DRAG N DROP APP

**CLIENT**

package.json

{

"name": "client",

"version": "0.1.0",

"private": true,

"dependencies": {

"@testing-library/jest-dom": "^5.17.0",

"@testing-library/react": "^13.4.0",

"@testing-library/user-event": "^13.5.0",

"axios": "^1.7.7",

"dotenv": "^16.4.5",

"react": "^18.3.1",

"react-dom": "^18.3.1",

"react-draggable": "^4.4.6",

"react-redux": "^9.1.2",

"react-saga": "^0.3.1",

"react-scripts": "5.0.1",

"react-toastify": "^10.0.5",

"web-vitals": "^2.1.4",

"zustand": "^4.5.5"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import { Provider } from 'react-redux';

import reportWebVitals from './reportWebVitals';

import { ToastContainer, toast } from 'react-toastify';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

<ToastContainer />

</React.StrictMode>

);

reportWebVitals();

App.js

import { useEffect, useState, useRef } from 'react';

import axios from 'axios';

import User from './components/user.component';

import 'react-toastify/dist/ReactToastify.css';

import { toast } from 'react-toastify';

function App() {

const [data, setData] = useState([]);

const [loading, setLoading] = useState(true);

const errorShown = useRef(false);

const fetchUsersTasks = async () => {

try {

setLoading(true);

const response = await axios.get(`${process.env.REACT\_APP\_API\_URL}/all\_task\_usernames`);

setData(response.data.usersWithTaskTexts);

setLoading(false);

} catch (error) {

if (!errorShown.current) {

console.error('Error fetching tasks:', error);

toast.error('Error fetching tasks');

errorShown.current = true;

setLoading(false);

}

}

};

useEffect(() => {

fetchUsersTasks();

}, []);

const moveTask = async (sourceUserId, targetUserId, taskText) => {

try {

// Ensure the request URL matches your API endpoint

const response = await axios.post(`${process.env.REACT\_APP\_API\_URL}/move-task`, {

sourceUserId,

targetUserId,

taskText,

});

setData((prevUsers) => {

// If the source and target users are the same, return previous state without any changes

if (parseInt(sourceUserId, 10) === parseInt(targetUserId, 10)) {

return prevUsers;

}

const updatedUsers = prevUsers.map((user) => {

if (user.user\_id === parseInt(sourceUserId, 10)) {

// Remove task from source user

return {

...user,

tasks: user.tasks.filter((task) => task.text !== taskText),

};

}

if (user.user\_id === parseInt(targetUserId, 10)) {

// Add task to target user

return {

...user,

tasks: [...user.tasks, { text: taskText }],

};

}

return user;

});

return updatedUsers;

});

} catch (error) {

// Enhanced error logging

console.error('Error moving task:', error.response ? error.response.data : error.message);

toast.error('Failed to move task');

}

};

return (

<div className="App bg-zinc-900 h-auto">

<center>

<h1 className="text-[145px] font-semibold text-white">Users Tasks</h1>

</center>

<div className="flex flex-row flex-wrap justify-center items-center gap-4 p-4">

{loading ? (

<p className="text-white">Loading...</p>

) : (

data.map((element) => (

<User

key={element.user\_id}

data={element}

moveTask={moveTask}

/>

))

)}

</div>

</div>

);

}

export default App;

user

import React from 'react';

const User = ({ data, moveTask }) => {

//ondrop

const handleOnDrop = (e) => {

e.preventDefault();

const droppedTaskText = e.dataTransfer.getData('text');

const sourceUserId = parseInt(e.dataTransfer.getData('user\_id'));

const targetUserId = data.user\_id;

// Call the moveTask function from the parent component

moveTask(sourceUserId, targetUserId, droppedTaskText);

};

//ondrag

const handleOnDragOver = (e) => {

e.preventDefault(); // Allow drop

};

//dragStart

const handleOnDragStart = (e, text, id) => {

e.dataTransfer.setData('text', text);

e.dataTransfer.setData('user\_id', id);

};

return (

<div

className="text-white text-[24px] bg-zinc-700 rounded-lg p-2 m-2 h-[20rem] w-[20rem]"

onDragOver={handleOnDragOver}

onDrop={handleOnDrop}

>

<p className="text-white font-semibold">{data.name}</p>

{data.tasks && data.tasks.length > 0 ? (

data.tasks.map((task, index) => (

<p

key={task.text} // Ensure unique keys

className="cursor-move bg-zinc-600 rounded-lg p-2 mb-2"

draggable

onDragStart={(e) => handleOnDragStart(e, task.text, data.user\_id)}

>

{task.text}

</p>

))

) : (

<p>No tasks available</p>

)}

</div>

);

};

export default User;

**SERVER**

const express = require('express');

const app = require('./app');

const userRoutes = require('./router/userRoutes'); // Adjust the path as necessary

// Middleware to parse JSON

app.use(express.json());

// Use the user routes

app.use('/api', userRoutes); // Prefix all routes with /api

// Error handling middleware

app.use((err, req, res, next) => {

res.status(500).json({

message: 'Internal Server Error',

error: err.message

});

});

const PORT = process.env.PORT || 8000;

app.listen(PORT, () => {

console.log(`Server is listening on port ${PORT}`);

});

app.js

const express = require('express');

const cors = require('cors'); // Import CORS middleware

const app = express();

// Import routes

const userRoutes = require('./router/userRoutes');

// Middleware setup

app.use(cors());

app.use(express.json());

app.get('/', (req, res)=>{

res.end("Server is running");

})

// Route setup

app.use('/api', userRoutes);

app.use((err, req, res, next) => {

console.error('Error:', err.message);

res.status(500).json({

message: 'Internal Server Error',

error: err.message

});

});

module.exports = app;

userRoutes.js

const express = require('express');

const router = express.Router();

const userController = require('../controller/userController')

router.get('/all\_users', userController.findAllUsers);

router.get('/all\_tasks', userController.findAllTasks);

router.get('/all\_usernames', userController.findAllUsersName);

router.get('/all\_task\_usernames', userController.fetchTasksWithUserNames);

router.post('/move-task', userController.updateData);

module.exports = router;

userController.js

const { PrismaClient } = require('@prisma/client');

const Prisma = new PrismaClient();

exports.findAllUsers = async (req, res) => {

try {

// Fetch all users

const users = await Prisma.user.findMany(); // No need for include here

// Send response

res.status(200).json({ // Use 200 for success

message: 'success',

users

});

} catch (error) {

// Handle error properly

console.error('Error fetching users:', error); // Log the error for debugging

res.status(500).json({ // Use 500 for server errors

message: 'fail',

error: error.message

});

}

};

exports.findAllUsersName = async (req, res) => {

try {

// Fetch all users

const users = await Prisma.user.findMany({

select: {

name: true

}

}); // No need for include here

// Send response

res.status(200).json({ // Use 200 for success

message: 'success',

users

});

} catch (error) {

// Handle error properly

console.error('Error fetching users:', error); // Log the error for debugging

res.status(500).json({ // Use 500 for server errors

message: 'fail',

error: error.message

});

}

};

exports.findAllTasks = async (req, res) => {

try {

// Fetch all tasks

const tasks = await Prisma.task.findMany(); // Omit include if not using

// Send response

res.status(200).json({

message: 'success',

tasks

});

} catch (error) {

// Handle error properly

console.error('Error fetching tasks:', error); // Log the error for debugging

res.status(500).json({

message: 'fail',

error: error.message

});

}

};

exports.fetchTasksWithUserNames = async (req, res) => {

try {

const usersWithTaskTexts = await Prisma.user.findMany({

include: {

tasks: {

select: {

text: true, // Only include the 'text' field for each task

},

},

},

orderBy: {

user\_id: 'asc', // Order users by their 'id' in ascending order

},

});

res.status(200).json({

message: 'success',

usersWithTaskTexts

})

} catch (error) {

console.log("users with task", error.message);

res.status(500).json({

message: "fail",

error: error.message

})

}

}

exports.updateData = async (req, res) => {

const { taskText, sourceUserId, targetUserId } = req.body;

console.log(req.body);

try {

// Validate input

if (!taskText || !sourceUserId || !targetUserId) {

return res.status(400).json({ message: 'Missing required fields' });

}

// Find the task for the specific user

const task = await Prisma.task.findFirst({

where: {

user\_id: parseInt(sourceUserId, 10),

text: taskText,

},

});

// Check if the task was found

if (!task) {

console.log('Task not found');

return res.status(404).json({ message: 'Task not found' });

}

// Check if source and target users are different

if (parseInt(sourceUserId, 10) !== parseInt(targetUserId, 10)) {

// Create the new task for the target user

const newTask = await Prisma.task.create({

data: {

user\_id: parseInt(targetUserId, 10),

text: taskText,

},

});

// Delete the original task

await Prisma.task.delete({

where: {

task\_id: task.task\_id,

},

});

console.log('New task created:', newTask);

res.status(200).json({

message: 'Task moved successfully',

newTask,

});

} else {

// If the task is not moved, just respond with success

res.status(200).json({

message: 'Task remained with the same user',

});

}

} catch (error) {

console.error('Error moving task:', error.message);

res.status(500).json({

message: 'Failed to move task',

error: error.message,

});

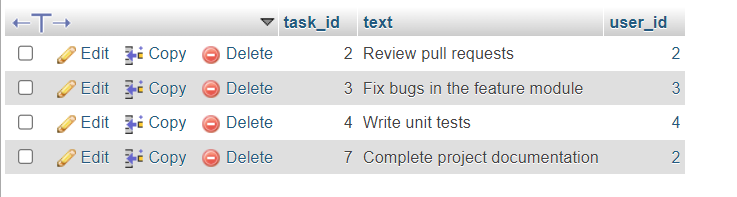
} finally {

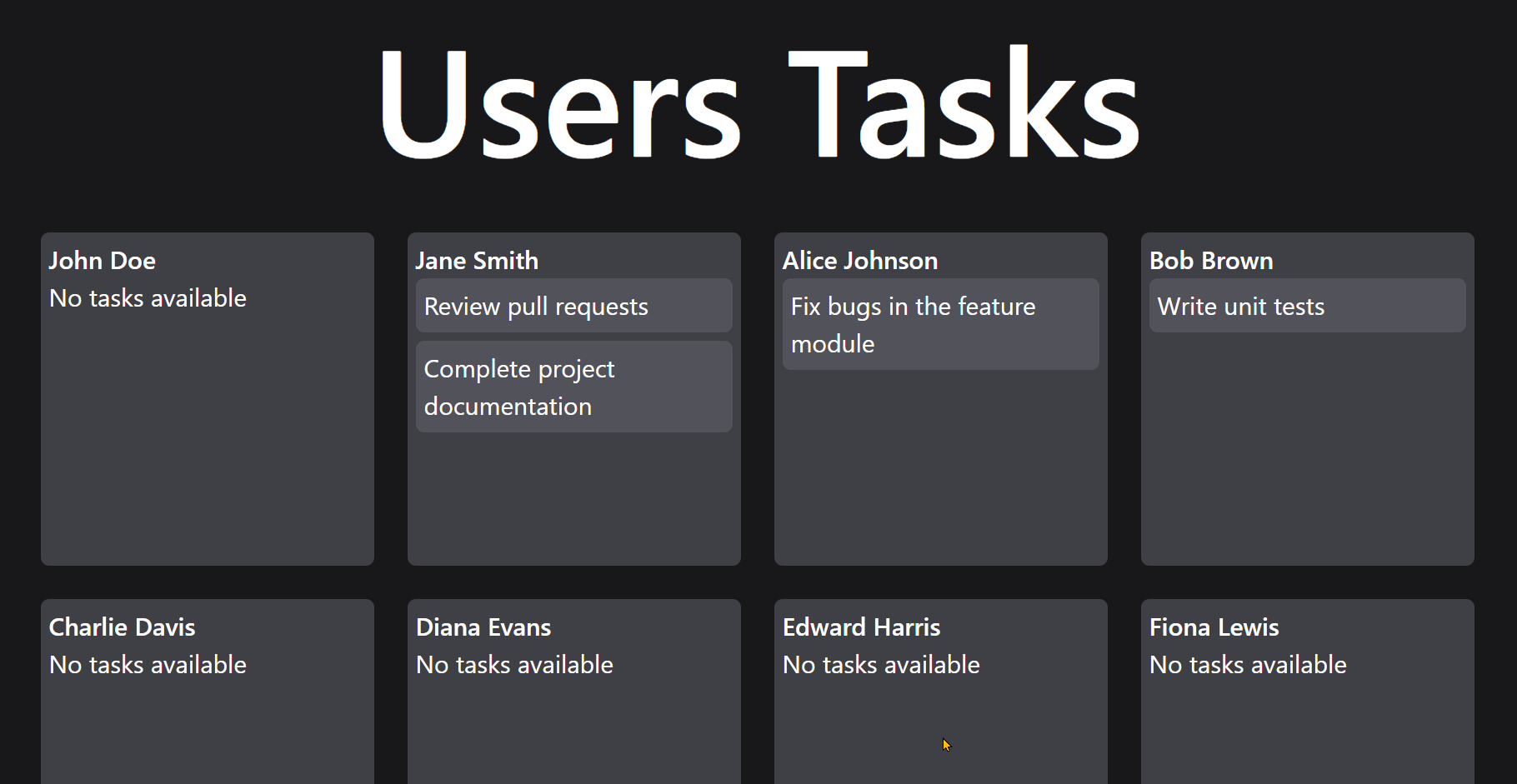
await Prisma.$disconnect();

}

};

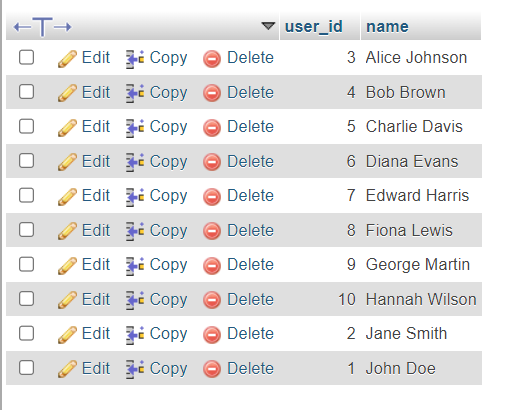
OUTPUT-

BEFORE



AFTER-

**Moving task from user 2 to user 1**

****