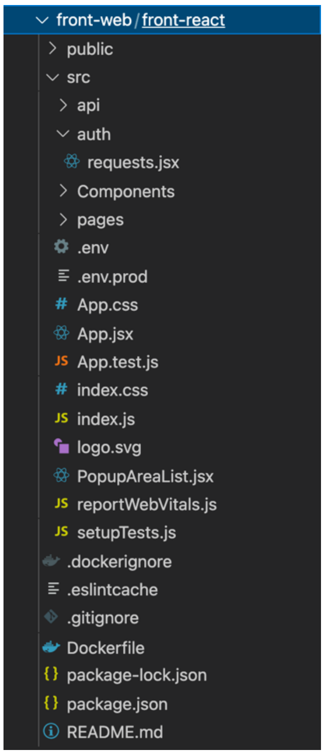
Documentation Front Web

1. Architecture of the code



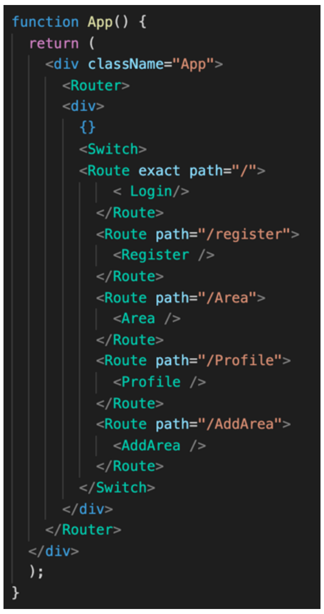
```
front-web / front-react
  > public
  v src
    > api
    v auth
        requests.jsx
    > Components
    > pages
    .env
    .env.prod
    App.css
    App.jsx
    App.test.js
    index.css
    index.js
    logo.svg
    PopupAreaList.jsx
    reportWebVitals.js
    setupTests.js
  .dockerignore
  .eslintcache
  .gitignore
  Dockerfile
  package-lock.json
  package.json
  README.md
```

The React projet here presents itself in different parts starting with

1.1 The Folder "auth "

In the folder auth contains all the requests needed to communicate with the API of the server. The principal folders and file that the API communicates with can be found in the folders Components and pages, while also getting the routes to each page from the file App.jsx.

The main parts of the website include the profile, authentication of the services and then the areas that can be created and then viewed on the area hub.



```
function App() {
  return (
    <div className="App">
      <Router>
      <div>
        {}
        <Switch>
        <Route exact path="/">
            < Login/>
          </Route>
          <Route path="/register">
            <Register />
          </Route>
          <Route path="/Area">
            <Area />
          </Route>
          <Route path="/Profile">
            <Profile />
          </Route>
          <Route path="/AddArea">
            <AddArea />
          </Route>
        </Switch>
      </div>
    </Router>
  </div>
  );
}
```

In front-web/front-react/src/App.jsx, we have defined our routes.

- "/" corresponds with the login page where the user can sign in with an existing account.
- "/Register" corresponds with the login page where the user can sign up with a new account.
- "/Area" corresponds with the main web page where the user will find the Area hub and the APK Download.
- "/Profile" Where the can sign up to all their services. (ex: Google/Discord or even Trello).
- "/AddArea" Where the user can create new areas while will then area on the Area Hub.

1.2 Package.json

Here contains all the dependencies used during the length of our projects.
Our project is running on react version “17.0.1”.

Removing all the default libraries given with React we have added:

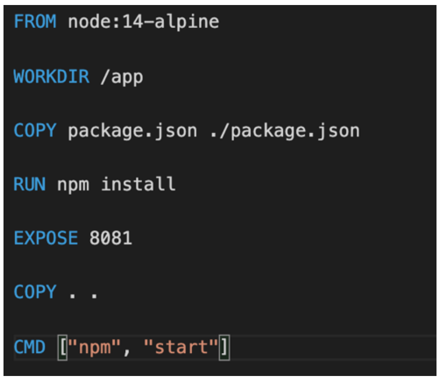
- “Material-UI” used to achieve our design language to match that of flutter.
- “react-icons” Not all icons that we needed could be found in Material-Ui, this library was used in for the services icons”
- “material-ui-popup-state” Takes care of the boilerplate for common Menu, Popover and Popper use cases.”
- “react-google-button” Simple Google sign in button for React that follows Google's style guidelines.
- “Axios” Promise based HTTP client for the browser.

1.3 Docker and the automatic building.

The Dockerfile is present in the project. It helps set up the environment for our react website.

Using the work directory /app.

Copying the package.json dependencies needed to run the site and installing them before exposing them to the necessary port before starting using the command “NPM START”

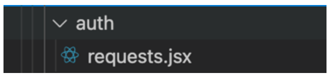


```
FROM node:14-alpine

WORKDIR /app

COPY package.json ./package.json

RUN npm install

EXPOSE 8081

COPY . .

CMD ["npm", "start"]
```

2. The code in detail.

2.1 API

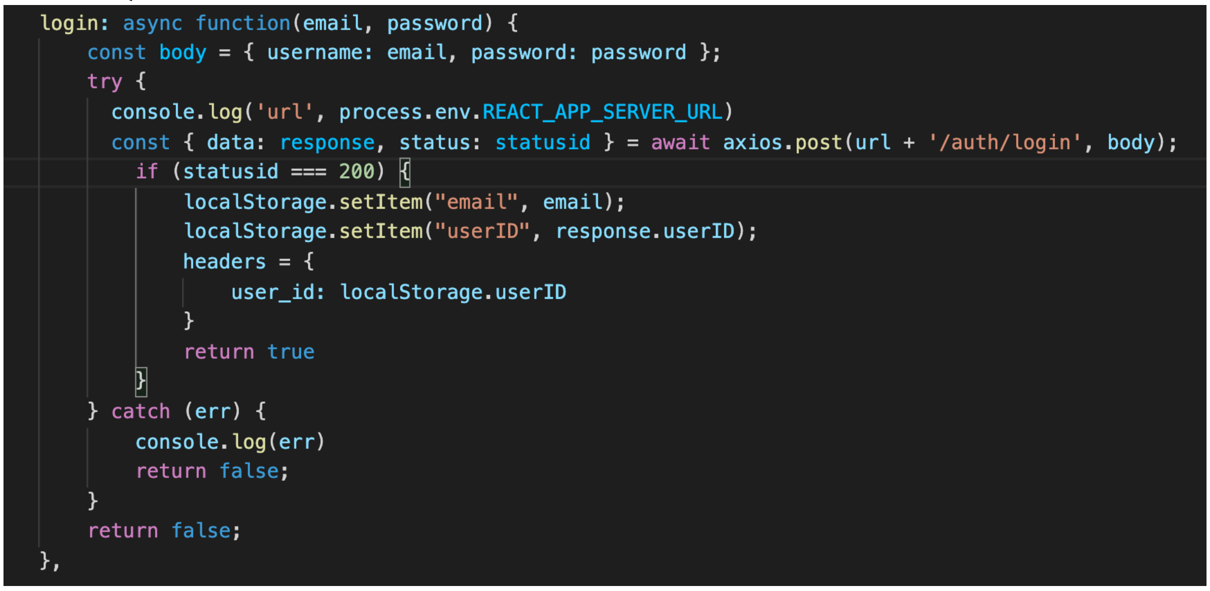
2.1.1 General



L’API is housed completely in auth in the file requests.jsx. All requests use Axios that connect to the server using the url “https://area.gen-host.fr” stored in the Docker-Compose as an environment variable.

2.1.2 3 example of the requests:

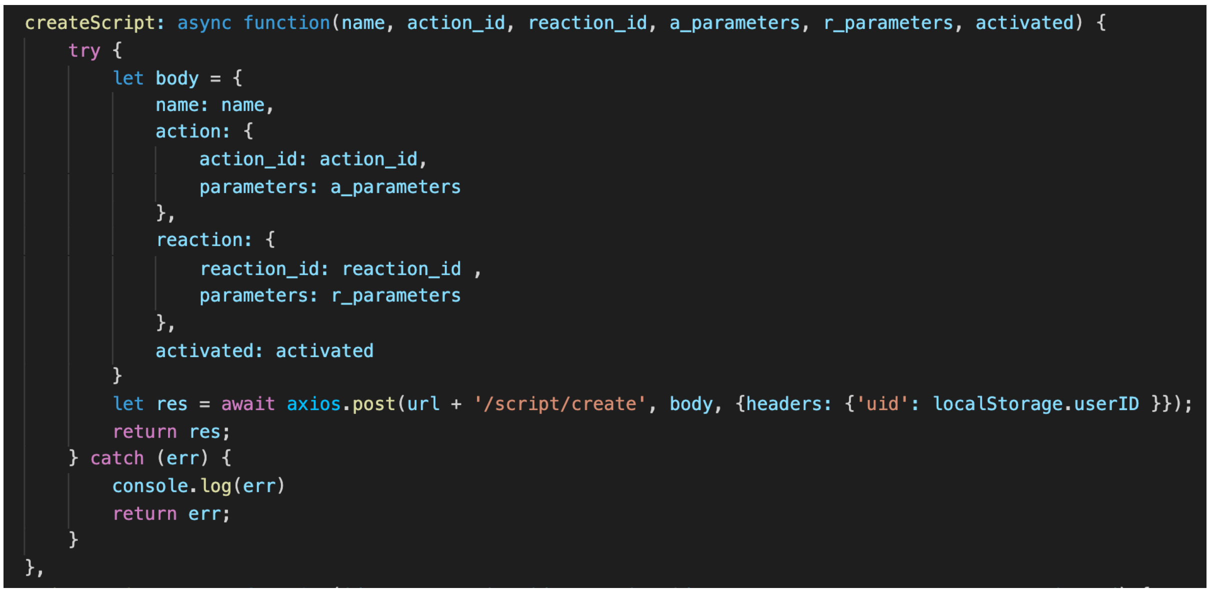
A GET REQUEST:



```
login: async function(email, password) {
    const body = { username: email, password: password };
    try {
      console.log('url', process.env.REACT_APP_SERVER_URL)
      const { data: response, status: statusid } = await axios.post(url + '/auth/login', body);
        if (statusid === 200) {
            localStorage.setItem("email", email);
            localStorage.setItem("userID", response.userID);
            headers = {
                user_id: localStorage.userID
            }
            return true
        }
    } catch (err) {
        console.log(err)
        return false;
    }
    return false;
},
```

Here we generate the URL from the environments variables in the docker-compose and send the request using a async function that gets an username and password from the text fields given to the user and we send is information using the header to allows us to identify the user. It returns true on success and console error message and false when it fails

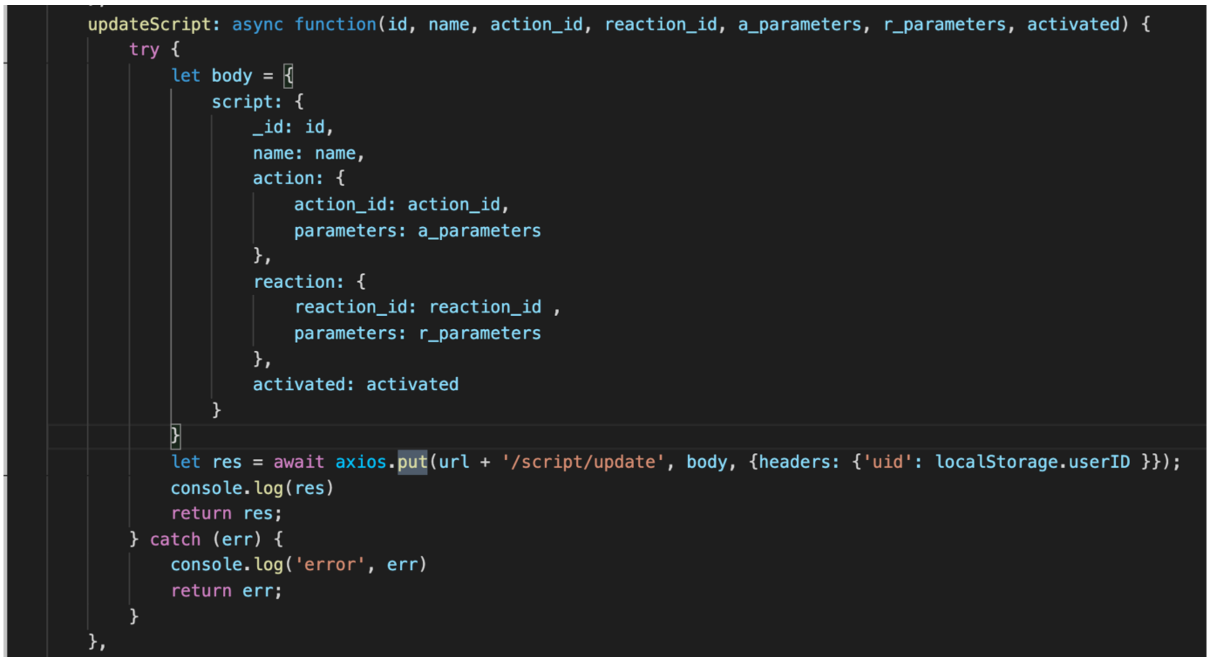
A POST REQUEST:



```
createScript: async function(name, action_id, reaction_id, a_parameters, r_parameters, activated) {
    try {
        let body = {
            name: name,
            action: {
                action_id: action_id,
                parameters: a_parameters
            },
            reaction: {
                reaction_id: reaction_id ,
                parameters: r_parameters
            },
            activated: activated
        }
        let res = await axios.post(url + '/script/create', body, {headers: {'uid': localStorage.userID }});
        return res;
    } catch (err) {
        console.log(err)
        return err;
    }
},
```

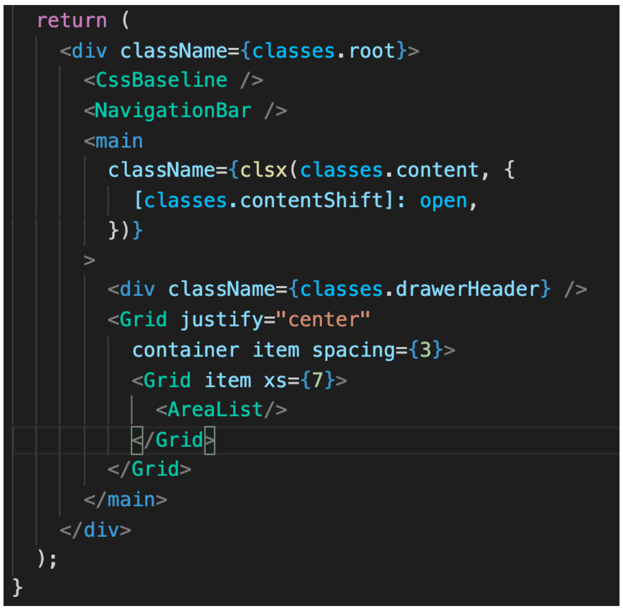
Here we are asking the server to post a new area to the area hub. it takes the specific parameters name, actions { action_id and the necessary parameters }, reactions { action_id and the necessary parameters }, and if the area should be activated once created" it then use the same url as the get request and returns a result or a console error in this try, catch loop.

A PUT REQUEST.


```
updateScript: async function(id, name, action_id, reaction_id, a_parameters, r_parameters, activated) {
    try {
        let body = {
            script: {
                _id: id,
                name: name,
                action: {
                    action_id: action_id,
                    parameters: a_parameters
                },
                reaction: {
                    reaction_id: reaction_id ,
                    parameters: r_parameters
                },
                activated: activated
            }
        }
        let res = await axios.put(url + '/script/update', body, {headers: {'uid': localStorage.userID }});
        console.log(res)
        return res;
    } catch (err) {
        console.log('error', err)
        return err;
    }
},
```

The same modifications for POST but this is for modifying the already given information for an AREA.

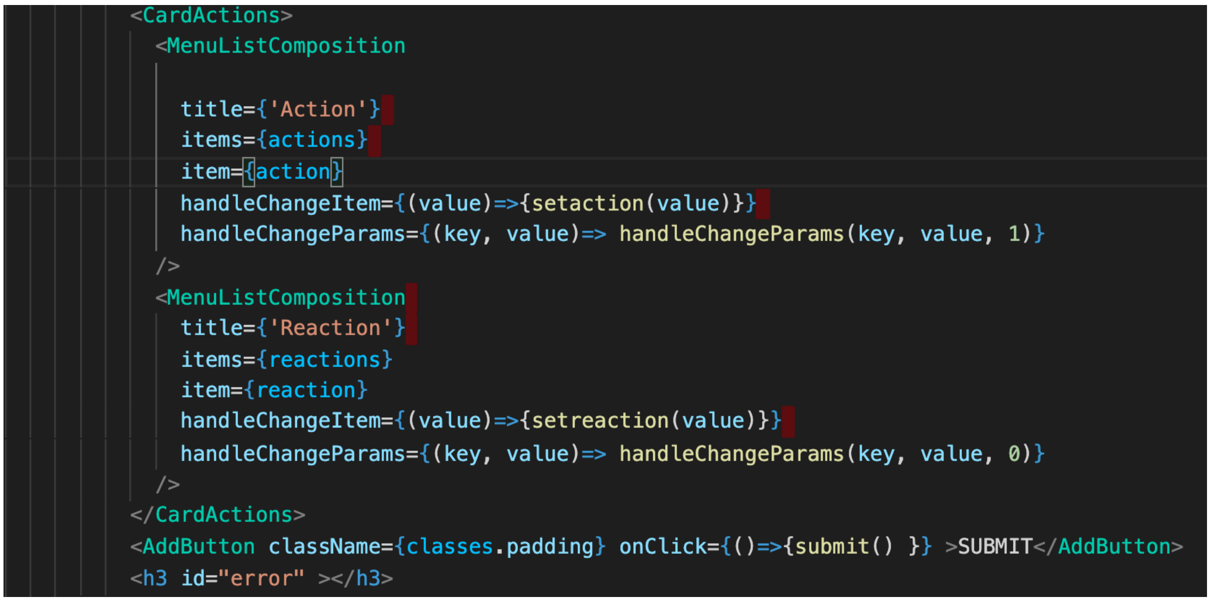
2.1.3. An Area Creation front end


```
  return (
    <div className={classes.root}>
      <CssBaseline />
      <NavigationBar />
      <main
        className={clsx(classes.content, {
          [classes.contentShift]: open,
        })}
      >
        <div className={classes.drawerHeader} />
        <Grid justify="center"
          container item spacing={3}>
          <Grid item xs={7}>
            <AreaList/>
          </Grid>
        </Grid>
      </main>
    </div>
  );
}
```

It starts in pages/Addarea.jsx. Here is the information showed to the user.

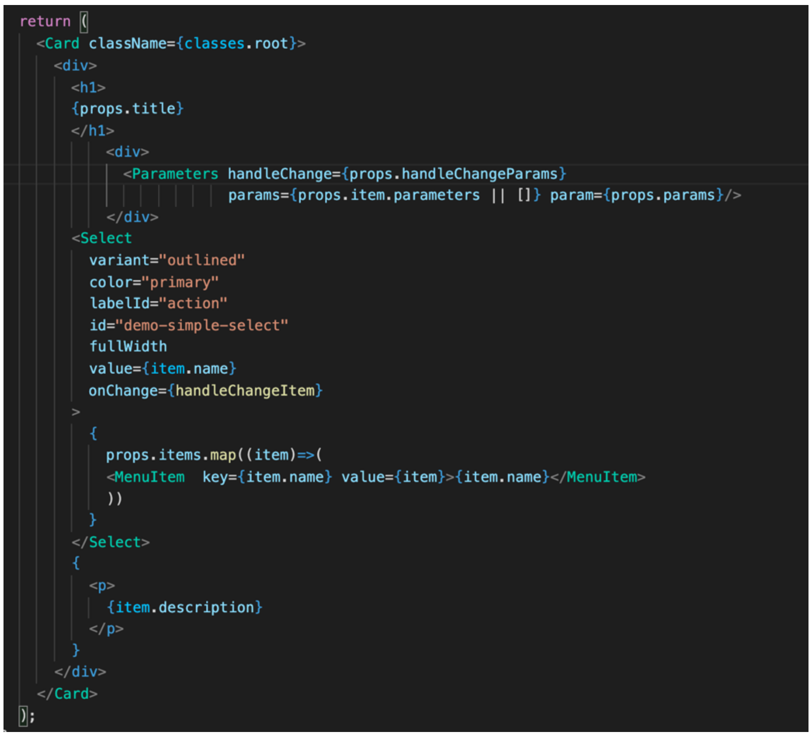
The AreaList component which hold the add area component stored in "./Components/AreaList."

Here in Components/AreaList.jsx you will find Add Area card with a basic login text field and down below you'll find this bit of code:



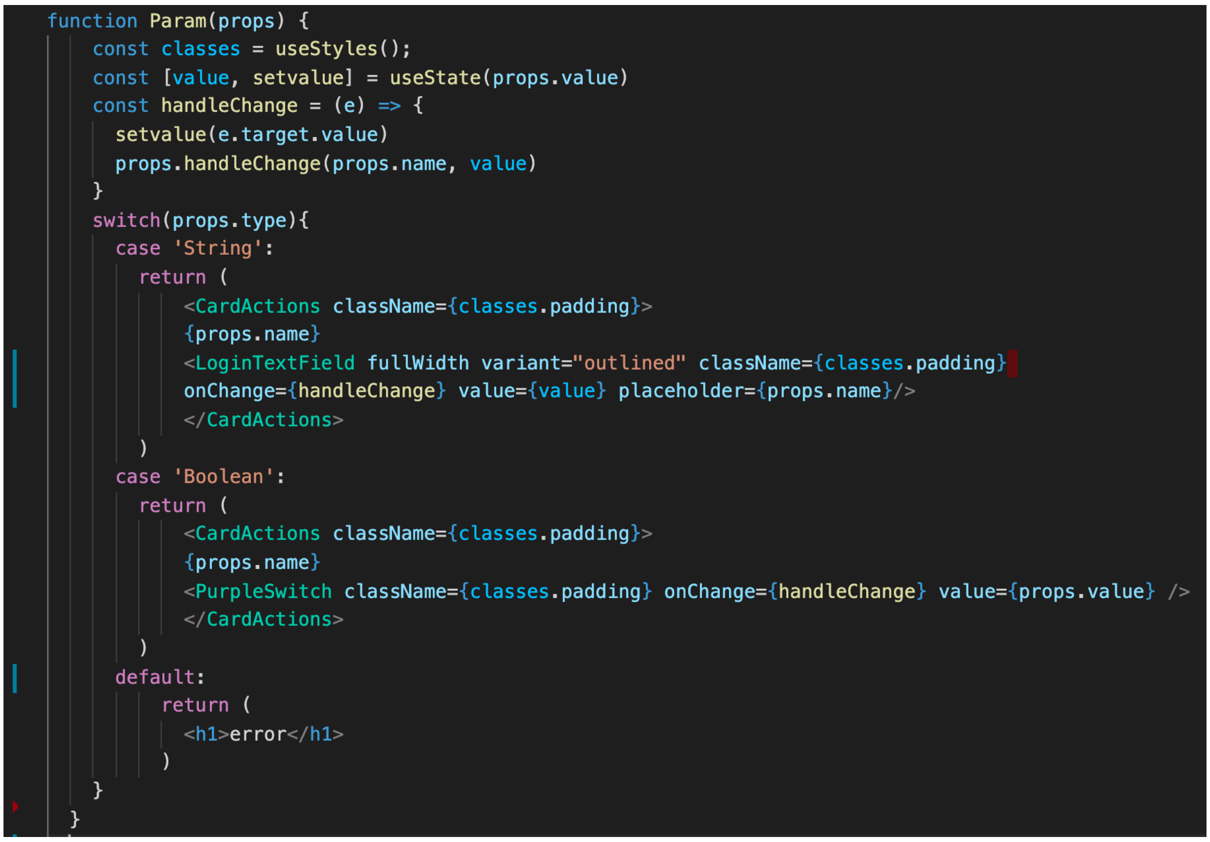
```jsx
<CardActions>
  <MenuListComposition

    title={'Action'}
    items={actions}
    item={action}
    handleChangeItem={(value)=>{setaction(value)}}
    handleChangeParams={(key, value)=> handleChangeParams(key, value, 1)}
  />
  <MenuListComposition
    title={'Reaction'}
    items={reactions}
    item={reaction}
    handleChangeItem={(value)=>{setreaction(value)}}
    handleChangeParams={(key, value)=> handleChangeParams(key, value, 0)}
  />
</CardActions>
<AddButton className={classes.padding} onClick={()=>{submit() }} >SUBMIT</AddButton>
<h3 id="error" ></h3>
```

MenuListComposition stores this bit of code which holds the parameters Component stored in ./Component/Params.jsx and a Material-Ui Select Components for the specific conditions for the Action and REActions.



```jsx
return (
  <Card className={classes.root}>
    <div>
      <h1>
      {props.title}
      </h1>
        <div>
          <Parameters handleChange={props.handleChangeParams}
                      params={props.item.parameters || []} param={props.params}/>
        </div>
      <Select
        variant="outlined"
        color="primary"
        labelId="action"
        id="demo-simple-select"
        fullWidth
        value={item.name}
        onChange={handleChangeItem}
      >
        {
          props.items.map((item)=>(
          <MenuItem  key={item.name} value={item}>{item.name}</MenuItem>
          ))
        }
      </Select>
      {
        <p>
          {item.description}
        </p>
      }
    </div>
  </Card>
);
```

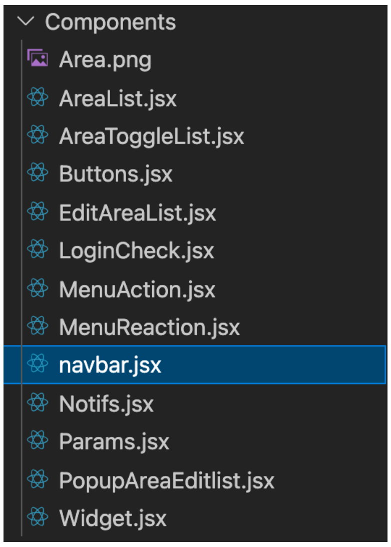
Params.jsx is where the parameters are stored for each Action and REAction if a text field or switch is needed to define specific data before making the AREA button labelled submit.

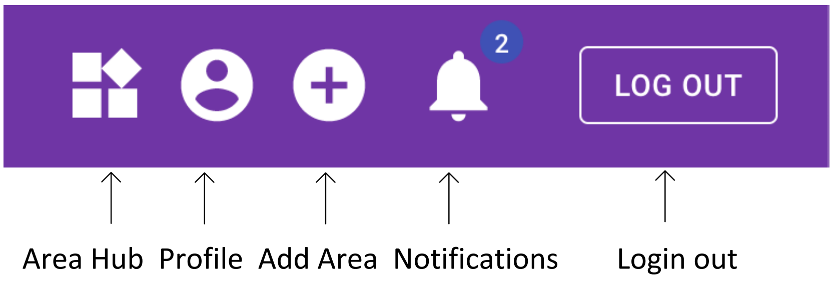


```
function Param(props) {
    const classes = useStyles();
    const [value, setvalue] = useState(props.value)
    const handleChange = (e) => {
      setvalue(e.target.value)
      props.handleChange(props.name, value)
    }
    switch(props.type){
      case 'String':
        return (
            <CardActions className={classes.padding}>
            {props.name}
            <LoginTextField fullWidth variant="outlined" className={classes.padding}
            onChange={handleChange} value={value} placeholder={props.name}/>
            </CardActions>
        )
      case 'Boolean':
        return (
            <CardActions className={classes.padding}>
            {props.name}
            <PurpleSwitch className={classes.padding} onChange={handleChange} value={props.value} />
            </CardActions>
        )
      default:
          return (
            <h1>error</h1>
          )
    }
  }
```

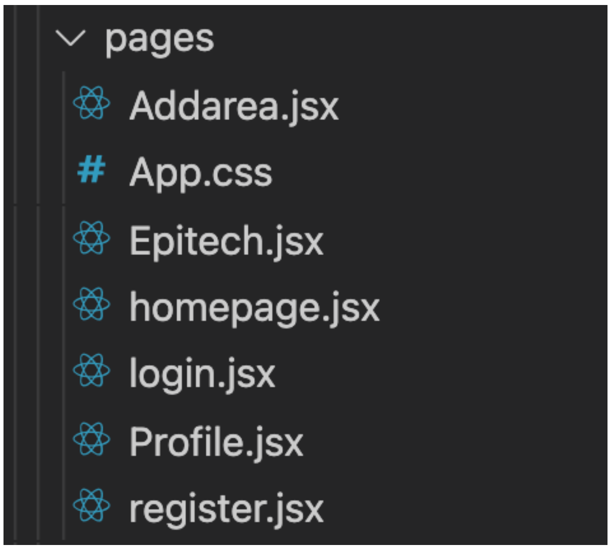
2.2 The navigation bar

2.2.1. The components


Components
Area.png
AreaList.jsx
AreaToggleList.jsx
Buttons.jsx
EditAreaList.jsx
LoginCheck.jsx
MenuAction.jsx
MenuReaction.jsx
navbar.jsx
Notifs.jsx
Params.jsx
PopupAreaEditlist.jsx
Widget.jsx



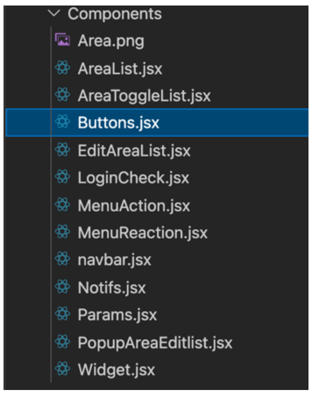
Here the navbar is used throughout the project using icons from the Material-Ui icon library to indicate what the user needs to click in order to go from all pages indicated here. It can be found inputed at the top of every page.

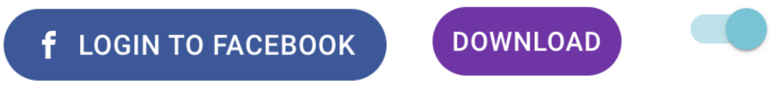

pages
Addarea.jsx
App.css
Epitech.jsx
homepage.jsx
login.jsx
Profile.jsx
register.jsx

Here are all the pages the navbar goes to.

Important information: Epitech.jsx is the Area Hub/ homepage of the website.

2.2.1. The constants


Components
Area.png
AreaList.jsx
AreaToggleList.jsx
Buttons.jsx
EditAreaList.jsx
LoginCheck.jsx
MenuAction.jsx
MenuReaction.jsx
navbar.jsx
Notifs.jsx
Params.jsx
PopupAreaEditlist.jsx
Widget.jsx

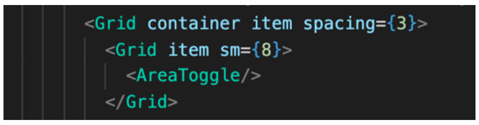


In ./Components/Buttons.jsx you'll find all the custom constant buttons/Text Field/ Switch which use the colors specified in the graphic design documentation. Examples can be found above.

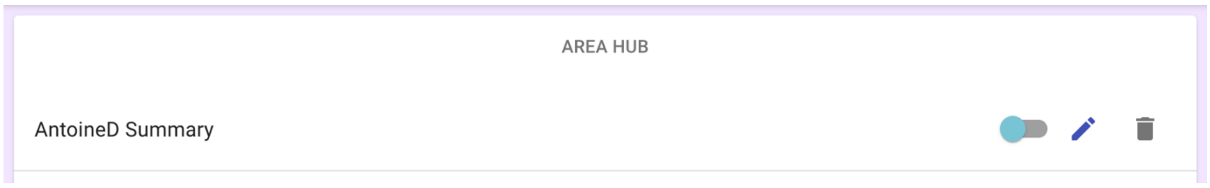
2.2.2. The screens

AREA HUB:

In ./pages/Epitech.jsx you'll find the area hub which contains the AreaToggle{./Components/AreaToggleList } component which houses the basic information seen below. In AreaToggleList.jsx you have a specific component PopoverPopupEDIT {'./Component/PopupAreaEditlist'} which works at creating the popup window when clicking on this icon:



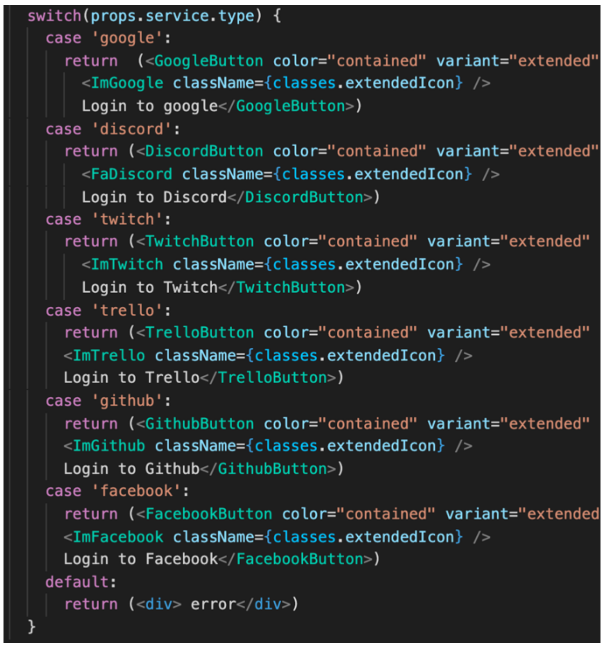
```
<Grid container item spacing={3}>
  <Grid item sm={8}>
    <AreaToggle/>
  </Grid>
```



AREA HUB

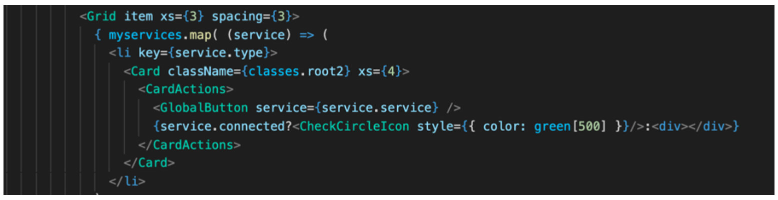
AntoineD Summary

PROFILE:



```
switch(props.service.type) {
  case 'google':
    return  (<GoogleButton color="contained" variant="extended"
      <ImGoogle className={classes.extendedIcon} />
      Login to google</GoogleButton>)
  case 'discord':
    return (<DiscordButton color="contained" variant="extended"
      <FaDiscord className={classes.extendedIcon} />
      Login to Discord</DiscordButton>)
  case 'twitch':
    return (<TwitchButton color="contained" variant="extended"
      <ImTwitch className={classes.extendedIcon} />
      Login to Twitch</TwitchButton>)
  case 'trello':
    return (<TrelloButton color="contained" variant="extended"
    <ImTrello className={classes.extendedIcon} />
    Login to Trello</TrelloButton>)
  case 'github':
    return (<GithubButton color="contained" variant="extended"
    <ImGithub className={classes.extendedIcon} />
    Login to Github</GithubButton>)
  case 'facebook':
    return (<FacebookButton color="contained" variant="extended
    <ImFacebook className={classes.extendedIcon} />
    Login to Facebook</FacebookButton>)
  default:
    return (<div> error</div>)
}
```

In ./pages/Profile.jsx you'll find the Profile which contains all the service logins. The buttons rely on a switch case and a map loop to know which services are connected and which are know as seen here.



```
<Grid item xs={3} spacing={3}>
  { myservices.map( (service) => (
    <li key={service.type}>
      <Card className={classes.root2} xs={4}>
        <CardActions>
          <GlobalButton service={service.service} />
          {service.connected?<CheckCircleIcon style={{ color: green[500] }}/>:<div></div>}
        </CardActions>
      </Card>
    </li>
```