Sure thing, Raghavendra! Let's integrate the Express backend and React frontend into a single project. Here's a detailed step-by-step guide:

1. \*\*Directory Structure:\*\*

We'll structure the project with the frontend and backend in their respective directories.

```

snowflake-app/

├── backend/

│ ├── config/

│ │ └── snowflakeConfig.js

│ ├── controllers/

│ │ └── roleController.js

│ ├── routes/

│ │ └── roleRoutes.js

│ ├── app.js

│ ├── package.json

│ └── .env

└── frontend/

├── public/

├── src/

│ ├── components/

│ │ └── RoleHierarchy.js

│ ├── App.js

│ ├── index.js

├── package.json

├── .env

├── .gitignore

└── README.md

```

2. \*\*Setting Up the Backend:\*\*

Follow these steps to set up the backend:

\*\*a. Create `config/snowflakeConfig.js`:\*\*

```javascript

// backend/config/snowflakeConfig.js

module.exports = {

account: process.env.SNOWFLAKE\_ACCOUNT,

username: process.env.SNOWFLAKE\_USERNAME,

password: process.env.SNOWFLAKE\_PASSWORD,

role: process.env.SNOWFLAKE\_ROLE,

warehouse: process.env.SNOWFLAKE\_WAREHOUSE,

database: process.env.SNOWFLAKE\_DATABASE,

schema: process.env.SNOWFLAKE\_SCHEMA,

};

```

\*\*b. Create `.env`:\*\*

```env

SNOWFLAKE\_ACCOUNT=<your\_account>

SNOWFLAKE\_USERNAME=<your\_username>

SNOWFLAKE\_PASSWORD=<your\_password>

SNOWFLAKE\_ROLE=<your\_role>

SNOWFLAKE\_WAREHOUSE=<your\_warehouse>

SNOWFLAKE\_DATABASE=<your\_database>

SNOWFLAKE\_SCHEMA=<your\_schema>

PORT=5000

```

\*\*c. Create `controllers/roleController.js`:\*\*

```javascript

// backend/controllers/roleController.js

const snowflake = require('snowflake-sdk');

const config = require('../config/snowflakeConfig');

const connection = snowflake.createConnection(config);

connection.connect((err, conn) => {

if (err) {

console.error('Unable to connect: ' + err.message);

} else {

console.log('Successfully connected to Snowflake.');

}

});

const getRoleHierarchy = (req, res) => {

connection.execute({

sqlText: 'SELECT \* FROM role\_hierarchy', // Adjust this query to your needs

complete: (err, stmt, rows) => {

if (err) {

console.error('Error executing query: ' + err.message);

res.status(500).send('Error');

} else {

res.json(rows);

}

}

});

};

module.exports = { getRoleHierarchy };

```

\*\*d. Create `routes/roleRoutes.js`:\*\*

```javascript

// backend/routes/roleRoutes.js

const express = require('express');

const router = express.Router();

const { getRoleHierarchy } = require('../controllers/roleController');

router.get('/role-hierarchy', getRoleHierarchy);

module.exports = router;

```

\*\*e. Set up `app.js`:\*\*

```javascript

// backend/app.js

const express = require('express');

const app = express();

const dotenv = require('dotenv');

dotenv.config();

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

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

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

app.listen(port, () => {

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

});

```

\*\*f. Create `package.json`:\*\*

```json

// backend/package.json

{

"name": "snowflake-backend",

"version": "1.0.0",

"main": "app.js",

"scripts": {

"start": "node app.js",

"dev": "nodemon app.js"

},

"dependencies": {

"dotenv": "^10.0.0",

"express": "^4.17.1",

"snowflake-sdk": "^1.6.1"

},

"devDependencies": {

"nodemon": "^2.0.15"

}

}

```

3. \*\*Setting Up the Frontend:\*\*

Follow these steps to set up the frontend:

\*\*a. Set up React app:\*\*

Navigate to the `frontend` directory and create a React app:

```bash

cd frontend

npx create-react-app .

```

\*\*b. Install dependencies:\*\*

```bash

npm install axios react-treeview

```

\*\*c. Create `components/RoleHierarchy.js`:\*\*

```javascript

// frontend/src/components/RoleHierarchy.js

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

import axios from 'axios';

import TreeView from 'react-treeview';

import 'react-treeview/react-treeview.css';

const RoleHierarchy = () => {

const [roles, setRoles] = useState([]);

const [searchTerm, setSearchTerm] = useState('');

useEffect(() => {

axios.get('/api/role-hierarchy')

.then(response => {

setRoles(response.data);

})

.catch(error => {

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

});

}, []);

const handleSearchChange = (event) => {

setSearchTerm(event.target.value);

};

const filteredRoles = roles.filter(role =>

role.role\_name.toLowerCase().includes(searchTerm.toLowerCase())

);

const renderTree = (roles) => {

return roles.map(role => (

<TreeView

key={role.id}

nodeLabel={role.role\_name}

defaultCollapsed={true}

>

{role.sub\_roles && renderTree(role.sub\_roles)}

</TreeView>

));

};

return (

<div>

<input

type="text"

placeholder="Search roles"

value={searchTerm}

onChange={handleSearchChange}

/>

<div>

{renderTree(filteredRoles)}

</div>

</div>

);

};

export default RoleHierarchy;

```

\*\*d. Update `App.js`:\*\*

```javascript

// frontend/src/App.js

import React from 'react';

import RoleHierarchy from './components/RoleHierarchy';

const App = () => {

return (

<div className="App">

<h1>Snowflake Role Hierarchy</h1>

<RoleHierarchy />

</div>

);

};

export default App;

```

\*\*e. Set up proxy in `package.json`:\*\*

```json

// frontend/package.json

{

"name": "frontend",

"version": "0.1.0",

"private": true,

"dependencies": {

// ... your dependencies

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"proxy": "http://localhost:5000"

}

```

4. \*\*Running the Project:\*\*

Now, follow these steps to run the project:

\*\*a. Start the backend:\*\*

Navigate to the `backend` directory and start the backend server:

```bash

cd backend

npm install

npm run dev

```

\*\*b. Start the frontend:\*\*

Open another terminal, navigate to the `frontend` directory, and start the React app:

```bash

cd frontend

npm install

npm start

```

You should now have a full-stack application with an Express backend and a React frontend, displaying Snowflake role hierarchy with search functionality. Let me know how it goes!

Got it. We can modify the backend so it doesn't connect to Snowflake until an endpoint is hit. Let’s make sure the connection only initializes when it’s needed. Here’s how to do it:

1. \*\*Modify the `roleController.js`:\*\*

Update the controller to establish the connection only when the request is made.

```javascript

// backend/controllers/roleController.js

const snowflake = require('snowflake-sdk');

const config = require('../config/snowflakeConfig');

const connectToSnowflake = () => {

return new Promise((resolve, reject) => {

const connection = snowflake.createConnection(config);

connection.connect((err, conn) => {

if (err) {

reject('Unable to connect: ' + err.message);

} else {

resolve(conn);

}

});

});

};

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

try {

const connection = await connectToSnowflake();

connection.execute({

sqlText: 'SELECT \* FROM role\_hierarchy', // Adjust this query to your needs

complete: (err, stmt, rows) => {

if (err) {

console.error('Error executing query: ' + err.message);

res.status(500).send('Error');

} else {

res.json(rows);

}

}

});

} catch (error) {

console.error(error);

res.status(500).send('Error connecting to Snowflake');

}

};

module.exports = { getRoleHierarchy };

```

2. \*\*Create a `.env` file:\*\*

Ensure you have a `.env` file in your `backend` directory with your Snowflake configuration and port.

```env

SNOWFLAKE\_ACCOUNT=<your\_account>

SNOWFLAKE\_USERNAME=<your\_username>

SNOWFLAKE\_PASSWORD=<your\_password>

SNOWFLAKE\_ROLE=<your\_role>

SNOWFLAKE\_WAREHOUSE=<your\_warehouse>

SNOWFLAKE\_DATABASE=<your\_database>

SNOWFLAKE\_SCHEMA=<your\_schema>

PORT=5000

```

3. \*\*Update the `app.js`:\*\*

No changes needed here, as it should already be set up correctly to use the routes.

```javascript

// backend/app.js

const express = require('express');

const app = express();

const dotenv = require('dotenv');

dotenv.config();

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

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

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

app.listen(port, () => {

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

});

```

With these changes, the Snowflake connection will only be initiated when the `/role-hierarchy` endpoint is hit, thus avoiding the eager connection at server start-up. This should resolve the issue. Give it a go!

// frontend/src/components/RoleHierarchy.js

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

import axios from 'axios';

import { ReactFlow, MiniMap, Controls, Background } from '@xyflow/react';

const RoleHierarchy = () => {

const [roles, setRoles] = useState([]);

const [selectedRole, setSelectedRole] = useState(null);

const [newRoleName, setNewRoleName] = useState('');

const [nodes, setNodes] = useState([]);

const [edges, setEdges] = useState([]);

const transformDataToElements = useCallback((data, parentId = null, level = 0, posX = 0) => {

const nodes = [];

const edges = [];

let offsetX = posX;

data.forEach((role) => {

const nodeId = role.id.toString();

nodes.push({

id: nodeId,

type: 'default',

data: { label: role.role\_name },

position: { x: offsetX, y: level \* 100 }

});

if (parentId) {

edges.push({

id: `e${parentId}-${nodeId}`,

source: parentId,

target: nodeId,

type: 'smoothstep'

});

}

offsetX += 200; // Adjust the X position for spacing

const { nodes: childNodes, edges: childEdges } = transformDataToElements(role.sub\_roles || [], nodeId, level + 1, offsetX);

nodes.push(...childNodes);

edges.push(...childEdges);

});

return { nodes, edges };

}, []); // Add any dependencies here if needed

useEffect(() => {

axios.get('/api/role-hierarchy-sample')

.then(response => {

console.log('Fetched data:', response.data); // Log fetched data

setRoles(response.data); // Set roles from fetched data

const { nodes, edges } = transformDataToElements(response.data);

console.log('Transformed nodes:', nodes); // Log transformed nodes

console.log('Transformed edges:', edges); // Log transformed edges

setNodes(nodes);

setEdges(edges);

})

.catch(error => {

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

});

}, [transformDataToElements]);

const onNodeClick = (event, node) => {

const role = findRoleById(roles, parseInt(node.id));

console.log('Clicked role:', role); // Log clicked role

setSelectedRole(role);

};

const findRoleById = (rolesList, id) => {

for (const role of rolesList) {

if (role.id === id) return role;

const subRole = findRoleById(role.sub\_roles || [], id);

if (subRole) return subRole;

}

return null;

};

const handleAddRole = () => {

if (selectedRole && newRoleName) {

const newSubRole = {

id: Date.now(), // Just a placeholder ID for example

role\_name: newRoleName,

sub\_roles: []

};

selectedRole.sub\_roles.push(newSubRole);

const { nodes, edges } = transformDataToElements(roles);

setNodes(nodes);

setEdges(edges);

setNewRoleName('');

}

};

const handleDeleteRole = (roleToDelete) => {

const deleteRole = (rolesList) => {

return rolesList.filter(role => {

if (role.id === roleToDelete.id) return false;

role.sub\_roles = deleteRole(role.sub\_roles || []);

return true;

});

};

const updatedRoles = deleteRole(roles);

setRoles(updatedRoles);

const { nodes, edges } = transformDataToElements(updatedRoles);

setNodes(nodes);

setEdges(edges);

setSelectedRole(null);

};

return (

<div style={{ display: 'flex', height: '100vh' }}>

<div style={{ width: '70%', height: '100%' }}>

<ReactFlow

nodes={nodes}

edges={edges}

onNodeClick={onNodeClick}

style={{ width: '100%', height: '100%', backgroundColor: '#f0f0f0' }}

nodesDraggable // Enable nodes to be draggable

>

<MiniMap />

<Controls />

<Background />

</ReactFlow>

</div>

<div style={{ width: '30%', padding: '20px', borderLeft: '1px solid #ccc' }}>

{selectedRole && (

<>

<h3>Role Details</h3>

<p><strong>ID:</strong> {selectedRole.id}</p>

<p><strong>Name:</strong> {selectedRole.role\_name}</p>

<h4>Add Sub Role</h4>

<input

type="text"

placeholder="Sub Role Name"

value={newRoleName}

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

/>

<button onClick={handleAddRole}>Add Role</button>

<h4>Actions</h4>

<button onClick={() => handleDeleteRole(selectedRole)}>Delete Role</button>

</>

)}

</div>

</div>

);

};

export default RoleHierarchy;