Final

anonymous marking enabled

Submission date: 16-Jul-2023 11:22AM (UTC+0100)
Submission ID: 210273212
File name: Final.txt (19.96K)
Word count: 1323
Character count: 13106

Home Page

```
//importing modules
import React, { useState, useEffect } from 'react'
import './Main.css'
import { Button } from 'primereact/button';
import { Dialog } from 'primereact/dialog';
import { Paginator } from 'primereact/paginator';
import axios from 'axios';
import Popup from 'reactjs-popup';
import { useNavigate } from 'react-router-dom';
//functions declaration
var baseurl = 'http://localhost:8000/Task/task-data/'
export const Main = () => {
    const [visible, setVisible] = useState(false)
    const [data, setData] = useState([])
    const [name, setName] = useState('')
    const [isError, setIsError] = useState('')
    const [assignedDate, setAssignedDate] = useState()
    const [dueDate, setDueDate] = useState()
    const [level, setLevel] = useState()
    const [description, setDescription] = useState()
    const [input, setInput] = useState('')
    const [week, setWeek] = useState('')
```

```
const [form, setForm] = useState('')
const [openEdit,SetOpenEdit] = useState(false)
const [openDialog,SetOpenDialog] = useState(false)
const [idata,setIdata] = useState([])
const navigate = useNavigate()
async function getTaskData() {
    try {
        await axios.get(baseurl)
            .then((response) => {
                setData(response.data)
                console.log(response.data)
            })
    } catch (error) {
        console.log(error.message)
        setIsError(error.message)
    }
}
useEffect(() => {
    getTaskData();

}, [])

const Addhandler = () => {
```

```
    try {
        axios.post(`${baseurl}`, {
            task_name: name,
            task_assigned_date: assignedDate,
            task_due_date: dueDate,
            priority_level: level,
            description: description

        })
            .then((response) => {
                console.log(response.data)
                // setData(response.data)
                getTaskData();

            })
        SetOpenDialog(false)
    } catch (error) {
        console.log(error.message)
        setIsError(error.message)
    }
}

const deleteHandler = (id) => {
```

```
        try {
            var result = window.confirm("Are you sure ..? \n Do You Want to delte this
task..?");
            if (result) {
                axios.delete(`${baseurl}${id}`)
                    .then(() => {
                        getTaskData();


                    })
                    alert('task deleted')
            }
        } catch (error) {
            console.log(error)
            setIsError(error.message)
        }
    }


    const searchHandler = (value) => {
        try {
            axios.get(`${baseurl}filter/${value}`)
                .then((response) => {
                    console.log(response.data)
```

```
                // fetchHandler(week)
                setData(response.data)
                // getTaskData()
            })
    } catch (error) {
        console.log(error)
        setIsError(error.message)
    }
}

const editHandler = (id) => {
    console.log(idata)
    try {
        axios.put(`${baseurl}${id}`,{
            task_name: idata.task_name,
            task_assigned_date: idata.task_assigned_date,
            task_due_date: idata.task_due_date,
            priority_level: idata.priority_level,
            description: idata.description,
            status:idata.status,
        })
        .then((response) => {
            console.log(response.data)
```

```
            // setData(response.data)
            getTaskData();

        })
        SetOpenEdit(false);
    } catch (error) {

    }
}

const getForm = (id) => {
    console.log(id, '**************')
    console.log("clicked...!")
    setVisible(true)
    SetOpenEdit(true)
    try {
        axios.get(`${baseurl}${id}`)
        .then((response) => {
            console.log(response.data)
            setIdata(response.data)

        })
    } catch (error) {
```

```
    }
}

const taskData = (task) => {
    const tdata = {
        task_name: task.task_name,
        task_assigned_date:task.task_assigned_date,
        task_due_date:task.task_due_date,
        priority_level:task.priority_level,
        description:task.description
    }
    navigate("/task",{state : tdata});
}

const fetchHandler = (fetch, filter) => {
    try {
        if (filter === "") {
            console.log(fetch)
            axios.get(`${baseurl}fetch/${fetch}`)
                .then((response) => {
                    console.log(response.data)
                    // searchHandler(input)
```

```
            setData(response.data)
            // getTaskData()
        })
}
else if (fetch === "") {
    axios.get(`${baseurl}filter/${filter}`)
        .then((response) => {
            console.log(response.data)

            // fetchHandler(week)
            setData(response.data)
            // getTaskData()
        })
}
else {
    console.log(fetch, filter)
    axios.get(`${baseurl}full-search/${fetch}/${filter}`)
        .then((response) => {
            console.log(response.data)
            // searchHandler(input)
            setData(response.data)
            // getTaskData()
        })
```

```
            }
        } catch (error) {
            console.log(error)
            setIsError(error.message)
        }
    }
    const addfooterContent = (

        <div>
            <Button label="Cancel" icon="pi pi-times" onClick={() => SetOpenDialog(false)}
className="p-button-text" />
            <Button label="Add" icon="pi pi-check" onClick={Addhandler} autoFocus />
        </div>
    );

    const editfooterContent = (

        <div>
            <Button label="Cancel" icon="pi pi-times" onClick={() => SetOpenEdit(false)}
className="p-button-text" />
            <Button label="Edit" icon="pi pi-check" onClick={() => editHandler(idata.id)}
autoFocus />
        </div>
```

```
    );
//Rendering to the web page
return (
        <>
            <Dialog header="Add Task"  visible={openDialog}  style={{ width: '50%' }}
onHide={() => SetOpenDialog(false)} footer={addfooterContent}>
                <hr />
                <form className='form-horizontal' style={{ width: '80%', marginLeft: '10%'
}}>
                    <label htmlFor='add'> Task Name *</label>
                    <input type='text' placeholder='Enter Task Name' onChange={(e) =>
setName(e.target.value)} />
                    <label htmlFor='add'> Assigned date *</label>
                    <input type='date' placeholder='Enter Assigned date' onChange={(e) =>
setAssignedDate(e.target.value)} />
                    <label htmlFor='add'> Due Date *</label>
                    <input type='date' placeholder='Enter Due Date' onChange={(e) =>
setDueDate(e.target.value)} />
                    <label htmlFor='add'> Level *</label>
                    <input type='text' placeholder='Enter Level' onChange={(e) =>
setLevel(e.target.value)} />
                    <label htmlFor='add'> Description *</label>
```

```
                <textarea type='text' placeholder='Enter Description' onChange={(e) =>
setDescription(e.target.value)} />
                </form>
            </Dialog>
            <Dialog header="Edit Task"  visible={openEdit} data={idata} style={{ width:
'50%' }} onHide={() => SetOpenEdit(false)} footer={editfooterContent}>
                <hr />
                <form className='form-horizontal' style={{ width: '80%', marginLeft: '10%'
}}>
                    <label htmlFor='edit'> Task Name *</label>
                    <input type='text' placeholder='Enter Task Name'
defaultValue={idata.task_name} onChange={(e) => setIdata((prevValues)=>
({...prevValues,task_name:e.target.value}))} />
                    <label htmlFor='edit'> Assigned date *</label>
                    <input type='date' placeholder='Enter Assigned date'
defaultValue={idata.task_assigned_date} onChange={(e) => setIdata((prevValues)=>
({...prevValues,task_assigned_date:e.target.value}))}/>
                    <label htmlFor='edit'> Due Date *</label>
                    <input type='date' placeholder='Enter Due Date'
defaultValue={idata.task_due_date} onChange={(e) => setIdata((prevValues)=>
({...prevValues,task_due_date:e.target.value}))} />
                    <label htmlFor='edit'> Level *</label>
```

```
                <input type='text' placeholder='Enter Level'
defaultValue={idata.priority_level} onChange={(e) => setIdata((prevValues)=>
({...prevValues,priority_level:e.target.value}))} />
                <label htmlFor='edit'> Description *</label>
                <textarea type='text' placeholder='Enter Description'
defaultValue={idata.description} onChange={(e) => setIdata((prevValues)=>
({...prevValues,description:e.target.value}))} />
                <label htmlFor='edit'> Status *</label>
                <input type='text' placeholder='Status of task' defaultValue={idata.status}
onChange={(e) => setIdata((prevValues)=> ({...prevValues,status:e.target.value}))} />

            </form>
        </Dialog>

        <div className='container'>
            <div className='header'>
                <i className="fa fa-tasks" aria-hidden="true"></i>  
                LINE UP
            </div>
            <div className='Search'>
                <div className="input-group">
                    <div className="form-outline">
```

```
                    <input type="search" id="form1" className="form-control"
placeholder='Enter Task Name / Level' onChange={(e) => setInput(e.target.value)} />
                    <label className="form-label" htmlFor="form1"></label>
                </div>
                <button type="button" className="btn btn-primary" onClick={() =>
searchHandler(input)}>
                    <i className="fas fa-search"></i>
                </button>   
                <select className="fetch-data" onChange={(e) =>
setWeek(e.target.value)} onClick={() => fetchHandler(week, input)}>
                    <option value="all">All</option>
                    <option value="thisweek">This Week</option>
                    <option value="lastweek">Last Week</option>

                </select>
                <button type="button" className="btn btn-primary add-task"
onClick={() => SetOpenDialog(true)}> Add Details </button>
            </div>

        </div>
        <div className='content'>
            {
                data !== [] && data?.map(task => {
```

```
                return (

                    <div key={task.id} className='card' style={{ width: '80%',
marginLeft: '20%', backgroundColor: '#ECF0F1 ' }} >

                        <div className="card-body" onClick={() => taskData(task)}>
                            <h4 style={{ color: '#2C3E50 ', fontSize: '35px'
}}>{task.task_name}</h4>
                            <button type="button" className='btn btn-primary'
style={{color:'white',width:"20%",float:'right'}} onClick={() => taskData(task)}>View
Details</button>

                            {/* <p>{task.description}</p> */}
                        </div>
                        <div className='card-footer'>
                            <span style={{ color:'rgb(57, 57, 201)', fontWeight: 'bolder',
fontSize: '20px' }}>Due Date : {task.task_due_date}</span>   
                            {
                                (task.status)?<span style={{ color:'green', fontWeight: 'bold'
}}>Completed</span>:<span style={{ color:'red', fontWeight: 'bold'
}}>Incomplete</span>
                            }
```

```
                                <i className="fa fa-trash-o" id="pointer1" aria-hidden="true"
style={{ color: 'red' }} onClick={() => deleteHandler(task.id)}></i>
                                <i className="fa fa-pencil-square-o" id="pointer2" aria-
hidden="true" onClick={() => getForm(task.id)} style={{ color: '#CA6F1E' }}></i>


                            </div>


                        </div>
                    );
                })}
            </div>
        </div>
    </>
  )
}
//Backend connection
from django.shortcuts import render
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework import status
from django.db.models import Q
from .models import taskModel
from .serializers import taskModelSerializer
```

```
from datetime import datetime,timedelta
import calendar

class tasks(APIView):

    def get(self,request):
        obj = taskModel.objects.all()
        serializer = taskModelSerializer(obj, many = True)
        return Response(serializer.data,status=status.HTTP_200_OK)

    def post(self,request):
        serializer = taskModelSerializer(data = request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,status=status.HTTP_201_CREATED)
        else:
            return Response(serializer.errors,status=status.HTTP_400_BAD_REQUEST)

class taskDataManipulate(APIView):

    def get(self,request,id):
        try:
            obj = taskModel.objects.get(id = id)
```

```
            serializer = taskModelSerializer(obj)
            return Response(serializer.data,status=status.HTTP_200_OK)
        except taskModel.DoesNotExist:
            msg = {'msg':'not found'}
            return Response(msg,status=status.HTTP_404_NOT_FOUND)


    def put(self,request,id):
        try:
            obj = taskModel.objects.get(id = id)
            print(obj)
            updatedTaskData = {}
            updatedTaskData['task_name'] = request.data['task_name']
            updatedTaskData['task_assigned_date'] = request.data['task_assigned_date']
            updatedTaskData['task_due_date'] = request.data['task_due_date']
            updatedTaskData['priority_level'] = request.data['priority_level']
            updatedTaskData['description'] = request.data['description']
            updatedTaskData['status'] = request.data['status']
            print(updatedTaskData)


            serializer = taskModelSerializer(obj,data = updatedTaskData)
            if serializer.is_valid():
                serializer.save()
```

```
            return
Response(serializer.data,status=status.HTTP_205_RESET_CONTENT)
        else:
            return
Response(serializer.errors,status=status.HTTP_400_BAD_REQUEST)


    except taskModel.DoesNotExist:
        msg = {'msg':'not found'}
        return Response(msg,status=status.HTTP_404_NOT_FOUND)


def delete(self,request,id):
    try:
        obj = taskModel.objects.get(id = id)
        msg = {'msg':'deleted'}
        obj.delete()
        return Response(msg,status=status.HTTP_204_NO_CONTENT)


    except taskModel.DoesNotExist:
        msg = {'msg':'Not Found'}
        return Response(msg,status=status.HTTP_404_NOT_FOUND)



class taskDataFilter(APIView):
```

```
    def get(self,request,text):
        try:
            obj = taskModel.objects.filter(Q(task_name__icontains = text) |
Q(priority_level__icontains = text))
            serializer = taskModelSerializer(obj,many=True)
            return Response(serializer.data,status=status.HTTP_200_OK)
        except taskModel.DoesNotExist:
            msg = {'msg':'Not found'}
            return Response(msg,status=status.HTTP_404_NOT_FOUND)


class FetchTaskData(APIView):
    def get(self,request,text):
        try:
            startdate = datetime.today()
            if(text == 'thisweek'):
                # startdates = startdate + timedelta()
                startdate = startdate + timedelta(days = -(startdate.weekday() ))
                enddate = startdate + timedelta(days = 6)
                print(f"start date : {startdate} - enddate : {enddate}")
                obj = taskModel.objects.filter(Q(task_assigned_date__range =
(startdate,enddate)) | Q(task_due_date__range = (startdate,enddate)) |
Q(task_assigned_date__lte = enddate,task_due_date__gte = enddate))
```

```
            elif(text == 'lastweek'):
                enddate = startdate + timedelta(days = -(startdate.weekday()+1))
                startdate = enddate+ timedelta(days = -(enddate.weekday() ))
                print(f"start date : {startdate} - enddate : {enddate}")
                obj = taskModel.objects.filter(Q(task_assigned_date__range =
(startdate,enddate)) | Q(task_due_date__range = (startdate,enddate)) |
Q(task_assigned_date__lte = enddate,task_due_date__gte = enddate))

            elif(text == 'all'):
                obj=taskModel.objects.all()

            else:
                raise NameError(text)

            serializer = taskModelSerializer(obj,many=True)
            return Response(serializer.data,status=status.HTTP_200_OK)

        except taskModel.DoesNotExist:
            msg = {'msg':'Not Found'}
            return Response(msg,status=status.HTTP_404_NOT_FOUND)

class FullSearch(APIView):
```

```
    def get(self,request,filter,fetch):


        # fetch = request.data['week']
        if fetch == "" and filter == "":
            obj = taskModel.objects.all()
            serializer = taskModelSerializer(obj,many=True)
            return Response(serializer.data,status=status.HTTP_200_OK)


        elif filter == "" :
            try:
                obj = taskModel.objects.filter(Q(task_name__icontains = filter) |
Q(priority_level__icontains = filter))
                serializer = taskModelSerializer(obj,many=True)
                return Response(serializer.data,status=status.HTTP_200_OK)
            except taskModel.DoesNotExist:
                msg ={'msg':'not found'}
                return Response(msg,status=status.HTTP_404_NOT_FOUND)


        else:
            startdate = datetime.today()
            if fetch == "thisweek":
                startdate = startdate + timedelta(days = -(startdate.weekday() ))
                enddate = startdate + timedelta(days = 6)
```

```
            print(f"start date : {startdate} - enddate : {enddate}")
        elif fetch == "lastweek":
            enddate = startdate + timedelta(days = -(startdate.weekday()+1))
            startdate = enddate+ timedelta(days = -(enddate.weekday() ))
            print(f"start date : {startdate} - enddate : {enddate}")

        else:
            raise NameError(fetch)

        fetched_records = taskModel.objects.filter(Q(task_assigned_date__range =
(startdate,enddate)) | Q(task_due_date__range = (startdate,enddate)) |
Q(task_assigned_date__lte = enddate,task_due_date__gte = enddate))

    try:
        obj = fetched_records.filter(Q(task_name__icontains = filter) |
Q(priority_level__icontains = filter))
        serializer = taskModelSerializer(obj,many=True)
        return Response(serializer.data,status=status.HTTP_200_OK)
    except fetched_records.DoesNotExist:
        msg ={'msg':'not found'}
        return Response(msg,status=status.HTTP_404_NOT_FOUND)
```

Final

ORIGINALITY REPORT

32%	31%	5%	24%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.researchgate.net Internet Source	5%
2	stackoverflow.com Internet Source	4%
3	repositorio.ug.edu.ec Internet Source	3%
4	forum.djangoproject.com Internet Source	3%
5	Submitted to University of Greenwich Student Paper	2%
6	Submitted to Birla Institute of Technology and Science Pilani Student Paper	2%
7	Submitted to University of Hertfordshire Student Paper	2%
8	codeclimate.com Internet Source	2%
9	gitlab.sliit.lk Internet Source	2%

10	blog.openreplay.com Internet Source	2%
11	git.trustie.net Internet Source	1%
12	blog.enriqueoriol.com Internet Source	1%
13	gitlab.stud.iie.ntnu.no Internet Source	1%
14	blog.learncodeonline.in Internet Source	1%
15	Submitted to McNeese State University Student Paper	1%
16	www.maibornwolff.de Internet Source	1%
17	Submitted to Florida State University Student Paper	1%

Exclude quotes	On	Exclude matches	< 3 words
Exclude bibliography	Off		

Final