	Used for the data store
Formik	Used for the forms / validations



```
{
  "name": "ocean",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.14.1",
    "@testing-library/react": "^11.2.7",
    "@testing-library/user-event": "^12.8.3",
    "antd": "^4.16.13",
    "apexcharts": "^3.29.0",
    "axios": "^0.22.0",
    "body-parser": "^1.19.0",
    "bootstrap": "^5.1.1",
    "classnames": "^2.3.1",
    "concurrently": "^6.4.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "formik": "^2.2.9",
    "geocoder": "^0.2.3",
    "google-maps-react": "^2.0.6",
    "moment": "^2.29.1",
    "prop-types": "^15.7.2",
    "react": "^17.0.2",
    "react-apexcharts": "^1.3.9",
    "react-date-picker": "^8.3.3",
    "react-dom": "^17.0.2",
    "react-dropdown-date": "^2.2.7",
    "react-google-autocomplete": "^2.6.1",
    "react-hot-toast": "^2.1.1",
    "react-icons": "^4.2.0",
    "react-input-mask": "^2.0.4",
    "react-map-gl": "^6.1.17",
    "react-map-gl-geocoder": "^2.2.0",
    "react-month-picker": "^2.2.1",
    "react-otp-input": "^2.4.0",
    "react-redux": "^7.2.5",
    "react-router-dom": "^5.3.0",
    "react-scripts": "4.0.3",
    "react-select": "^5.2.1",
    "reactstrap": "^8.10.0",
    "redux": "^4.1.1",
    "redux-devtools-extension": "^2.13.9",
    "redux-thunk": "^2.4.0",
    "web-vitals": "^1.1.2"
  },
  "proxy": "http://10.0.1.45:8000/"
```

Coding Flow and Structure

Let’s go through the coding structure.

As you can see we have used a formik for generating the entire dashboard forms.

What is formik?

Formik is the world's most popular open source form library for React and React Native. Formik is used for form validation or dynamic forms. let’s have a look at once.

```
<Formik
  initialValues={{ email: '', password: '' }}
  validate={(values) => {
    const errors = {};
    if (!values.email) {
      errors.email = 'Email is Required';
    } else if (!/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i.test(values.email)) errors.email = 'Invalid email address';

    if (!values.password) {
      errors.password = 'Password is Required';
    } else if (values.password.length <= 6) {
      errors.password = 'Password must be more than 6 characters';
    }
    return errors;
  }}
  onSubmit={async (values, { setSubmitting }) => {
    dispatch(userLogin({ login_name: values.email, password: values.password }))
      .then((data) => {
        if (data) {
          history.push(`${portalUrl}/dashboard`);
          sendNotification('success', 'Login Successfully', 1000, 'center-top');
        }
      });
    setSubmitting(false);
  }}
>
  {({
    values,
    errors,
    touched,
    handleChange,
    handleBlur,
    handleSubmit,
    isSubmitting,
  }) => (
    <Form onSubmit={handleSubmit}>
      <FormGroup>
        <Label for="Email">Email</Label>
        <Input
          type="email"
          name="email"
          id="Email"
          onChange={handleChange}
          onBlur={handleBlur}
          value={values.email}
        />
        <p className="text-danger m-0">{errors.email && touched.email && errors.email}</p>
```

As you can see we have used three method based on formik

1	Initial Values
2	Validations

3	Submit Button Method

These three methods are going to be very helpful for generating a proper form and it is also easy for managing the API data.

Note: On some cases we have used the YUP validation just because of the dynamic forms. but this form is used only for normal fields. that's why you can see a if else validation case.

After generating the form

Once you create the forms with perfect validation now it's to to send the data to the backend.

We had also created a method for that which is going to be very helpful for you.

Let me explain you with the example:

As you can see we have used a **REDUX** method for sending and receiving the data.

What is redux?

Redux is an open-source JavaScript library for managing and centralizing application state. It is most commonly used with libraries such as React or Angular for building user interfaces.

It's also help you to manage and store the data on reducer.

```
> Email-Verification
> EnergyClimateChange
> Faq
> ForgotPassword
> Login
> ManageTeam
> NonCarbonEmission
> Password-Security
> Profile
> ProjectDetail
> ProjectOverview
> Register
  > redux
    JS action.js
    JS constant.js
    JS reducer.js
    JS index.js
> Tutorial
> Tutorial-Page
> WasteTailingManagement
> Water-Management
> Layout
> redux
> Routes
> Utils
# App.css
```

```
129                 history.push('/email-verification');
130                }
131                setSubmitting(false);
132              });
133            }}
134          >
135            {({
136              values, errors, touched, handleChange, handleBlur, handleSubmit, isSubmitting
137            }) => (
138              <Form onSubmit={handleSubmit}>
139                <Row>
140                  <Col sm="6">
141                    <FormGroup>
142                      <Label for="firstName">First Name</Label>
143                      <Input
144                        type="text"
145                        name="first_name"
146                        id="firstName"
147                        onChange={handleChange}
148                        onBlur={handleBlur}
149                        value={values.first_name}
150                      />
151                      <p className="text-danger m-0">{errors.first_name && touched.first_na
152                    </FormGroup>
153                  </Col>
154                  <Col sm="6">
155                    <FormGroup>
156                      <Label for="lastName">Last Name</Label>
157                      <Input
158                        type="text"
```

With the help of redux you can fetch the data from any of the component without fetching the API again and again. it is very easy to use

Let me show you the example:

As you can see i am importing the country and industry from the register reducer. It means I have already fetched the API on register Page.

```
30      business_sector_id: 0,
31      website: '',
32      geo_country_id: 0,
33      password: '',
34      confirmPassword: '',
35      // user_verification_link: `${URL}/email-verification`,
36    };
37    const history = useHistory();
38    const dispatch = useDispatch();
39    const {
40      countries,
41      industries,
42      // registerStatus,
43      userEmailVerification,
44    } = useSelector((state) => state.registerReducer);
45
46    useEffect(() => {
47      const initialLoad = () => {
48        dispatch(getCountries());
49        dispatch(getIndustries());
50      };
51
52      initialLoad();
53    }, [dispatch]);
```

Basically Redux have 3 files:

1	Constant
2	Reducer
3	Action

What are the use of these 3 Files?

Let's start with the constant,

Constant: is basically used for create a static variable that's are going to connect with the reducer and action file:

Reducer: Reducer is basically used for storing the data and passing to the other components

Action: Action is used for performing the action like what API you are going to be call and what are the names of API and action and what are the method you are going to be call

POST/DELETE/UPDATE

Let me show you some example:

Constant File:

```
Help                                                constant.js - Two-Ocean - Visual Studio Code
JS index.js ...\Login   JS index.js ...\Register   JS constant.js ×
src > frontend > client > src > Container > Register > redux > JS constant.js > ...
1   export const COUNTRY_LIST = 'COUNTRY_LIST';
2   export const INDUSTRY_LIST = 'INDUSTRY_LIST';
3   export const USER_REGISTER = 'USER_REGISTER';
4   export const EMAIL_VERIFICATION_START = 'EMAIL_VERIFICATION_START';
5   export const EMAIL_VERIFICATION_SUCCESS = 'EMAIL_VERIFICATION_SUCCESS';
6   export const EMAIL_VERIFICATION_ERROR = 'EMAIL_VERIFICATION_ERROR';
7   export const RESEND_EMAIL_VERIFICATION_START = 'RESEND_EMAIL_VERIFICATION_START';
8   export const RESEND_EMAIL_VERIFICATION_SUCCESS = 'RESEND_EMAIL_VERIFICATION_SUCCESS';
9   export const RESEND_EMAIL_VERIFICATION_ERROR = 'RESEND_EMAIL_VERIFICATION_ERROR';
10  export const USER_DATA = 'USER_DATA';
11  export const SET_USER_TOKEN = 'SET_USER_TOKEN';
12  export const EMAIL_CHECK_SUCCESS = 'EMAIL_CHECK_SUCCESS';
13  export const EMAIL_CHECK_START = 'EMAIL_CHECK_START';
14  export const EMAIL_CHECK_ERROR = 'EMAIL_CHECK_ERROR';
15  export const FORGOT_PASSWORD = 'FORGOT_PASSWORD';
16  export const RESEND_FORGOT_PASSWORD_START = 'RESEND_FORGOT_PASSWORD_START';
17  export const RESEND_FORGOT_PASSWORD_ERROR = 'RESEND_FORGOT_PASSWORD_ERROR';
18  export const RESEND_FORGOT_PASSWORD_SUCCESS = 'RESEND_FORGOT_PASSWORD_SUCCESS';
19  export const FORGOT_PASSWORD_TOKEN = 'FORGOT_PASSWORD_TOKEN';
20  export const ADD_NEW_PASSWORD = 'ADD_NEW_PASSWORD';
21  export const USER_PROFILE = 'USER_PROFILE';
22  export const UPDATE_USER_PROFILE = 'UPDATE_USER_PROFILE';
23  export const CHANGE_PASSWORD = 'CHANGE_PASSWORD';
24  export const CREATE_PROJECT_MEMEBER = 'CREATE_PROJECT_MEMEBER';
25
```

Action File:

```
action.js - Two-Ocean - Visual Studio Code
JS index.js ...\Login   JS index.js ...\Register   JS action.js ×
src > frontend > client > src > Container > Register > redux > JS action.js > registerUser > <function> > then() callback
19    USER_PROFILE, UPDATE_USER_PROFILE, CHANGE_PASSWORD, CREATE_PROJECT_MEMEBER,
20  } from './constant';
21  import {
22    META_GEO_COUNTRIES,
23    META_BUSINESS_SECTOR,
24    USERS,
25    APP_USER_SESSION,
26    APP_USER_LOGIN_CREDENTIAL_VERIFY,
27    PROJECT_MEMBERS,
28  } from '../../../Utils/apiList';
29  import Api from '../../../Utils/api';
30  import { errorsHandler } from '../../../Utils/globalFunctions';
31
32  // Fetching countries list
33  export const getCountries = () => (dispatch) => Api.get(META_GEO_COUNTRIES, { action: 'filter', format: 'dereference' }).then(
34    (res) => {
35      dispatch({ type: COUNTRY_LIST, payload: res.data.resource_list });
36    },
37    (err) => err,
38  );
39
```

Reducer File:

```
Help                                   reducer.js - Two-Ocean - Visual Studio Code
... JS index.js ...\Login   JS index.js ...\Register   JS reducer.js ×
src > frontend > client > src > Container > Register > redux > JS reducer.js > [⊙] INITIAL_STATE
1   import {
2     COUNTRY_LIST, EMAIL_VERIFICATION_ERROR, EMAIL_VERIFICATION_START,
3     EMAIL_VERIFICATION_SUCCESS, INDUSTRY_LIST, SET_USER_TOKEN, USER_REGISTER,
4     RESEND_EMAIL_VERIFICATION_START, RESEND_EMAIL_VERIFICATION_ERROR,
5     RESEND_EMAIL_VERIFICATION_SUCCESS, EMAIL_CHECK_START, EMAIL_CHECK_SUCCESS,
6     EMAIL_CHECK_ERROR, FORGOT_PASSWORD, RESEND_FORGOT_PASSWORD_START,
7     RESEND_FORGOT_PASSWORD_SUCCESS, RESEND_FORGOT_PASSWORD_ERROR, FORGOT_PASSWORD_TOKEN,
8     ADD_NEW_PASSWORD,
9     USER_PROFILE,
10    UPDATE_USER_PROFILE,
11    CHANGE_PASSWORD,
12    CREATE_PROJECT_MEMEBER,
13  } from './constant';
14
15  const INITIAL_STATE = {
16    countries: [],
17    industries: [],
18    getUserProfiledata: [],
19    registerStatus: false,
20    forgotPassword: false,
21    emailVerificationLoading: false,
22    emailVerification: false,
23    resendEmailForVerification: '',
24    resendEmailForgot: '',
25    userToken: null,
26    newPasswordUpdate: null,
27    userEmailVerification: {
28      loading: false,
29      status: false,
30    },
31  };
32
33  const reducer = (state = INITIAL_STATE, action) => {
34    switch (action.type) {
35      case COUNTRY_LIST:
36        return {
37          ...state, countries: action.payload,
38        };
39      case INDUSTRY_LIST:
40        return {
41          ...state, industries: action.payload,
42        };
43      case USER_REGISTER:
44        return {
45          ...state, registerStatus: true, resendEmailForVerification: action.payload.email_id,
46        };
```

As you can see the above images these all are the combination of redux part.

Now Let's move the the next step Error cases and Success Cases Notifications:

When You fetch the some API now you will have the 2 type of response cases one is **Success** and second one is **Failed.**

For managing the both cases we have created a global function and pass into the Action method

let me show you with the example:

As you can see we have used a **errorhandler** method for managing the errors. this is a global function that we have stored in globally function

```
  });
};

// User Login
export const userLogin = (values) => (dispatch) => (
  Api.post(APP_USER_SESSION, values).then((res) => {
    dispatch({ type: SET_USER_TOKEN, payload: res.data.resource.attributes.access_token });
    return true;
  }, (err) => {
    errorsHandler(err);
    return false;
  })
);

// User Email Check
```

This is our global function method that we are using for each and every error cases

```
Help                                   globalFunctions.js - Two-Ocean - Visual Studio Code
JS action.js   JS globalFunctions.js ×
src > frontend > client > src > Utils > JS globalFunctions.js > ...
  } from '../Components/common';
  // import { useSelector } from 'react-redux';

  export const sendNotification = (type, message, duration = 4000, position = 'top-right') => {
    toast[type](message, {
      duration,
      position,
    });
  };

  export const errorsHandler = (error) => {
    if (error.response.status === 400 && error?.response?.data?.return_status?.length > 0) {
      sendNotification('error', error.response.data.return_status[0].message);
    } else if (error.response.status === 401) {
      sendNotification('error', error.response.data.return_status);
    } else if (error.response.status === 500) {
      sendNotification('error', error.response.data.return_status);
    } else if (error.response.status === 404) {
      sendNotification('error', error.response.data.return_status);
    }
  };

```

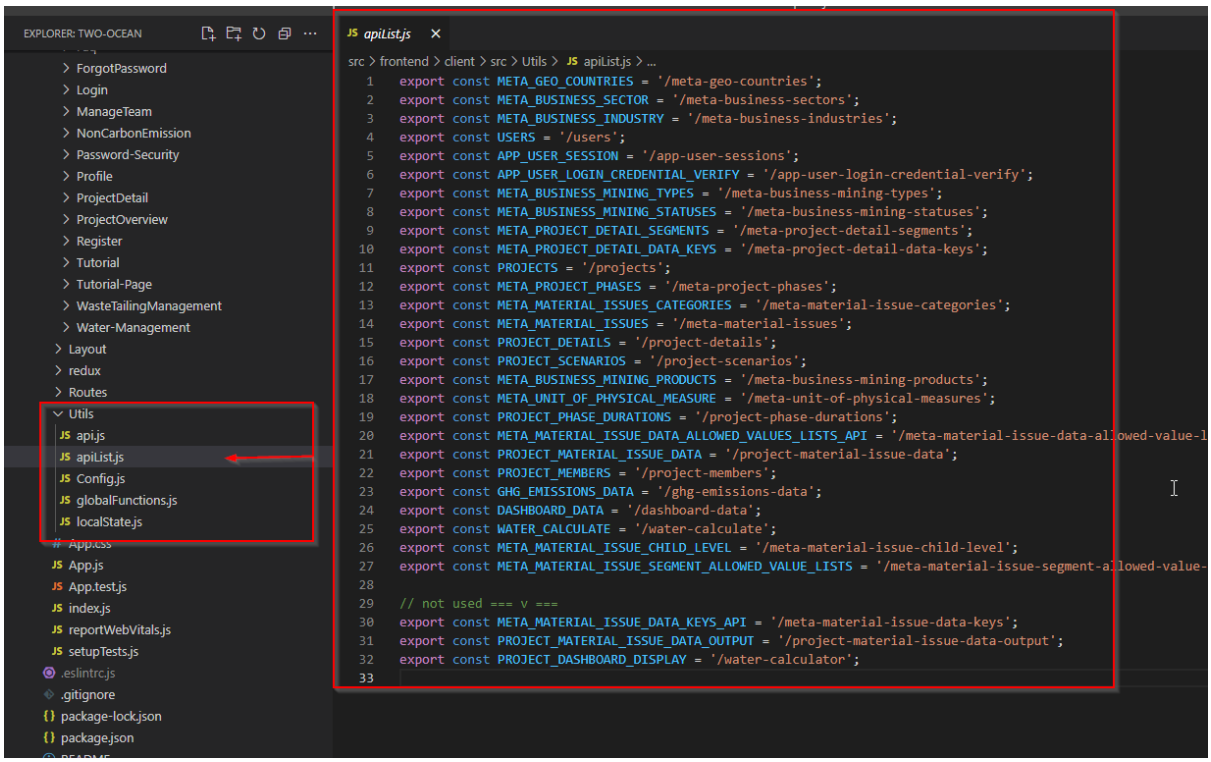
Also for managing the error we have also used a package called React-Toast

It’s also help us to show the messages on a popup and looks pretty.

I hope you understand all the flow how we are generating the form and sending or getting the data from backend. Now let's move to the next step with some additional features that we have used.

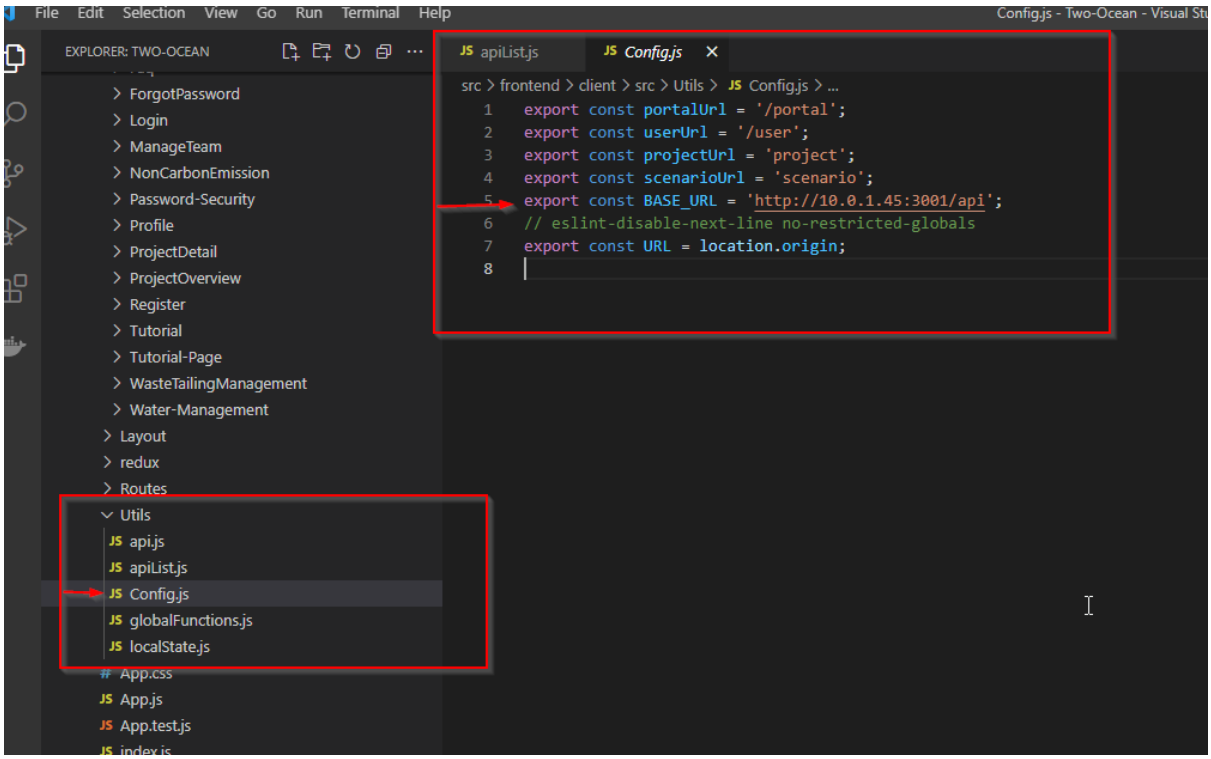
How to find all API names that are used in TOS?

When you open the Utils page you will find the all API names that we have used for developing the TOS Tool. We have used a const method for storing the API names, which means you do not have to redeclare the API name while fetching the data just use the const name.



How to find the base URL of API ?

Under the Utils page you will also find the **config.js** File. In this file you will find the all fixed routing values. fixed routings are those things which are not going to be changed like API base url and portal URL text etc.



Major Use Of Global Functions:

Global function are those function which is going to be used for the entire project. let me explain you with the example:

Suppose if you want to use the USER_ID you do not have to create the function for it, just go to the global function and check the **getUserId** function. This function will provide you the current user id

It's not all about the user id you can also use so many things from the global function, like getting the difference from the years and getting the names of phases with the phase id and many more things.

```
25      sendNotification('error', error.response.data.return_status);
26    } else if (error.response.status === 404) {
27      sendNotification('error', error.response.data.return_status);
28    }
29  };
30
31  export const getUserId = () => {
32    try {
33      const token = JSON.parse(localStorage.getItem('appState')).registerReducer.userToken;
34      const forgotTokeSplit = token.split('.');
35      return JSON.parse(atob(forgotTokeSplit[1])).user_id;
36    } catch (e) {
37      return 0;
38    }
39  };
40
41  export const getYearDifference = (from = null, to = null) => {
42    if (from && to) {
43      const start = moment(from);
44      const end = moment(to);
45      const duration = end.diff(start, 'years', true).toFixed(1);
46      return duration >= 0 ? duration : '';
47    }
48    return '';
49  };
50
```

What is the use of localstate.js

We have used the localstate.js file for storing the reducer into the localstorage. without the localstorage you cannot save the data and pass into from one component to another component

```
1   export function restoreLocalState() {
2     let state = {};
3
4     try {
5       const storedState = localStorage.getItem('appState');
6
7       if (storedState) {
8         state = JSON.parse(storedState);
9       }
10    } catch (err) {
11      // If stored data is not a strigified JSON this might fail,
12      // that's why we catch the error
13    }
14
15    return state;
16  }
17
18  export function storeLocalState(state) {
19    localStorage.setItem('appState', JSON.stringify(state));
20  }
21
```

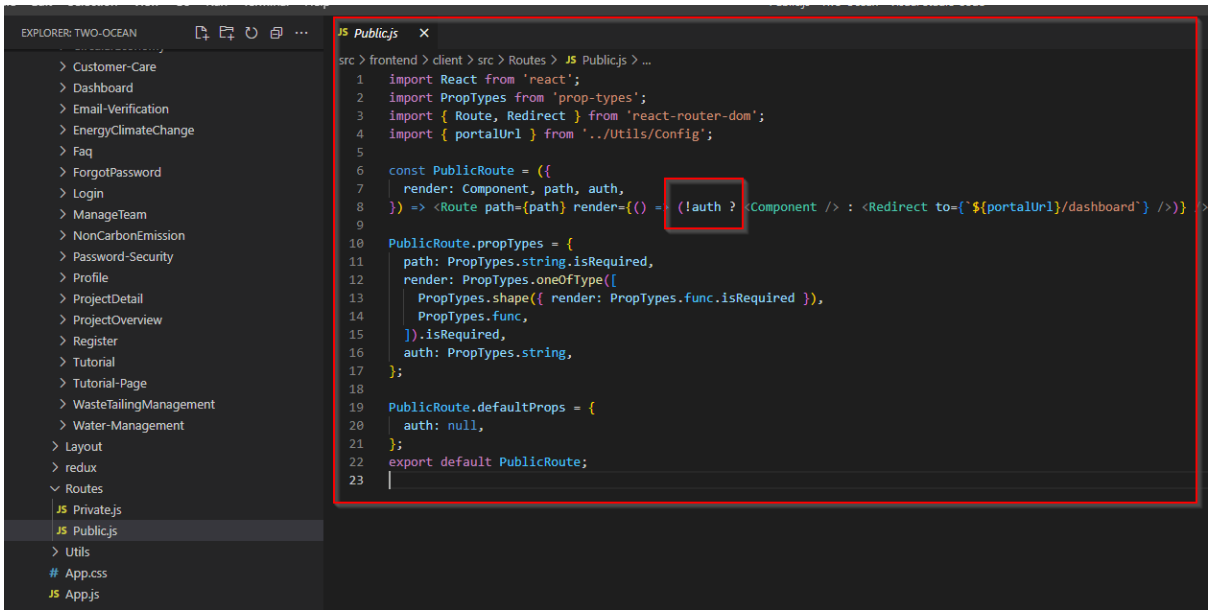
What is the use of localstate.js

We have used the localstate.js file for storing the reducer into the localstorage. without the localstorage you cannot save the data and pass into from one component to another component

Difference between Private and Public Route:

Private Route: Private Routes are those routes which have authorization access like when user logged in then they can access the dashboard and rest of the projects URL but they cannot access the unauthorized URL like Login, register etc.

Public Route: Public Routes are those routes which is not required any authorization process means you can access those pages by visiting through the URL, example: login, register, forgot Password etc. but you cannot view the dashboard until you do not have any authorization.



Layout Explanation: (Routing)

Let’s start with the layout how many types of layout we have till now.

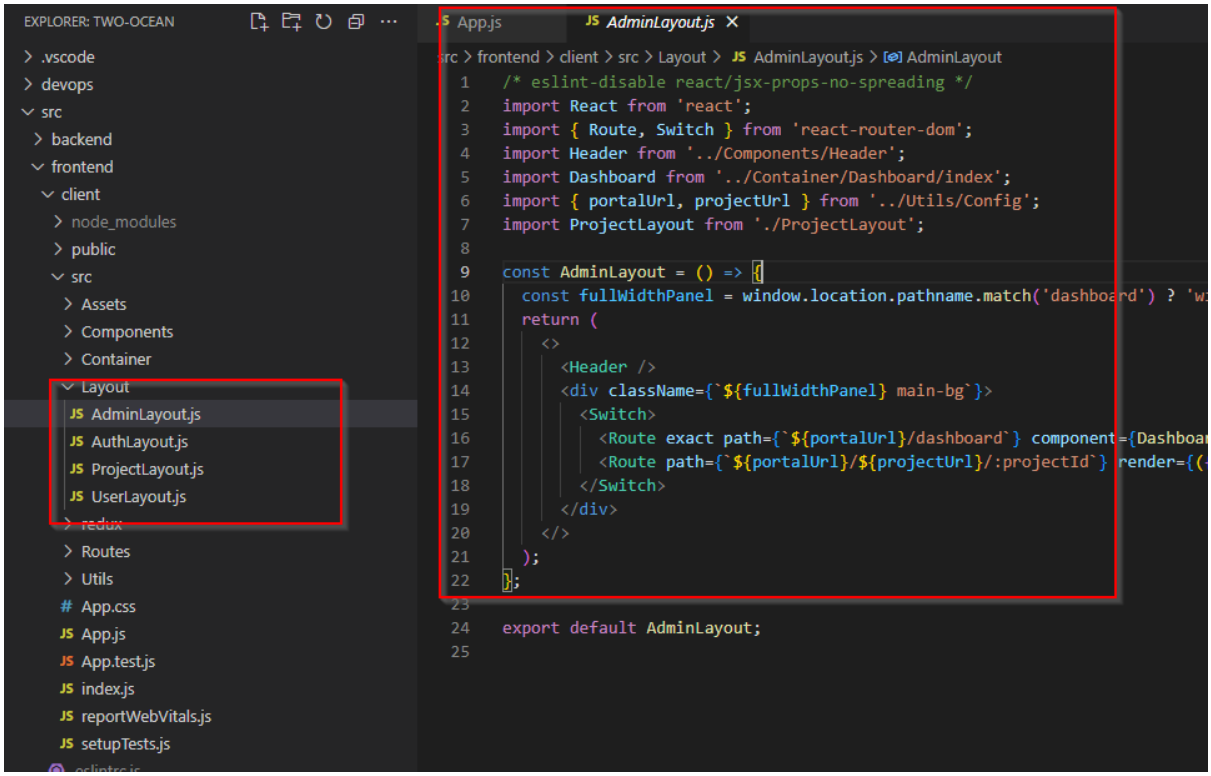
Right now we have four Type of layouts.

1) Admin Layout

2) Auth Layout

3) Project Layout

4) User Layout



1) What is Admin Layout ?

After login the web page you will find the admin layout routes. Under the admin layout you will find the two types of layout structure One is dashboard (without sidebar) another one is project layout (With sidebar).

(Note: Project layout and admin layout are the combination of each other you cannot access the project layout without the admin layout)

2) What is Auth Layout

Auth Layout is only used for public routes, means you can access the auth layout without login the webpage example: login/register/forgot email etc. pages

3) What is User Layout

User Layout are those layout which you can access the after login. This layout represent the user routings like user profile page, membership page, faq page, etc. These pages are represent to the

users and routing is also different for that.

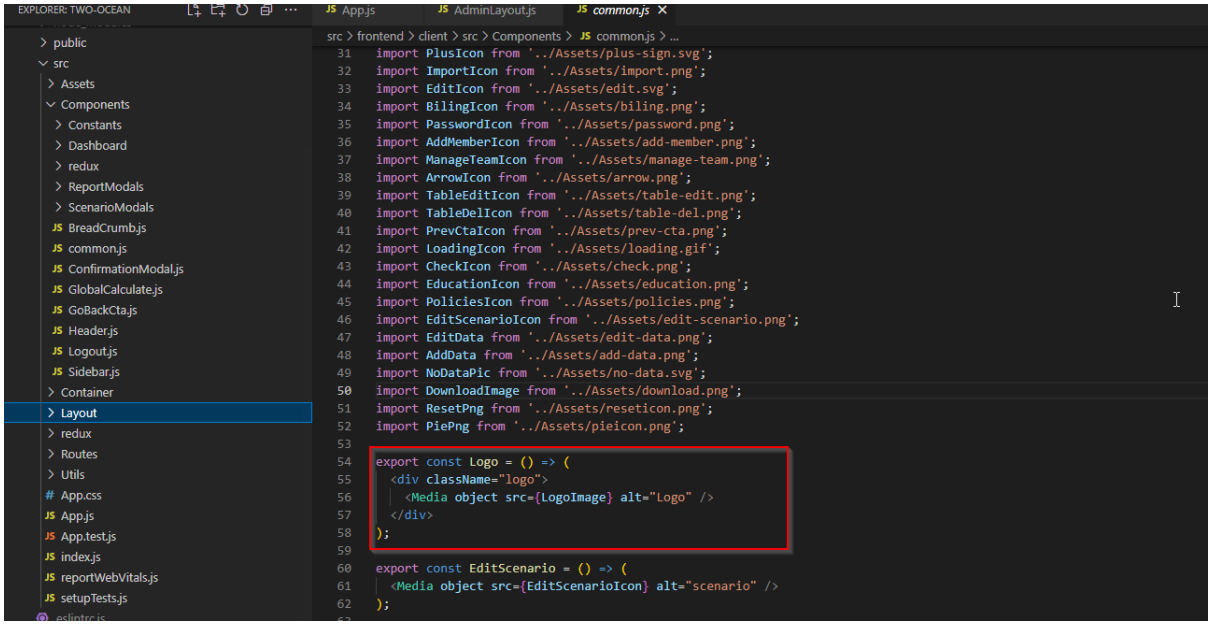
4) What is Project Layout

Project layout are child layout of admin layout. which means you can access those layout also after login the webpage. it also contain some dynamic url's which is used for the project routing.

(Example: Sending the scenario url on dashboard page)

What are the use common.js File?

Common.js file is basically used for **image routing.** Instead of using image directly into the component please use the common.js file for storing the image route and then import the image to the other components.

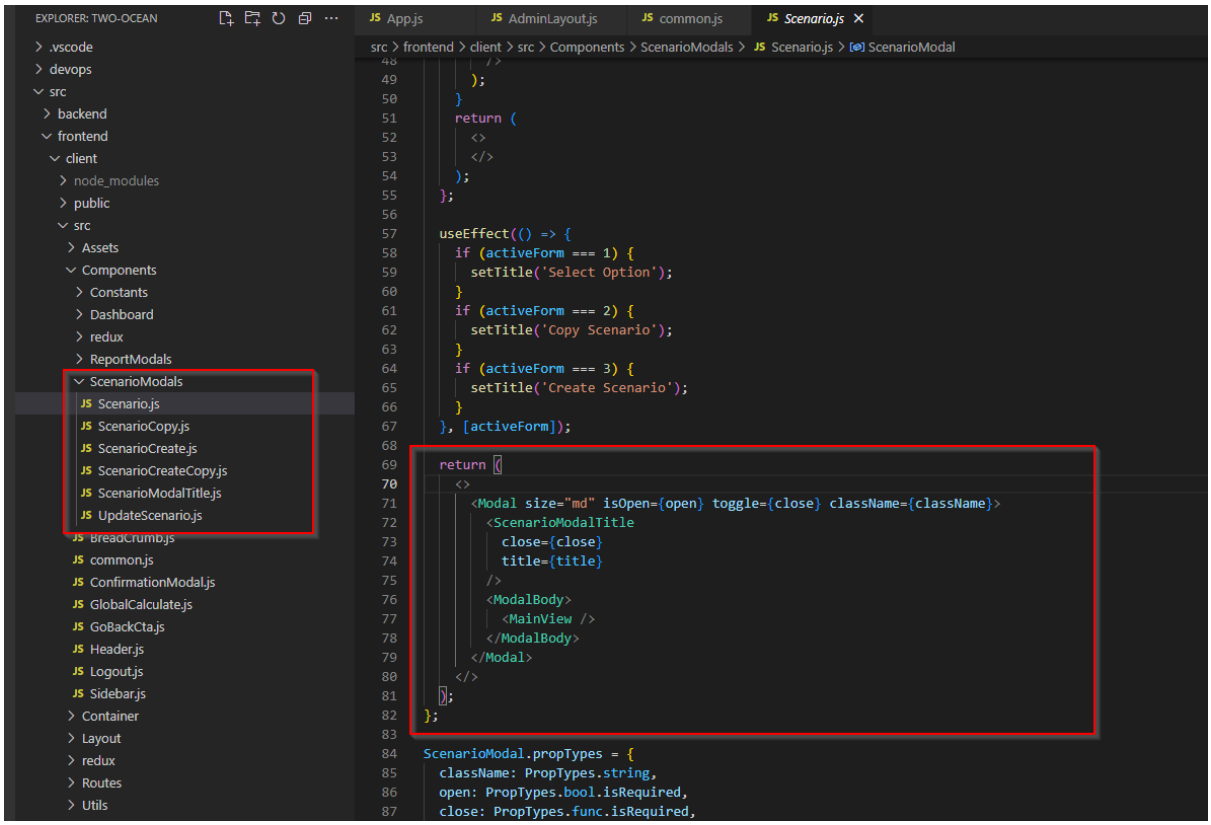


What are the use of Scenario ?

Scenario is the parent of whole data under the particular project. You can create multiple scenario's under one project. it means you can add multiple data on single project based on the scenario

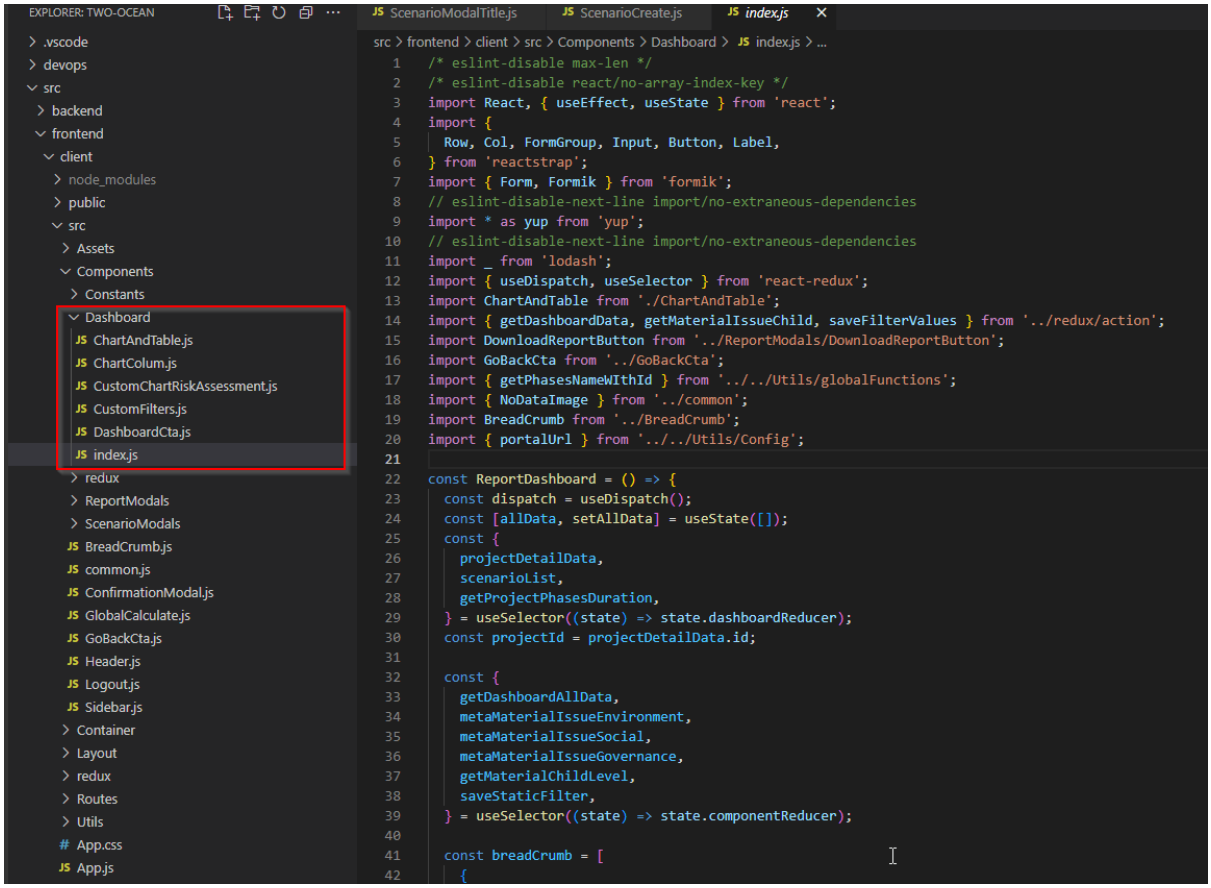
Under the scenario Modal Folder you will find **scenario.js** this is the main file of scenario.

You can also find scenario create file and scenario update file on that folder. **Also remember one thing scenario copy does not work right now this is only just a modal showcase and once you create a project a BASECASE scenario will be generated automatically**



What is Dashboard ?

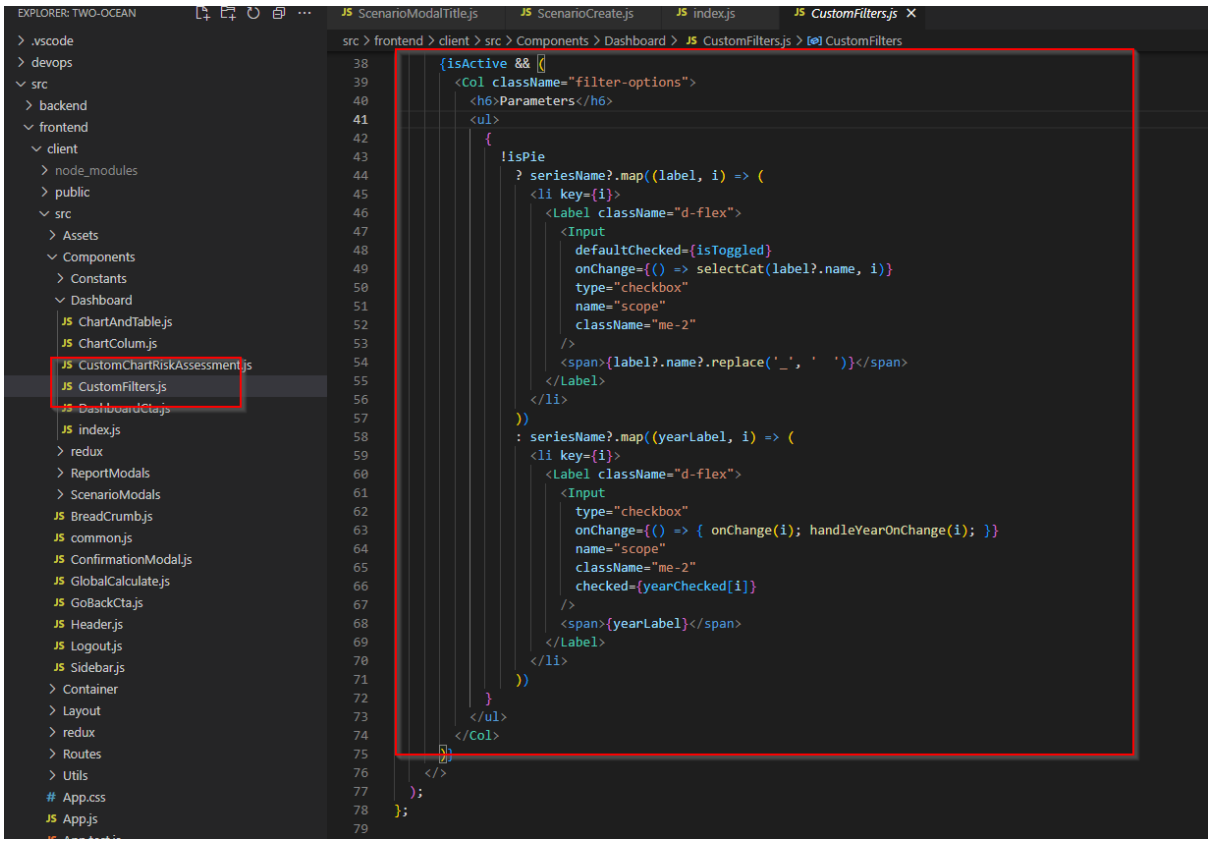
Dashboard is the collection of all data that you created based on years/ scenarios and material issues. when you open the dashboard folder you will find the **index.js** file. this is the filter file that show on the dashboard page when you visit the first time.



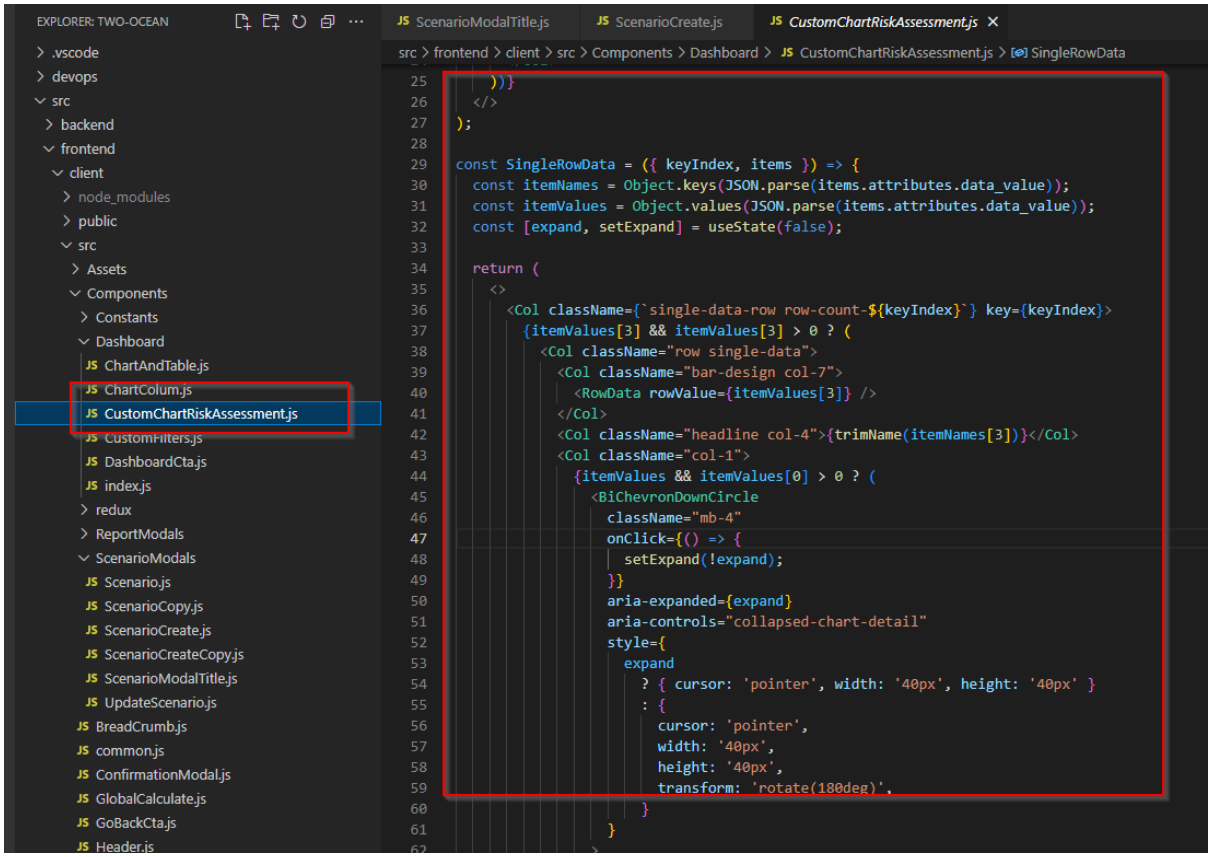
This filter include the project id, scenario id, all of the years and all material issue types
after applying all these filters you will find the exact result charts and table data according to that.

NOTE: We have used a single API for generating the dashboard Data Also we have used the APEX chart library for that.

Also we have use a custom filter feature on APEX chart you can also find that page named **CustomFilters.js** basically this file is used for generating the custom toggle functionality on APEX charts.



Also we have one more important thing on dashboard page. we have created a custom chart as per the requirement this chart is not related to the apex chart. we have created this chart based on the range slider. Filename is **CustomChartRiskAssessment.js**



Now let’s move to the container page.

Container is the page folder of the project which means it include all of the pages of the project.

As you can see we have all of the pages according to the folder structure. it means you can easily find out the page based on the routing (URL)

```
> Components
v Container
  > BaseCase
  > Biling
  > CircularEconomy
  > Customer-Care
  > Dashboard
  > Email-Verification
  > EnergyClimateChange
  > Faq
  > ForgotPassword
  > Login
  > ManageTeam
  > NonCarbonEmission
  > Password-Security
  > Profile
  > ProjectDetail
  > ProjectOverview
  > Register
  > Tutorial
  > Tutorial-Page
  > WasteTailingManagement
  > Water-Management
> Layout
> redux
> Routes
> Utils
```

```
25
26  const breadCrumb = [
27    {
28      active: false,
29      link: `${portalUrl}/dashboard`,
30      name: 'Home',
31    },
32    {
33      active: false,
34      link: '',
35      name: `${projectDetailData.attributes.name}`,
36    },
37    {
38      active: true,
39      link: `${scenarioId}`,
40      name: `${scenarioName}`,
41    },
42  ];
43
44  return (
45    <>
46      <Row className="pt-2 pb-2">
47        <BreadCrumb breadCrumb={breadCrumb} />
48        <Col className="d-flex align-items-center justify-content-end report_and_download">
49          <GlobalCalculateData />
50          <DownloadReportButton />
51          <DashboardCta />
52        </Col>
53      </Row>
54
55      <Col className="all_services mt-3">
```

You are already familiar with the scenario as i mentioned it above you can also check the default scenario page name **BASECASE**

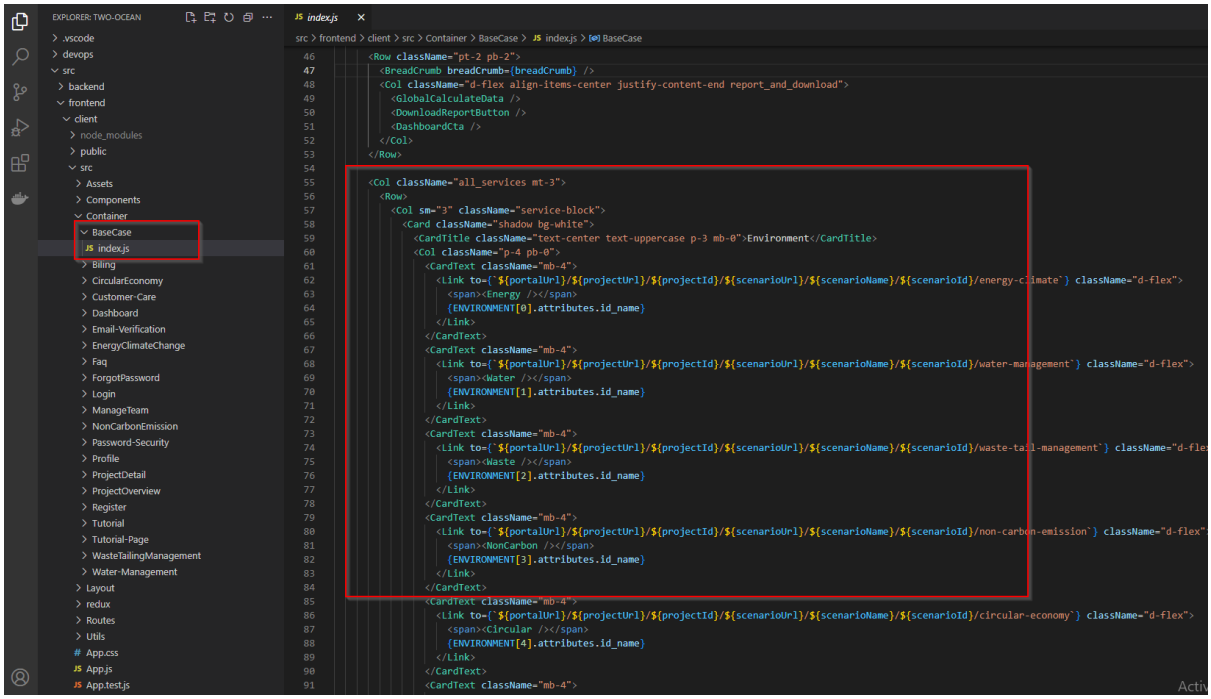
This is the collection of all data as you can see the below image we have passed all dynamic routes through the link.

and one more thing you can check here we have used a local JSON for getting the names

{ENVIRONMENT[1].attributes.id_name}

On some cases we have used API response into local JSON just because on loading time.

and this static JSON is not going to change.



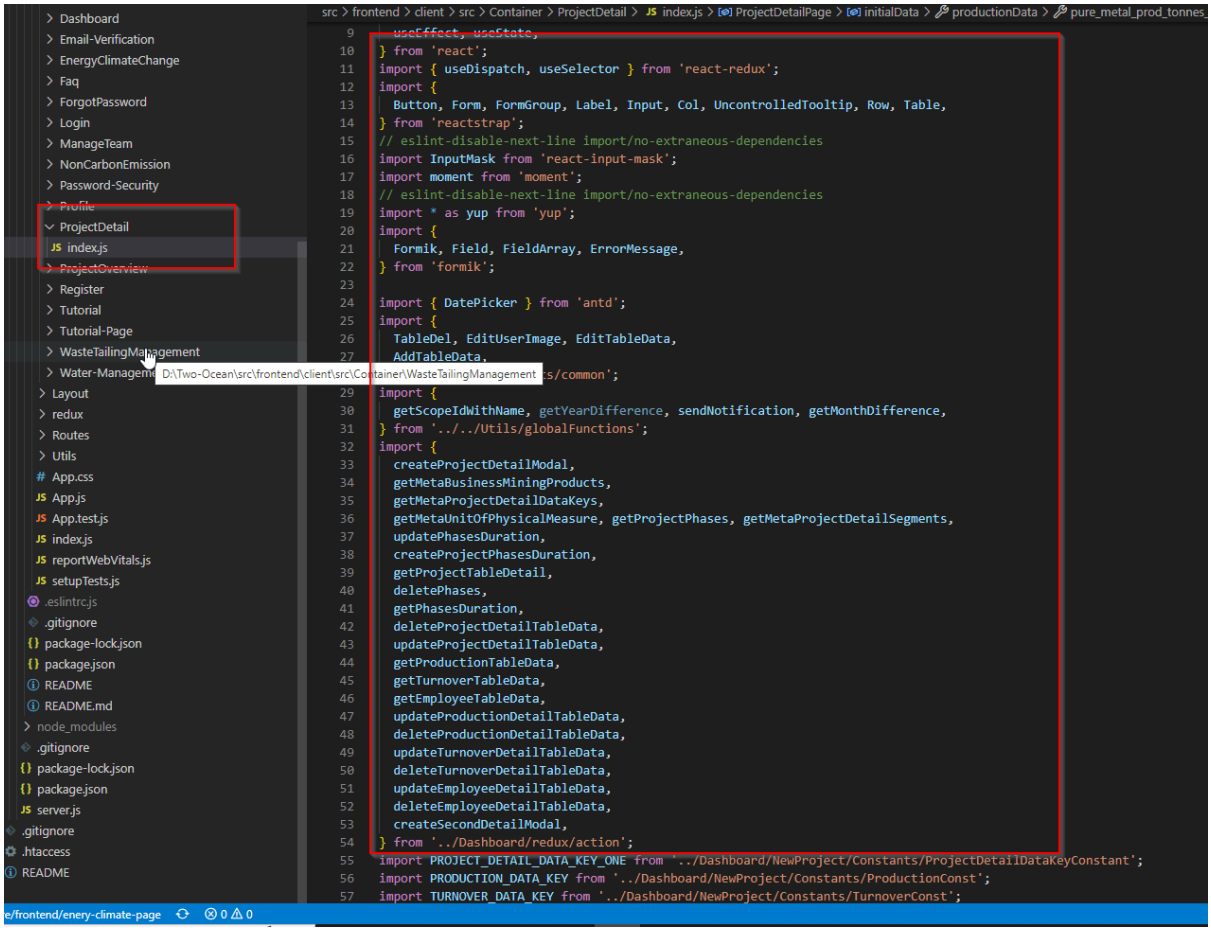
Let’s move to the project detail page ?

Project detail page is one of the most important page of the project. it will include all of the important features of the entire project.

While create the project you can see a lot of option like adding the years and products based on the years all these things are going to be use on internal pages when you try to add some data on table field.

let me show you how we develop the project detail page.

Once you visit the product detail page first you can see the import methods of all required things



After moving a bit down now you can see all of the field that we have on project detail page set as initial field according to the formik standard.

```
EnergyClimateChange
Faq
ForgotPassword
Login
ManageTeam
NonCarbonEmission
Password-Security
Profile
ProjectDetail
index.js
ProjectOverview
Register
Tutorial
Tutorial-Page
WasteTailingManagement
Water-Management
Layout
redux
Routes
Utils
App.css
App.js
App.test.js
index.js
reportWebVitals.js
setupTests.js
.eslintrc.js
.gitignore
package-lock.json
package.json
README
README.md
node_modules
.gitignore
package-lock.json
package.json
server.js
.gitignore
.htaccess
README
```

```
const dispatch = useDispatch();
const [isOpen, setIsOpen] = useState(false);
const [isActive, setActive] = useState('false');
const handleToggle = () => {
  setActive(!isActive);
};

const initialData = {
  picked: [],
  explorationstartdate: '',
  explorationenddate: '',
  constructionstartdate: '',
  constructionenddate: '',
  operationstartdate: '',
  operationenddate: '',
  closurestartdate: '',
  closureenddate: '',
  postclosurestartdate: '',
  postclosureenddate: '',
  projectTable: [{
    extracted_product_id: '',
    unit_of_physical_measure_id: '',
    estimated_ore_resources: '',
    avg_purity_ore_percentage: '',
    group_id: '',
  }],
  productionData: [{
    date_year: '',
    product: '',
    ore_production_tonnes_per_annum: '',
    ore_grade_percentage: '',
    pure_metal_prod_tonnes_per_annum: '',
    group_id: '',
  }],
  turnOver: [{
    date_year: '',
    usd_million: '',
    group_id: '',
  }],
  empDetails: [{
    date_year: '',
    company_employees: '',
    contract_employees: '',
    group_id: '',
  }],
};
```

also moving a bit down more you can see all of the initial loading API and getting data from the reducer for this page:

```
Dashboard
Email-Verification
EnergyClimateChange
Faq
ForgotPassword
Login
ManageTeam
NonCarbonEmission
Password-Security
Profile
ProjectDetail
index.js
ProjectOverview
Register
Tutorial
Tutorial-Page
WasteTailingManagement
Water-Management
Layout
redux
Routes
Utils
App.css
App.js
App.test.js
index.js
reportWebVitals.js
setupTests.js
.eslintrc.js
```

```
src > frontend > client > src > Container > ProjectDetail > index.js > ProjectDetailPage
    group_id: '',
  }],
};
const [initial, setInitial] = useState(initialData);

const {
  projectDetailData,
  projectPhasesData,
  metaBusinessMiningProducts,
  metaUnitOfPhysicalMeasure,
  metaProjectDetailSegments,
  getProjectPhasesDuration,
  getProjectTableDetailData,
  getProductionTableDetailData,
  getTurnoverTableDetailData,
  getEmployeeTableDetailData,
} = useSelector((state) => state.dashboardReducer);
const projectId = projectDetailData.id;

useEffect(() => {
  const initialLoad = () => {
    dispatch(getProjectPhases());
    dispatch(getMetaBusinessMiningProducts());
    dispatch(getMetaUnitOfPhysicalMeasure());
    dispatch(getMetaProjectDetailSegments());
    dispatch(getMetaProjectDetailDataKeys());
    dispatch(getPhasesDuration(projectId));
  };
  initialLoad();
  // eslint-disable-next-line
}, [dispatch]);

// PROJECT PHASES ==============================================>>>>>
```

Now once you’ve done with all above things now you can see the project phases section. project phases include all of the years that we have. Also you can add new years or remove the years from the phases. Below listed image code are setting the data based on the project phases ID.

```
EnergyClimateChange
Faq
ForgotPassword
Login
ManageTeam
NonCarbonEmission
Password-Security
Profile
ProjectDetail
index.js
ProjectOverview
Register
Tutorial
Tutorial-Page
WasteTailingManagement
Water-Management
Layout
redux
Routes
Utils
App.css
App.js
App.test.js
index.js
reportWebVitals.js
setupTests.js
.eslintrc.js
.gitignore
package-lock.json
package.json
README
README.md
node_modules
```

```
131     };
132     initialLoad();
133     // eslint-disable-next-line
134   }, [dispatch]);
135
136   // PROJECT PHASES ==================================================>>>>>
137   const phasesIndex = [
138     { id: 1, name: 'Exploration' },
139     { id: 2, name: 'Construction' },
140     { id: 3, name: 'Operation' },
141     { id: 4, name: 'Closure' },
142     { id: 5, name: 'postClosure' },
143   ];
144   useEffect(() => {
145     const pick = [];
146     const startEnd = {};
147     if (getProjectPhasesDuration) {
148       getProjectPhasesDuration.forEach((item, i) => {
149         const { name } = phasesIndex.find((phase) => item?.id?.phase_id === phase?.id);
150         pick.push(name);
151         startEnd[`${name.toLowerCase()}startdate`] = moment(item?.attributes?.start_date);
152         startEnd[`${name.toLowerCase()}enddate`] = moment(item?.attributes?.end_date);
153       });
154       setInitial({
155         ...initial,
156         picked: pick,
157         ...startEnd,
158       });
159     }
160     // eslint-disable-next-line
161   }, [getProjectPhasesDuration, setInitial]);
162
163   // PROJECT DETAIL ==================================================>>>>>
164   useEffect(() => {
165     const initialLoad = () => {
166       const defaultData = initialData.projectTable.map((attributes) => ({
167         attributes,
168       }));
169       if (defaultData.length > 0) {
```

Now let’s move to the next part, project detail page **Table flow**

NOTE: Once you move to the table part remember one thing that we have 4 types of method used for each and every table. Create/Update/Delete/View Let me explain you with the screenshot.

Let’s move to the next flow how we are set the data into the table.

here is the code example of the setting data into the table: this is the view method use for single table. it means each table have separate code for setting the data.

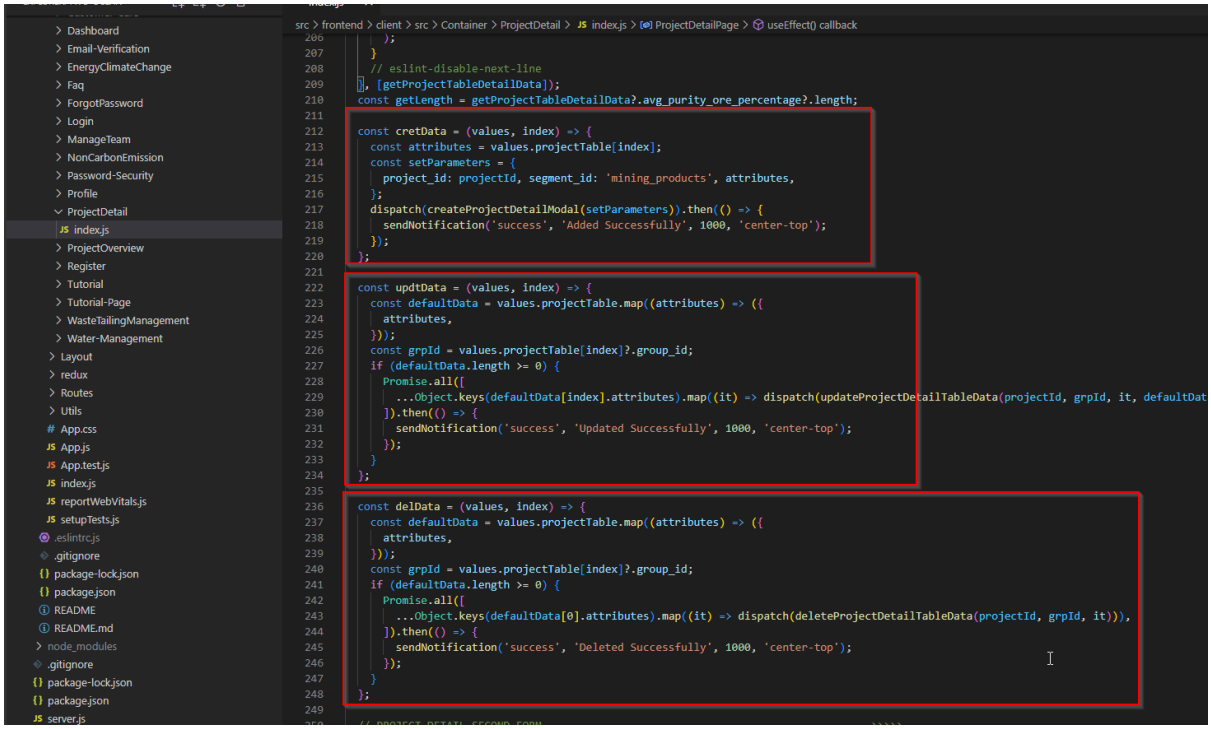
hope you understand the view method of table.

```
// PROJECT DETAIL ==================================================>>>>>
useEffect(() => {
  const initialLoad = () => {
    const defaultData = initialData.projectTable.map((attributes) => ({
      attributes,
    }));
    if (defaultData.length > 0) {
      Promise.all([
        ...Object.keys(defaultData[0].attributes).map((it) => dispatch(getProjectTableDetail(projectId, it))),
      ]);
    }
  };
  initialLoad();
  // eslint-disable-next-line
}, [dispatch, isOpen]);

useEffect(() => {
  const checkValue = Object.keys(getProjectTableDetailData).length !== 0;
  if (checkValue) {
    const setData = [];
    // eslint-disable-next-line array-callback-return
    Object.keys(getProjectTableDetailData).filter((item, i) => i === 3).map((data, i) => {
      const { length } = getProjectTableDetailData[data];
      // eslint-disable-next-line array-callback-return
      Array.apply(null, Array(length)).map(() => {}).forEach((it, k) => {
        let allData = null;
        Object.keys(getProjectTableDetailData).forEach((newData, l) => {
          allData = {
            avg_purity_ore_percentage: getProjectTableDetailData?.avg_purity_ore_percentage[k]?.attributes?.data_value,
            estimated_ore_resources: getProjectTableDetailData?.estimated_ore_resources[k]?.attributes?.data_value,
            extracted_product_id: getProjectTableDetailData?.extracted_product_id[k]?.attributes?.data_value,
            unit_of_physical_measure_id: getProjectTableDetailData?.unit_of_physical_measure_id[k]?.attributes?.data_value,
            group_id: getProjectTableDetailData?.unit_of_physical_measure_id[k]?.id?.group_id,
          };
        });
        setData.push(allData);
      });
    });
    setInitial(
      {
        ...initial,
        projectTable: setData,
      },
    );
  }
  // eslint-disable-next-line
```

Now let me explain you the rest of the others 3 methods **Create/Update/Delete**

Create Method is used for creating the new field on each of the table and same method used for update and delete.

As you can see the below listed image i have create the three different method for each table and we have almost 4 tables on project detail page. which means we have created 4 times all 4 methods. hope you will understand it.



How to generate tables for each material issue ?

If you do not have any information about material issue let me explain you once.

Once you open the any of the project from the dashboard you will see the sidebar. Under the sidebar you can see a lot of navigations Category Like:

Environment / Social / Governance / Economics

These all are the categories of material issue under these categories you can see some child navigation like:

Energy and Climate Change / Water Management / Waste and Tailings Management etc

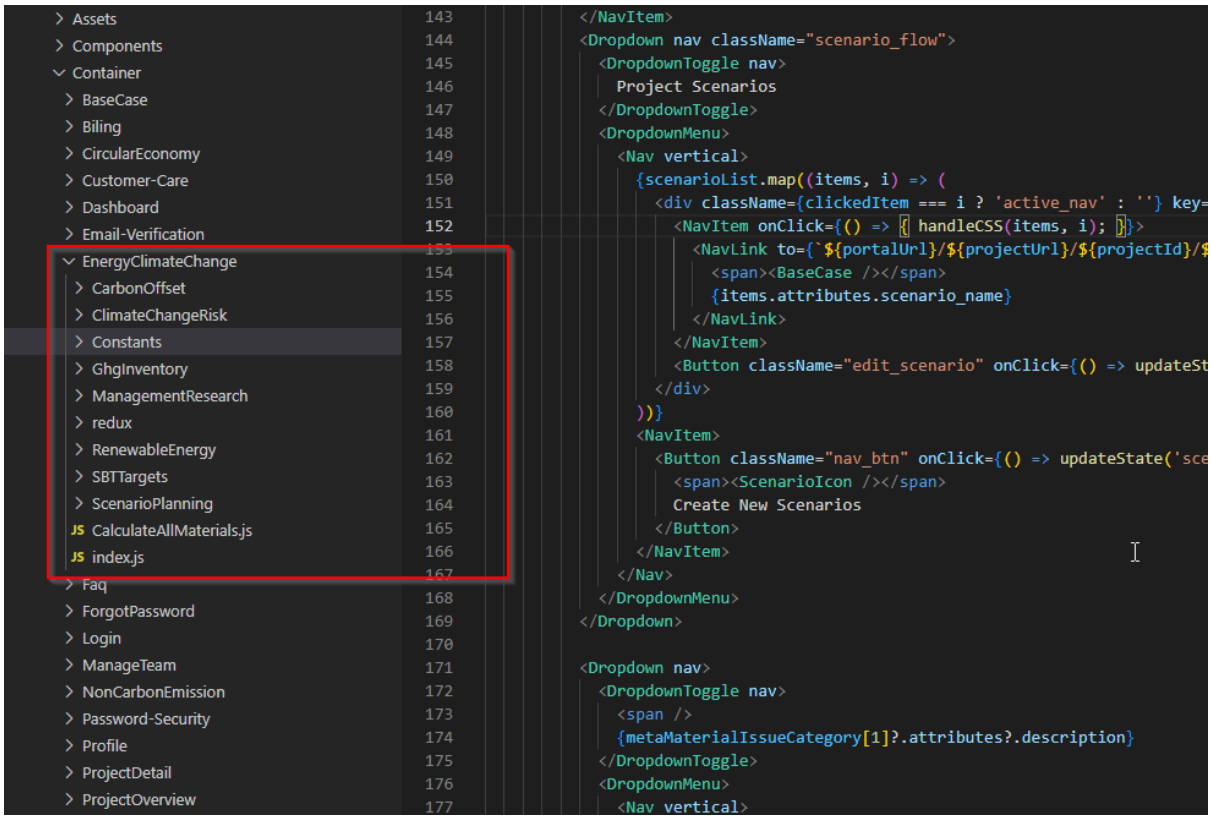
These are the material issues.

and under the material issue you will find some categories levels example:

GHG Inventory / Management Approach / Renewable Energy etc.

hope you understand the what is material issue.

Now let’s move to the material issue ID 2 folder let me explain you with the screenshot.

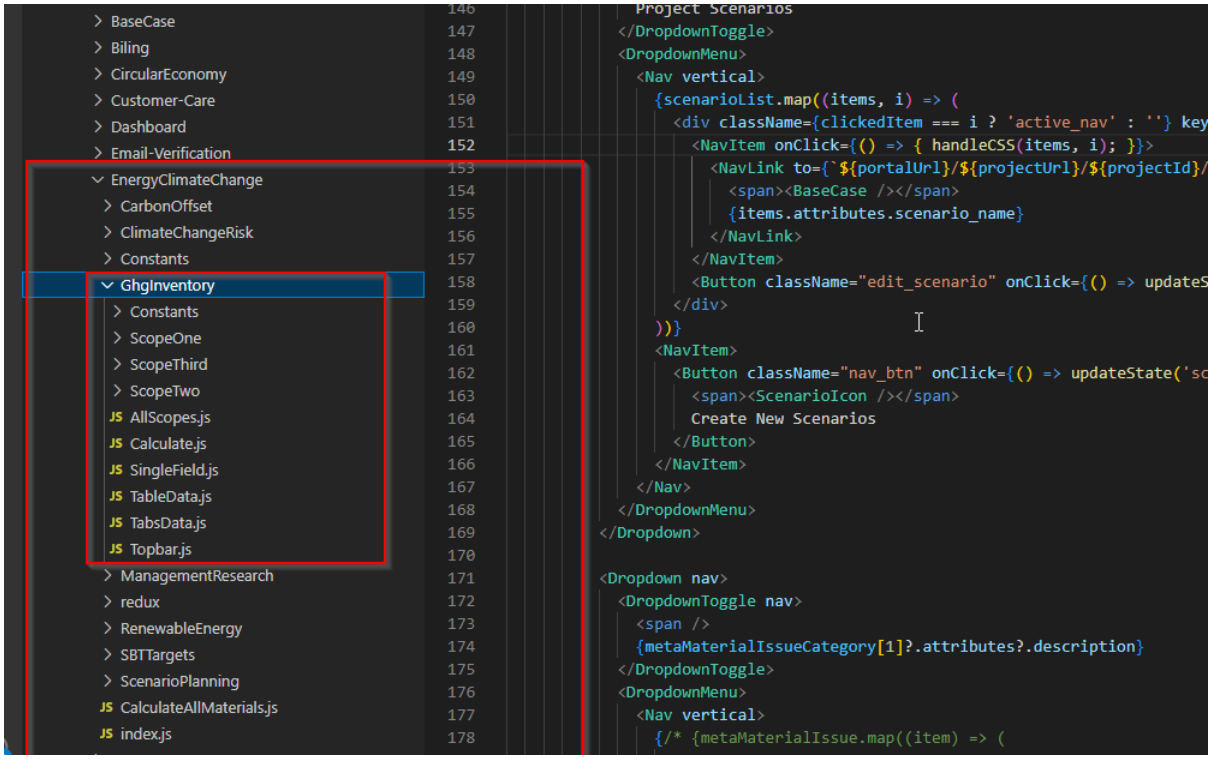


As you can see the above image we are in energy and climate folder which means we are going to access the material issue id 2.

Under the material issue id you can see a lot of categories available like:

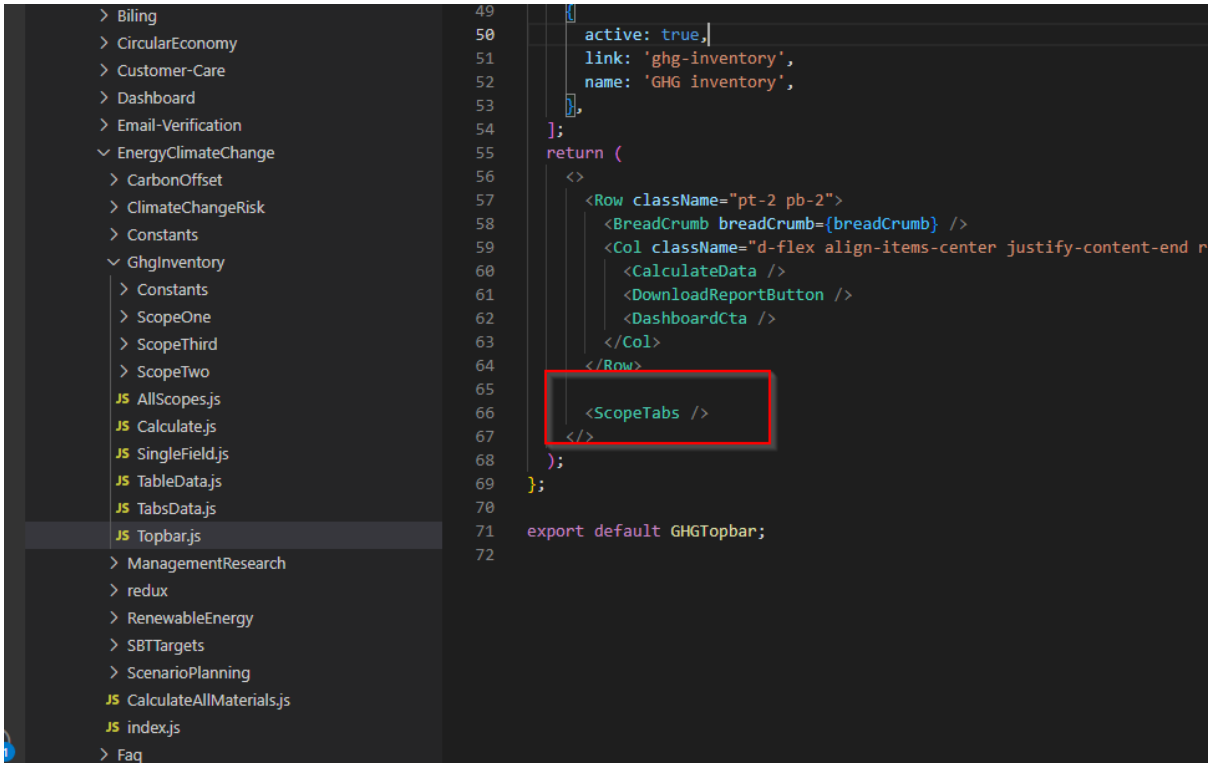
GhgInventory / ManagementResearch / RenewableEnergy / SBTTarget etc.

Once you enter to the **GHGInventory** category you can see a lot of folder under that.



Topbar.js is the parent of the ghg category. Under the topbar you will find the **Allscopes.js** file

All scope include the tabs parent (file name called **AllScopes.js**)



All scope is the parent of all the tabs when you visit the category level you will understand it better.

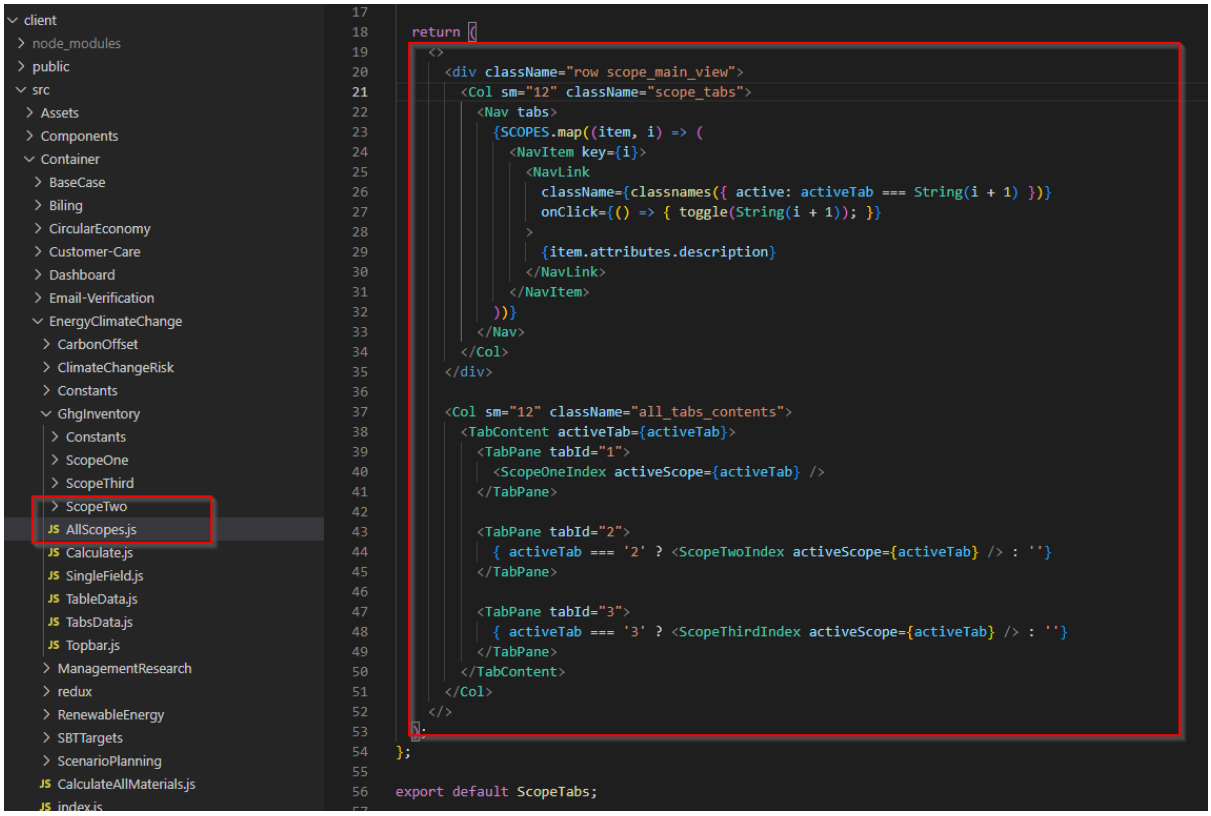
let me show you the All scope code.

Note: here you can see one more previous thing that we have also create a local json file and include here for making the tabs title.

Now why we are using the local json instead of using the API. this is just because of speed and optimization.

Also one more thing here you can see is we have created a three tabs and include the

Scope One / Scope Two and Scope Three On each of the tab:



Now let’s move to the next step of scopes 1, 2 and 3 method:

what is scope 1, 2 and 3

Please Note carefully this is the one of the most important topic. Once you open the Scope 1 **Index.html** File now you can see the Table generating method. This method is used for generating the whole table based on the segment id and local json of table headings.

Remember one thing you need the segment id and table headings local json for generating the table and you can also provide the title of the table by adding the s_name: which means segment name.

```
/* eslint-disable react/no-array-index-key */
/* eslint-disable max-len */
import React, { useState } from 'react';
import {
  Col, TabContent, TabPane, Nav, NavItem, NavLink,
} from 'reactstrap';
import PropTypes from 'prop-types';
import classnames from 'classnames';
import TabsData from '../TabsData';
import SCOPE_INSIDE_TABS from './Constants/ScopeInsideConstant';
import STATIONARY_COMBUSTION from './Constants/StationaryConstant';
import TRANSPORTATION_ONE from './Constants/TransportationOne';
import TRANSPORTATION_TWO from './Constants/TransportationTwo';
import FUGITIVE_EMISSION from './Constants/FugitiveEmissionConstant';
import PROCESS_EMISSION from './Constants/ProcessEmissionConstant';
import TRANSPORTATION_THREE from './Constants/TransporationThree';

const ScopeOneIndex = ({ activeScope }) => {
  const [activeTab, setActiveTab] = useState('1');
  const toggle = (tab) => {
    if (activeTab !== tab) setActiveTab(tab);
  };

  const allTabsLists = [
    [{ s_id: 1, s_data: STATIONARY_COMBUSTION }],
    [{
      s_id: 2, s_data: TRANSPORTATION_ONE, s_name: 'Road Transport -- Transportation Method - 1 (if fuel data is available)',
    }, { s_id: 3, s_data: TRANSPORTATION_TWO, s_name: 'Road Transport -- Transportation Method - 2 (if distance travelled data is avai
      s_id: 51, s_data: TRANSPORTATION_THREE, s_name: 'Non-Road Mobile Machinery',
    }],
    [{ s_id: 4, s_data: FUGITIVE_EMISSION }],
    [{ s_id: 5, s_data: PROCESS_EMISSION }],
  ];

  return (
    <>
      <div className="scope_inside_parent row">
        <Col sm="12" className="scope_inside_tabs">
          <Nav tabs>
            {SCOPE_INSIDE_TABS.map((item, i) => (
              <NavItem key={i}>
                <NavLink
                  className={classnames({ active: activeTab === String(i + 1) })}
                  onClick={() => { toggle(String(i + 1)); }}
                >
                  {item.attributes.description}
                </NavLink>
```

This is segment id

This is local JSON file

After creating the successfully local JSON just include here and add the segment id of the table. You can easily find the segment id of the table through the API or you can contact with the backend team regarding that.

Now let’s move to the next step now we have create a loop based on local json and segment id

as you can see the below image and we transfer the data into the **tabsData.js** file

```
Email-Verification
EnergyClimateChange
  CarbonOffset
  ClimateChangeRisk
  Constants
  GhgInventory
    Constants
    ScopeOne
    ScopeThird
    ScopeTwo
    AllScopes.js
    Calculate.js
    SingleField.js
    TableData.js
    TabsData.js
    Topbar.js
  ManagementResearch
  redux
  RenewableEnergy
  SBTTargets
  ScenarioPlanning
  CalculateAllMaterials.js
  index.js
Faq
ForgotPassword
Login
ManageTeam
NonCarbonEmission
Password-Security
Profile
ProjectDetail
ProjectOverview
Register
Tutorial
```

```
42                <NavLink
43                  className={classnames({ active: activeTab === String(i + 1) })}
44                  onClick={() => { toggle(String(i + 1)); }}
45                >
46                  {item.attributes.description}
47                </NavLink>
48              </NavItem>
49            ))}
50          </Nav>
51        </Col>
52      </div>
53
54      <Col sm="12">
55        <TabContent activeTab={activeTab}>
56          {allTabsLists.map((items, i) => (
57            <>
58              {activeTab === String(i + 1)
59                ? (
60                  <TabPane tabId={String(i + 1)} key={i}>
61                    <TabsData tab={activeTab} activeScope={activeScope} StatCombConst={items} />
62                  </TabPane>
63                )
64                : null}
65            </>
66          ))}
67        </TabContent>
68      </Col>
69    </>
70  );
71};
72
73ScopeOneIndex.propTypes = {
74  activeScope: PropTypes.string.isRequired,
75};
76
77export default ScopeOneIndex;
78
```

Now open the **TabsData.js** File

Under the tabs.js file we have used the all existing years and pass the data into the **TableData.js** file

now we have all the years and all data that we need for generating the table.

Please check the below listed screenshot once:

```
Dashboard
Email-Verification
EnergyClimateChange
  CarbonOffset
  ClimateChangeRisk
  Constants
  GhgInventory
    Constants
    ScopeOne
    ScopeThird
    ScopeTwo
    AllScopes.js
    Calculate.js
    SingleField.js
    TableData.js
    TabsData.js
    Topbar.js
  ManagementResearch
  redux
  RenewableEnergy
  SBTTargets
  ScenarioPlanning
  CalculateAllMaterials.js
  index.js
Faq
ForgotPassword
Login
ManageTeam
NonCarbonEmission
Password-Security
Profile
ProjectDetail
ProjectOverview
Register
Tutorial
Tutorial-Page
WasteTailingManagement
Water-Management
```

```
      />
    ))}
  </>
);

return (
  <>
    <div className="row team_list_parent">
      <Col sm="12" className="manage_team_list tabs_headlines">
        <Nav tabs>
          {projectPhaseNames.map((item) => (
            <NavItem
              key={item.link}
              onClick={() => { toggle(item.link, item.id.phase_id, item); }}
            >
              <NavLink className={classnames({ active: activeTab === item.link })}>
                {getPhasesNameWIthId({ id: item.id.phase_id })}
              </NavLink>
            </NavItem>
          ))}
        </Nav>
      </Col>
    </div>

    <TabContent activeTab={activeTab} className="manage-team-tabs">
      {projectPhaseNames.map((item) => (
        <TabPane tabId={item.link} key={item.link}>
          <Col sm="12" className="years_list pt-3">
            {fullYears.length > 0 && fullYears.map((items, key) => (
              <Button key={items} id={key} onClick={(e) => handleCSS(e, items)} className={items === active
            ))}
          </Col>
        </TabPane>
      ))}
    </TabContent>
    {activeYearValue && phaseIdValue ? renderCombustion(type) : ''}

  </>
);
};

TabsData.propTypes = {
  tab: PropTypes.string.isRequired,
  activeScope: PropTypes.string.isRequired,
```

```
Calculate.js
SingleField.js
TableData.js
TabsData.js
Topbar.js
ManagementResearch
redux
RenewableEnergy
SBTTargets
ScenarioPlanning
CalculateAllMaterials.js
index.js
aq
orgotPassword
ogin
ManageTeam
lonCarbonEmission
assword-Security
rofile
rojectDetail
rojectOverview
egister
utorial
utorial-Page
```

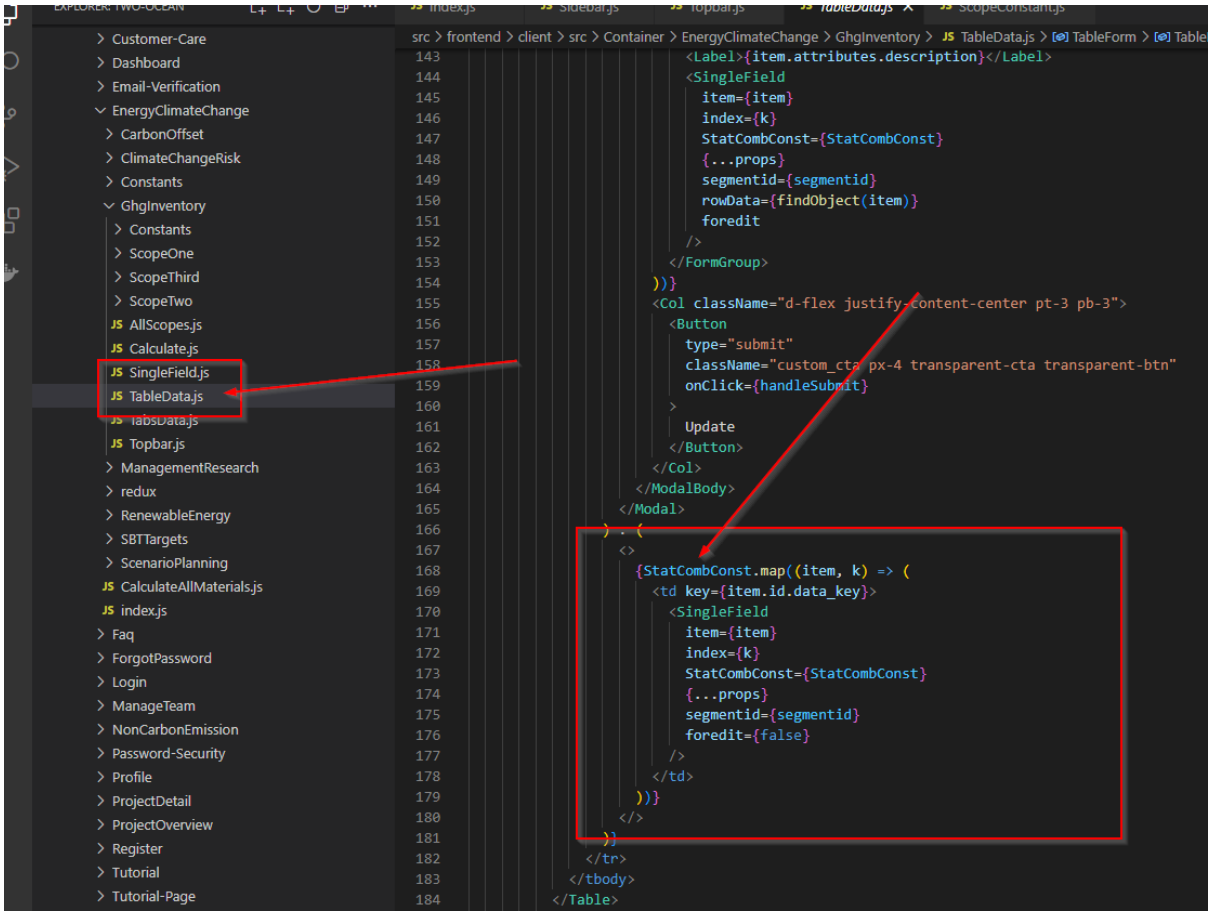
```
// years click function
const handleCSS = (e, year) => {
  setActiveYearValue(year);
  e.preventDefault();
};

const renderCombustion = () => (
  <>
    {StatCombConst.map((iidas) => (
      <TableData
        segmentName={iidas.s_name}
        onlyViewMode={iidas.only_view}
        segmentHeadings={iidas.s_heading}
        phaseIdValue={phaseIdValue}
        activeYearValue={activeYearValue}
        StatCombConst={iidas.s_data}
        segmentid={iidas.s_id}
        activeScope={activeScope}
      />
    ))}
  </>
);

return (
  <>
    <div className="row team_list_parent">
      <Col sm="12" className="manage_team_list tabs_headlines">
        <Nav tabs>
          {projectPhaseNames.map((item) => (
```

Now it’s time to go to the **tableData.js** File

This file is basically used for generating the tables. We have used the **WIthFormik method**

for generating the table dynamically.

NOTE: Remember one thing i am just giving you the instruction of these files you do not have to do any work on these files. your work is only dependent on Scope 1, 2, 3 Index.js file where you have to add the segment id and Local Json File, Rest of the table will be generated automatically



As you can see the we have imported a **singlefield** this component is basically used for generating the inputs or select options based on the requirement of the Table Heading.

let me explain you breifly:

When you create a local json of table heading you can see a **data_value_validation_id: 'list_single',**

in local json file which means this field is going to be a select option if you find something

data_value_validation_id: 'none', Which means you have to create the input text for that field.

Hope you got my point.

One More important thing that we have to add on the local json file is

Required / container and dependent_parent

You need to add these things as per your requirement.

```
      'unit_of_measure',
    ],
  },
  {
    attributes: {
      description: 'Consumption',
      data_value_validation_id: 'none',
      active: 1,
    },
    required: true,
    container: 'col-sm-2',
    references: {
      material_issue: {
        id: 2,
        link: 'http://10.0.1.108/api/v1/meta-material-issues/2',
      },
    },
    id: {
      material_issue_id: 2,
      material_issue_segment_id: 1,
      data_key: 'consumption_value',
    },
    link: 'http://10.0.1.108/api/v1/meta-material-issue-data-keys/%7B%22material_issue_id%22%3A2%2C%22material_issue_segment_id%22%3A1%2C%22data_key
    dependent_parent: [],
    dependent_data_keys: [],
  },
  {
    attributes: {
      description: 'Unit',
      data_value_validation_id: 'list_single',
      active: 1,
    },
    required: true,
    container: 'col-sm-2',
    references: {
      material_issue: {
        id: 2,
        link: 'http://10.0.1.108/api/v1/meta-material-issues/2',
      },
    },
    id: {
      material_issue_id: 2,
      material_issue_segment_id: 1,
      data_key: 'unit_of_measure',
    },
    link: 'http://10.0.1.108/api/v1/meta-material-issue-data-keys/%7B%22material_issue_id%22%3A2%2C%22material_issue_segment_id%22%3A1%2C%22data_key
    dependent_parent: [
      'fuel',
```

For example: If you find any object have some dependent value then to that value and add the dependent_parent Line and add the data_key of that object

Let me explain you with the example:

As you can see i have some dependent value on Technology used and the name of dependent is fuel and the data key is combustion_equipment_tech of the technology

```
49    },
50    {
51      attributes: {
52        description: 'Technology used',
53        data_value_validation_id: 'list_single',
54        active: 1,
55      },
56      required: true,
57      container: 'col-sm-2',
58      references: {
59        material_issue: {
60          id: 2,
61          link: 'http://10.0.1.108/api/v1/meta-material-issues/2',
62        },
63      },
64      id: {
65        material_issue_id: 2,
66        material_issue_segment_id: 1,
67        data_key: 'combustion_equipment_tech',
68      },
69      link: 'http://10.0.1.108/api/v1/meta-material-issue-data-keys/%7B%22material_issue_id%22%3A2%2C%22material_issue_segment_id%22%3A1%2C%22data_key%22%3A%22
70      dependent_parent: [
71        'size_combustion_equipment',
72      ],
73      dependent_data_keys: [
74        'fuel',
75      ],
76    },
77    {
78      attributes: {
```

now go to the fuel object and add the dependent_parent. Under the dependent parent add the data_key of the technology ‘combustion_equipment_tech’

```
JS TransportationOne.js
JS TransportationTwo.js
JS index.js
> ScopeThird
> ScopeTwo
JS AllScopes.js
JS Calculate.js
JS SingleField.js
JS TableData.js
JS TabsData.js
JS Topbar.js
ManagementResearch
redux
RenewableEnergy
SBTTargets
ScenarioPlanning
CalculateAllMaterials.js
index.js
Faq
ForgotPassword
Login
ManageTeam
NonCarbonEmission
Password-Security
Profile
```

```
75      ],
76    },
77    {
78      attributes: {
79        description: 'Fuel',
80        data_value_validation_id: 'list_single',
81        active: 1,
82      },
83      required: true,
84      container: 'col-sm-2',
85      references: {
86        material_issue: {
87          id: 2,
88          link: 'http://10.0.1.108/api/v1/meta-material-issues/2',
89        },
90      },
91      id: {
92        material_issue_id: 2,
93        material_issue_segment_id: 1,
94        data_key: 'fuel',
95      },
96      link: 'http://10.0.1.108/api/v1/meta-material-issue-data-keys/%7B%22material_issue_id%
97      dependent_parent: [
98        'combustion_equipment_tech',
99      ],
100     dependent_data_keys: [
101       'unit_of_measure',
102     ],
103   },
104   {
```

Which means fuel field is dependent on the technology.

What does dependent means:

It means when you change the technology field fuel field will be going to be changed also.

I hope you understand the flow of the table how table are generating and rest of the things.

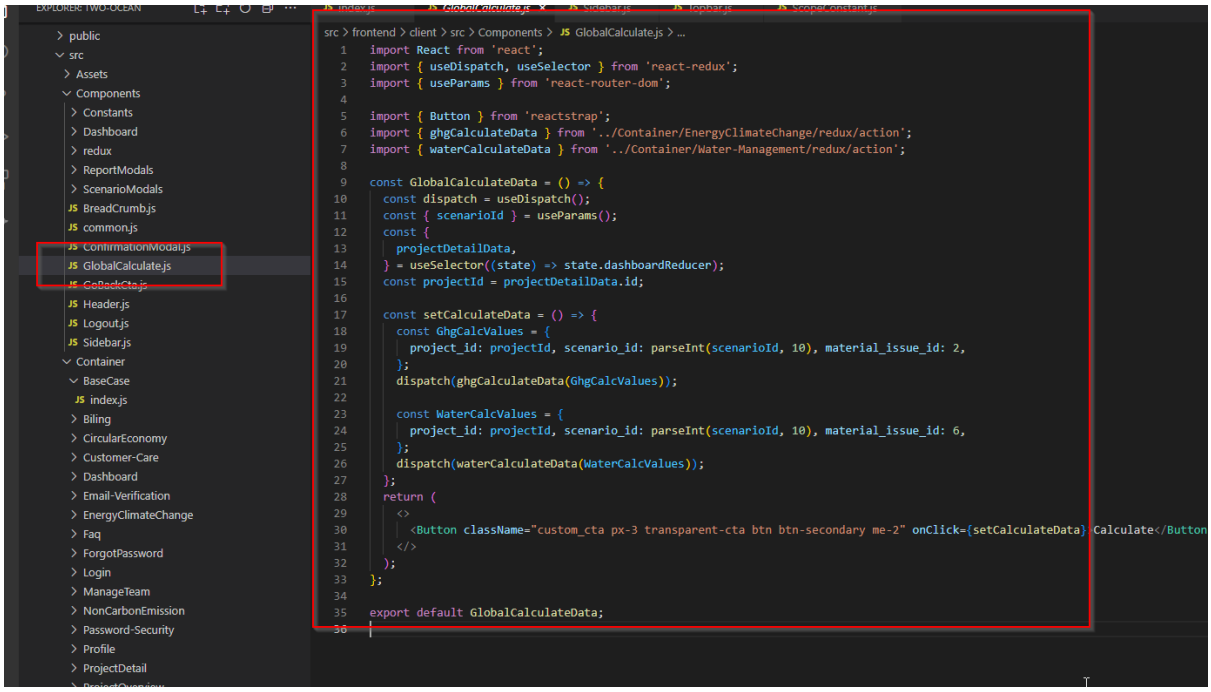
What is calculate method and how to use it ?

We have create a local and global calculate method for project

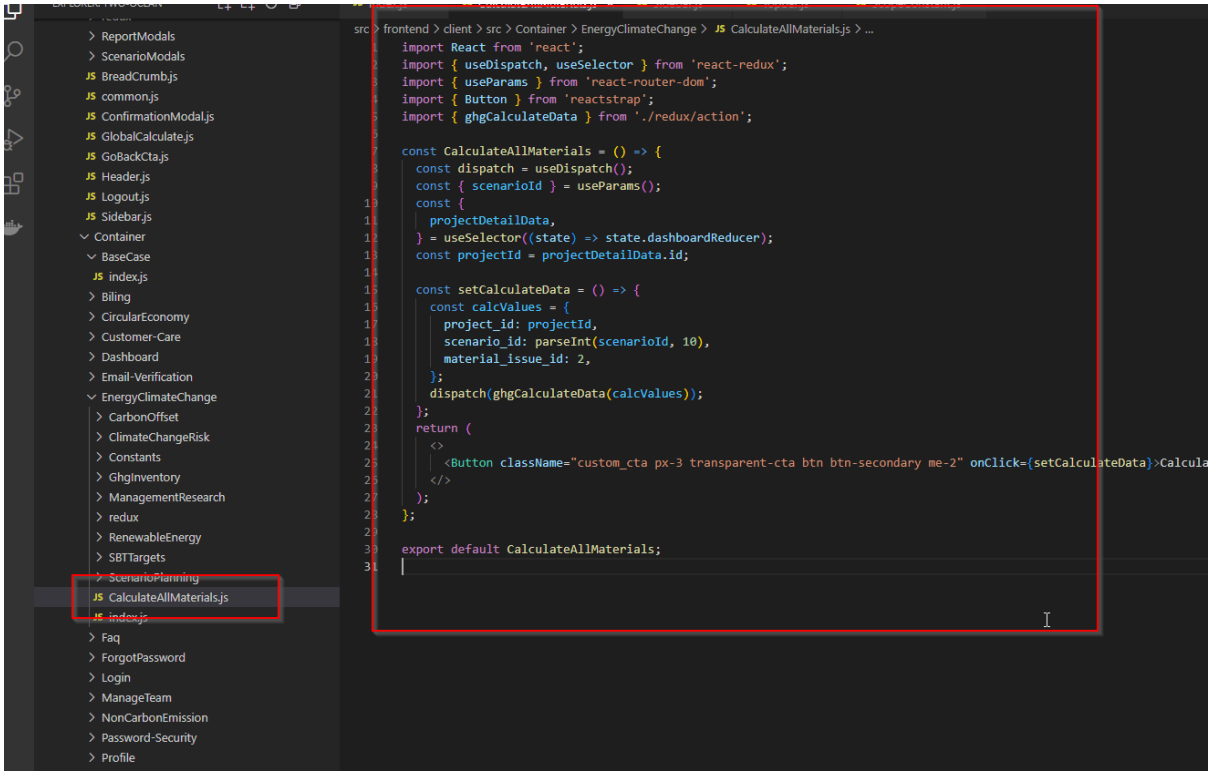
Local means you can calculate the one material issue at one time

Global means you can calculate all the material issue at once

You can easily find out those files on:



You can find the local calculate under the material issue folder:



Hope you will understand the flow of the project

Thanks.