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

import axios from 'axios';

const ShiftSwapRequests = () => {

const [requests, setRequests] = useState([]);

const [requesterName, setRequesterName] = useState("");

const [requesterEmployeeId, setRequesterEmployeeId] = useState("");

const [colleagueName, setColleagueName] = useState("");

const [currentShiftName, setCurrentShiftName] = useState("");

const [desiredShiftName, setDesiredShiftName] = useState("");

const [reason, setReason] = useState("");

const [startDate, setStartDate] = useState("");

const [endDate, setEndDate] = useState("");

const [showRequests, setShowRequests] = useState(false);

const [editId, setEditId] = useState(null);

const [successMessage, setSuccessMessage] = useState("");

// Fetch pending requests from the backend

const fetchPendingRequests = async () => {

try {

const response = await axios.get('http://localhost:5000/api/shift-swap/pending');

setRequests(Array.isArray(response.data) ? response.data : []);

} catch (error) {

console.error('Error fetching pending requests:', error);

}

};

useEffect(() => {

if (showRequests) {

fetchPendingRequests();

}

}, [showRequests]);

const handleSubmit = async (e) => {

e.preventDefault();

if (!requesterName || !requesterEmployeeId || !colleagueName || !currentShiftName || !desiredShiftName || !reason.trim() || !startDate || !endDate) {

alert("All fields are required!");

return;

}

const newRequest = {

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart: startDate,

shiftEnd: endDate,

};

try {

let response;

if (editId) {

response = await axios.put(`http://localhost:5000/api/shift-swap/${editId}`, newRequest);

setEditId(null);

} else {

response = await axios.post('http://localhost:5000/api/shift-swap', newRequest);

}

console.log('Response:', response.data);

setSuccessMessage("Your request has been successfully submitted. You will be responded to shortly.");

fetchPendingRequests();

resetForm();

setTimeout(() => setSuccessMessage(""), 5000); // Clear success message after 5 seconds

} catch (error) {

console.error('Error submitting shift swap request:', error);

}

};

const resetForm = () => {

setRequesterName("");

setRequesterEmployeeId("");

setColleagueName("");

setCurrentShiftName("");

setDesiredShiftName("");

setReason("");

setStartDate("");

setEndDate("");

};

const handleEdit = (id) => {

const requestToEdit = requests.find((request) => request.id === id);

if (requestToEdit) {

setRequesterName(requestToEdit.requester\_name);

setRequesterEmployeeId(requestToEdit.requester\_employee\_id);

setColleagueName(requestToEdit.colleague\_name);

setCurrentShiftName(requestToEdit.current\_shift\_name);

setDesiredShiftName(requestToEdit.desired\_shift\_name);

setReason(requestToEdit.reason);

setStartDate(requestToEdit.shift\_start);

setEndDate(requestToEdit.shift\_end);

setEditId(id);

}

};

// Internal styling

const styles = {

container: {

maxWidth: "700px",

margin: "auto",

padding: "20px",

backgroundColor: "#f8f9fa",

borderRadius: "8px",

boxShadow: "0px 4px 8px rgba(0, 0, 0, 0.1)",

textAlign: "center",

},

heading: { color: "#2c3e50" },

form: { display: "flex", flexDirection: "column", gap: "10px" },

input: {

padding: "10px",

border: "1px solid #ccc",

borderRadius: "5px",

},

button: {

backgroundColor: "#2c3e50",

color: "white",

padding: "10px",

border: "none",

borderRadius: "5px",

cursor: "pointer",

},

pendingButton: {

backgroundColor: "blue",

color: "white",

padding: "5px 10px",

border: "none",

borderRadius: "5px",

cursor: "default", // Disable click

},

successMessage: {

backgroundColor: "#d4edda",

color: "#155724",

padding: "10px",

borderRadius: "5px",

marginBottom: "20px",

},

tableContainer: { marginTop: "20px", textAlign: "left" },

table: { width: "100%", borderCollapse: "collapse" },

thTd: { border: "1px solid #ddd", padding: "8px", textAlign: "left" },

tableHeader: { backgroundColor: "#2c3e50", color: "white" },

};

return (

<div style={styles.container}>

<h2 style={styles.heading}>Shift Swap Requests</h2>

{/\* Success Message \*/}

{successMessage && (

<div style={styles.successMessage}>

{successMessage}

</div>

)}

<form onSubmit={handleSubmit} style={styles.form}>

<label> Your Name: </label>

<input

type="text"

placeholder="Enter your name"

value={requesterName}

onChange={(e) => setRequesterName(e.target.value)}

style={styles.input}

/>

<label> Your Employee ID: </label>

<input

type="text"

placeholder="Enter your employee ID"

value={requesterEmployeeId}

onChange={(e) => setRequesterEmployeeId(e.target.value)}

style={styles.input}

/>

<label> Colleague's Name: </label>

<input

type="text"

placeholder="Enter colleague's name"

value={colleagueName}

onChange={(e) => setColleagueName(e.target.value)}

style={styles.input}

/>

<label> Current Shift Name: </label>

<input

type="text"

placeholder="Enter current shift name"

value={currentShiftName}

onChange={(e) => setCurrentShiftName(e.target.value)}

style={styles.input}

/>

<label> Desired Shift Name: </label>

<input

type="text"

placeholder="Enter desired shift name"

value={desiredShiftName}

onChange={(e) => setDesiredShiftName(e.target.value)}

style={styles.input}

/>

<label> Reason for Swap: </label>

<textarea

placeholder="Provide a reason..."

value={reason}

onChange={(e) => setReason(e.target.value)}

style={styles.input}

></textarea>

<label> Start Date: </label>

<input

type="date"

value={startDate}

onChange={(e) => setStartDate(e.target.value)}

style={styles.input}

/>

<label> End Date: </label>

<input

type="date"

value={endDate}

onChange={(e) => setEndDate(e.target.value)}

style={styles.input}

/>

<button type="submit" style={styles.button}>

{editId ? "Update Swap Request" : "Submit Swap Request"}

</button>

</form>

<button

onClick={() => setShowRequests(!showRequests)}

style={{ ...styles.button, marginTop: "15px" }}

>

{showRequests ? "Hide Pending Requests" : "View Pending Requests"}

</button>

{showRequests && (

<div style={styles.tableContainer}>

<h3>View Shift Swap Requests</h3>

<table style={styles.table}>

<thead>

<tr style={styles.tableHeader}>

<th style={styles.thTd}>Requester Name</th>

<th style={styles.thTd}>Employee ID</th>

<th style={styles.thTd}>Colleague Name</th>

<th style={styles.thTd}>Current Shift</th>

<th style={styles.thTd}>Desired Shift</th>

<th style={styles.thTd}>Reason</th>

<th style={styles.thTd}>Start Date</th>

<th style={styles.thTd}>End Date</th>

<th style={styles.thTd}>Status</th>

</tr>

</thead>

<tbody>

{Array.isArray(requests) && requests.length > 0 ? (

requests.map((request) => (

<tr key={request.id}>

<td style={styles.thTd}>{request.requester\_name}</td>

<td style={styles.thTd}>{request.requester\_employee\_id}</td>

<td style={styles.thTd}>{request.colleague\_name}</td>

<td style={styles.thTd}>{request.current\_shift\_name}</td>

<td style={styles.thTd}>{request.desired\_shift\_name}</td>

<td style={styles.thTd}>{request.reason}</td>

<td style={styles.thTd}>{new Date(request.shift\_start).toLocaleString()}</td>

<td style={styles.thTd}>{new Date(request.shift\_end).toLocaleString()}</td>

<td style={styles.thTd}>

<button style={styles.pendingButton}>

{request.status}

</button>

</td>

</tr>

))

) : (

<tr>

<td colSpan="9" style={{ textAlign: "center" }}>No pending requests found.</td>

</tr>

)}

</tbody>

</table>

</div>

)}

</div>

);

};

export default ShiftSwapRequests; then the controller:const shiftSwapRequestModel = require('../models/shiftSwapRequest');

const createShiftSwapRequest = async (req, res) => {

const {

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd,

} = req.body;

try {

const request = await shiftSwapRequestModel.createShiftSwapRequest(

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd

);

res.status(201).json(request);

} catch (error) {

res.status(500).json({ message: 'Error creating shift swap request', error });

}

};

const getPendingRequests = async (req, res) => {

try {

const requests = await shiftSwapRequestModel.getPendingRequests();

res.status(200).json(requests);

} catch (error) {

res.status(500).json({ message: 'Error fetching pending requests', error });

}

};

const updateShiftSwapRequest = async (req, res) => {

const { id } = req.params;

const {

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd,

} = req.body;

try {

const updatedRequest = await shiftSwapRequestModel.updateShiftSwapRequest(

id,

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd

);

res.status(200).json(updatedRequest);

} catch (error) {

res.status(500).json({ message: 'Error updating shift swap request', error });

}

};

const deleteShiftSwapRequest = async (req, res) => {

const { id } = req.params;

try {

await shiftSwapRequestModel.deleteShiftSwapRequest(id);

res.status(200).json({ message: 'Shift swap request deleted successfully' });

} catch (error) {

res.status(500).json({ message: 'Error deleting shift swap request', error });

}

};

module.exports = {

createShiftSwapRequest,

getPendingRequests,

updateShiftSwapRequest,

deleteShiftSwapRequest,

};model:const pool = require('../config/db');

const createShiftSwapRequest = async (

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd

) => {

const query = `

INSERT INTO shift\_swap\_requests (

requester\_name, requester\_employee\_id, colleague\_name,

current\_shift\_name, desired\_shift\_name, reason, shift\_start, shift\_end

)

VALUES ($1, $2, $3, $4, $5, $6, $7, $8)

RETURNING \*;

`;

const values = [

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd,

];

const result = await pool.query(query, values);

return result.rows[0];

};

const getPendingRequests = async () => {

const query = `

SELECT \* FROM shift\_swap\_requests

WHERE status = 'pending';

`;

const result = await pool.query(query);

return result.rows;

};

const updateShiftSwapRequest = async (

id,

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd

) => {

const query = `

UPDATE shift\_swap\_requests

SET requester\_name = $1,

requester\_employee\_id = $2,

colleague\_name = $3,

current\_shift\_name = $4,

desired\_shift\_name = $5,

reason = $6,

shift\_start = $7,

shift\_end = $8

WHERE id = $9

RETURNING \*;

`;

const values = [

requesterName,

requesterEmployeeId,

colleagueName,

currentShiftName,

desiredShiftName,

reason,

shiftStart,

shiftEnd,

id,

];

const result = await pool.query(query, values);

return result.rows[0];

};

const deleteShiftSwapRequest = async (id) => {

const query = `

DELETE FROM shift\_swap\_requests

WHERE id = $1;

`;

await pool.query(query, [id]);

};

module.exports = {

createShiftSwapRequest,

getPendingRequests,

updateShiftSwapRequest,

deleteShiftSwapRequest,

};route:const express = require('express');

const router = express.Router();

const shiftSwapRequestController = require('../controllers/shiftSwapRequestController');

router.get('/shift-swap/pending', shiftSwapRequestController.getPendingRequests);

router.put('/shift-swap/:id', shiftSwapRequestController.updateShiftSwapRequest);

module.exports = router; and server.js :const express = require('express');

const cors = require('cors');

const dotenv = require('dotenv');

const bodyParser = require('body-parser');

const authRoutes = require('./routes/authRoutes');

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

const overtimeRoutes = require('./routes/overtimeRoutes');

const shiftSwapRequestRoutes = require('./routes/shiftSwapRequestRoutes'); // Add this line

const pool = require('./config/db');

dotenv.config();

const app = express();

app.use(cors());

app.use(bodyParser.json());

// Authentication routes

app.use('/api/auth', authRoutes);

// User management routes

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

// Overtime routes

app.use("/api", overtimeRoutes);

// Shift swap request routes

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

// Test database connection

pool.query('SELECT NOW()', (err, res) => {

if (err) {

console.error('Error connecting to the database:', err);

} else {

console.log('Database connection successful:', res.rows[0].now);

}

});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

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

});