

Employee CRUD operations using React and GraphQL

Practice Lab Handout

Problem Statement

The objective of this project is to develop a comprehensive Employee Management System that can handle the operations related to managing employees, departments, and designations within an organization. The system should provide a user-friendly interface for performing CRUD (Create, Read, Update, Delete) operations on employees, departments, and designations. Additionally, it should be able to handle associations between these entities, such as assigning employees to departments and designations, and managing hierarchical relationships among employees (e.g., managers and subordinates).

Feature	Description	Operations
Employee Management	Manage employee records.	Create, Read, Update, Delete, Find by ID, Find by Name
Department Management	Manage department records.	Create, Read, Update, Delete, Find by ID, Find by Name
Designation Management	Manage designation records.	Create, Read, Update, Delete, Find by ID, Find by Name

Non-Functional Requirements

Requirement	Description
User Interface	Clean, intuitive, and responsive UI using Bootstrap for styling.
Performance	Ensure quick response times for data retrieval and updates.
Scalability	Design backend to handle a growing number of records and increased traffic.
Maintainability	Modular and well-documented code for easy maintenance and future enhancements.

System Design

Frontend: React Application

Component	Description	Features
-----------	-------------	----------

Navbar	Navigation bar with links to different sections.	Home, Employees, Add Employee, Departments, Add Department, Designations, Add Designation
Employees	Displays a list of all employees.	Fetch and display employee data, Edit and Delete buttons
EmployeeForm	Form to add or edit employee details.	Input fields for Name, Email, Designation (dropdown), Department (dropdown), Manager ID
Departments	Displays a list of all departments.	Fetch and display department data, Edit and Delete buttons
DepartmentForm	Form to add or edit department details.	Input field for Name
Designations	Displays a list of all designations.	Fetch and display designation data, Edit and Delete buttons
DesignationForm	Form to add or edit designation details.	Input field for Name

Routing: React Router

Route	Component	Description
/	Welcome	Home page
/employees	Employees	List of employees
/add-employee	EmployeeForm	Form to add a new employee
/edit-employee/:id	EmployeeForm	Form to edit an existing employee
/departments	Departments	List of departments
/add-department	DepartmentForm	Form to add a new department
/edit-department/:id	DepartmentForm	Form to edit an existing department
/designations	Designations	List of designations
/add-designation	DesignationForm	Form to add a new designation
/edit-designation/:id	DesignationForm	Form to edit an existing designation

Backend: Apollo Server with Sequelize and MySQL

Model	Fields	Associations
Department	id (primary key), name	hasMany(Employee)
Designation	id (primary key), name	hasMany(Employee)
Employee	id (primary key), name, email, designation_id (foreign key), department_id (foreign key), manager_id (foreign key)	belongsTo(Department), belongsTo(Designation), belongsTo(Employee, as: 'Manager'), hasMany(Employee, as: 'Subordinates')

Mutation	Description
addDepartment(name: String!)	Add a new department
updateDepartment(id: Int!, name: String!)	Update an existing department
deleteDepartment(id: Int!)	Delete a department
addDesignation(name: String!)	Add a new designation
updateDesignation(id: Int!, name: String!)	Update an existing designation
deleteDesignation(id: Int!)	Delete a designation
addEmployee(name: String!, email: String!, designationId: Int!, departmentId: Int!, managerId: Int)	Add a new employee
updateEmployee(id: Int!, name: String!, email: String!, designationId: Int!, departmentId: Int!, managerId: Int)	Update an existing employee
deleteEmployee(id: Int!)	Delete an employee

Models

Type	Fields
Department	id, name
Designation	id, name
Employee	id, name, email, designation (object), department (object), manager (object), subordinates (list)

GraphQL Schema

Query	Description
departments	Retrieve all departments
department(id: Int!)	Retrieve a department by ID
designations	Retrieve all designations
designation(id: Int!)	Retrieve a designation by ID
employees	Retrieve all employees
employee(id: Int!)	Retrieve an employee by ID

Employee List Component (Employees.js)

Feature	Description
State Management	Manages the list of employees and loading state using useState.
Data Fetching	Fetches employees from the server when the component mounts using useEffect.
Handle Delete	Deletes an employee and updates the state.
UI Structure	Displays employees in a Bootstrap table with Edit and Delete buttons.

Navbar Component (Navbar.js)

Feature	Description
Navigation Bar	Renders a Bootstrap navigation bar with links to Home, Employees, Add Employee, Departments, Add Department, Designations, Add Designation.
Links	Uses Link component from react-router-dom for navigation without page reloads.

Employee Form Component (EmployeeForm.js)

Feature	Description
Form Management	Manages form state and validation using useForm from react-hook-form.
Data Fetching	Fetches departments, designations, and employee data (if editing) using useEffect.
Form Submission	Handles form submission to add or update an employee.
UI Structure	Displays a form with input fields for Name, Email, Designation, Department, and Manager ID.

Frontend: React Application

1. Project Setup

1. Create a new React application:

npx create-react-app graphql-client

cd graphql-client

This command creates a new React application called graphql-client and navigates into the project directory.

2. Install necessary dependencies:

npm install @apollo/client graphql react-hook-form react-router-dom bootstrap

- a. @apollo/client: Apollo Client for interacting with GraphQL APIs.
- b. graphql: Required for using GraphQL.
- c. react-hook-form: For managing forms in React.
- d. react-router-dom: For routing in React.
- e. bootstrap: For styling the UI.

1. Include Bootstrap CSS in index.js:

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import { ApolloProvider } from '@apollo/client';
import client from './apolloClient'; // Ensure this path is correct
import App from './App';
import 'bootstrap/dist/css/bootstrap.css';
```

```
const container = document.getElementById('root');
const root = createRoot(container);
```

```
root.render(
  <ApolloProvider client={client}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </ApolloProvider>
);
```

```
import React from 'react';
```

Explanation:

- This line imports the React library, which is essential for creating React components and using React features in your application.

```
import { createRoot } from 'react-dom/client';
```

Explanation:

- This line imports the createRoot function from the react-dom/client module. createRoot is part of React 18's new concurrent rendering API and replaces the deprecated ReactDOM.render method. It is used to create a root where the React application will be rendered.

```
import { ApolloProvider } from '@apollo/client';
```

Explanation:

- This line imports the ApolloProvider component from the @apollo/client package. ApolloProvider is a higher-order component used to provide the Apollo Client instance to the entire React component tree, enabling GraphQL functionalities.

```
import client from './apolloClient'; // Ensure this path is correct
```

Explanation:

- This line imports the Apollo Client instance from the apolloClient.js file located in the same directory (.). This client instance is configured to connect to your GraphQL server and handle network requests.

```
import App from './App';
```

Explanation:

- This line imports the main App component from the App.js file. This component is the root component of your React application where all other components are typically nested.

```
import 'bootstrap/dist/css/bootstrap.css';
```

Explanation:

- This line imports the Bootstrap CSS library. By including this line, you apply Bootstrap's styles globally to your React application, allowing you to use Bootstrap's CSS classes for responsive design and styling.

```
const container = document.getElementById('root');
```

Explanation:

- This line selects the DOM element with the id root. This is the container element where the React application will be mounted. Typically, this element is defined in the index.html file of your public directory.

```
const root = createRoot(container);
```

Explanation:

- This line creates a root using the createRoot function and assigns it to the variable root. This root is used to manage the React component tree and render the application within the selected container.

```
root.render(  
  <ApolloProvider client={client}>  
    <React.StrictMode>  
      <App />  
    </React.StrictMode>  
  </ApolloProvider>  
);
```

Explanation:

- **root.render(...):** Calls the render method on the root object to render the React application.
- **<ApolloProvider client={client}>:** Wraps the application with ApolloProvider and passes the Apollo Client instance as a prop. This ensures that all components within the application can access the Apollo Client for making GraphQL requests.
- **<React.StrictMode>:** Wraps the application with React.StrictMode. This is a development mode feature that helps identify potential problems in an application by performing additional checks and warnings.
- **<App />:** The main App component of the React application. This is where the application's component tree begins.

Summary

1. **React and Dependencies Import:** Imports React and necessary dependencies like createRoot, ApolloProvider, Apollo Client instance, and the main App component.
2. **Bootstrap CSS:** Imports Bootstrap CSS for styling.
3. **Container Element:** Selects the root DOM element where the React app will be rendered.

4. **Create Root:** Uses `createRoot` to create a root for the application.
5. **Render Application:** Renders the application by wrapping it with `ApolloProvider` (for GraphQL) and `React.StrictMode` (for additional checks during development) and then mounts the App component within this setup.

2. Setup Apollo Client (`apolloClient.js`):

```
// src/apolloClient.js
import { ApolloClient, InMemoryCache, HttpLink } from '@apollo/client';
const client = new ApolloClient({
  link: new HttpLink({
    uri: 'http://localhost:4000/', // Ensure this URL is correct
  }),
  cache: new InMemoryCache(),
});
export default client;
```

Explanation:

- **Creating the Apollo Client Instance:**
 - **new ApolloClient:** This creates a new instance of Apollo Client.
 - **Configuration Object:** Passed as an argument to `ApolloClient`, this object configures the client with necessary details like the GraphQL server URI and caching strategy.
 - **link Property:**
 - **new HttpLink({ uri: 'http://localhost:4000/' }):** Creates a new HTTP link with the specified URI. This URI should point to your GraphQL server. In this example, it's set to 'http://localhost:4000/', which implies that your GraphQL server is running locally on port 4000. If your GraphQL server is hosted elsewhere, replace this URI with the appropriate endpoint.
 - **cache Property:**
 - **new InMemoryCache():** Initializes the cache using Apollo's in-memory caching system. This helps to store and retrieve query results, which can improve the performance of your application by reducing the number of network requests. The `InMemoryCache` is a flexible and powerful tool for caching GraphQL query results.

```
export default client;
```

Explanation:

- **Exporting the Client Instance:**
 - This line exports the Apollo Client instance so that it can be imported and used in other parts of your application. Typically, this client instance will be provided to the entire React component tree using the `ApolloProvider` component, enabling all components to perform GraphQL operations.

Summary

1. **Imports:** The necessary Apollo Client modules (ApolloClient, InMemoryCache, HttpLink) are imported.
2. **Apollo Client Configuration:**
 - **HttpLink:** Specifies the URI of the GraphQL server. This should be updated to the actual endpoint of your GraphQL server.
 - **InMemoryCache:** Sets up caching for query results to improve performance.
3. **Export:** The configured Apollo Client instance is exported for use in other parts of the application, particularly in the ApolloProvider wrapper in your main application file.

By configuring the Apollo Client in this way, you ensure that your React application is set up to interact efficiently with a GraphQL server, leveraging caching to optimize performance and reduce unnecessary network requests.

2. Navigation Bar Component

- **Navbar.js:**

```
import React from 'react';
import { Link } from 'react-router-dom';

const Navbar = () => {
  return (
    <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
      <div className="container-fluid">
        <Link className="navbar-brand" to="/">Employee Management</Link>
        <div className="collapse navbar-collapse" id="navbarNav">
          <ul className="navbar-nav">
            <li className="nav-item">
              <Link className="nav-link" to="/">Home</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/employees">Employees</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/add-employee">Add Employee</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/departments">Departments</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/add-department">Add Department</Link>
            </li>
            <li className="nav-item">
              <Link className="nav-link" to="/designations">Designations</Link>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  );
}
```



```

    </li>
    <li className="nav-item">
      <Link className="nav-link" to="/add-designation">Add Designation</Link>
    </li>
  </ul>
</div>
</div>
</nav>
);
};
export default Navbar;

```

- **Component Setup:**
 - The Navbar component renders a Bootstrap navigation bar.
 - The Link component from react-router-dom is used to create navigation links without reloading the page.
- **Navigation Bar Structure:**
 - navbar navbar-expand-lg navbar-dark bg-dark: Bootstrap classes for a dark-themed, expandable navigation bar.
 - The navigation bar contains links to Home, Employees, Add Employee, Departments, Add Department, Designations, and Add Designation.

3. Employee List Component

```

import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import employeeService from '../services/employeeService';

```

```

const Employees = () => {
  const [employees, setEmployees] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchEmployees = async () => {
      const data = await employeeService.getEmployees();
      setEmployees(data);
      setLoading(false);
    };
    fetchEmployees();
  }, []);

```

```

const handleDelete = async (id) => {
  await employeeService.deleteEmployee(id);
  setEmployees(employees.filter(employee => employee.id !== id));
};

if (loading) return <p>Loading...</p>;

return (
  <div>
    <h2>Employees</h2>
    <table className="table table-striped">
      <thead>
        <tr>
          <th>Name</th>
          <th>Email</th>
          <th>Designation</th>
          <th>Department</th>
          <th>Manager</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {employees.map((employee) => (
          <tr key={employee.id}>
            <td>{employee.name}</td>
            <td>{employee.email}</td>
            <td>{employee.designation.title}</td>
            <td>{employee.department.name}</td>
            <td>{employee.manager ? employee.manager.name : 'N/A'}</td>
            <td>
              <Link className="btn btn-primary btn-sm mr-2" style={{margin:5}} to={` /edit-
employee/${employee.id}`}>Edit</Link>
              <button className="btn btn-danger btn-sm" onClick={() =>
handleDelete(employee.id)}>Delete</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
);
};

export default Employees;

```

- **Employees.js:**
 - **Component Setup:**
 - The Employees component fetches and displays a list of employees.
 - `useEffect` is used to fetch employees from the server when the component mounts.
 - **Data Fetching:**
 - `employeeService.getEmployees()`: Fetches the list of employees.
 - `setEmployees(data)`: Updates the state with the fetched employees.
 - **Data Deletion:**
 - `handleDelete`: Deletes an employee and updates the state.
 - **UI Structure:**
 - A Bootstrap table displays the employees with columns for Name, Email, Designation, Department, Manager, and Actions.
 - Each row includes Edit and Delete buttons.

4. Employee Form Component

- **EmployeeForm.js:**

```
// src/components/EmployeeForm.js
import React, { useEffect, useState } from 'react';
import { useForm } from 'react-hook-form';
import { useNavigate, useParams } from 'react-router-dom';
import employeeService from '../services/employeeService';

const EmployeeForm = () => {
  const { id } = useParams();
  const isEditing = !!id;
  const navigate = useNavigate();
  const { register, handleSubmit, setValue } = useForm();
  const [departments, setDepartments] = useState([]);
  const [designations, setDesignations] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      const departmentsData = await employeeService.getDepartments();
      const designationsData = await employeeService.getDesignations();
      setDepartments(departmentsData);
      setDesignations(designationsData);
    }

    if (isEditing) {
```

```

    const employeeData = await employeeService.getEmployee(parseInt(id));
    const { name, email, designation, department, manager } = employeeData;
    setValue('name', name);
    setValue('email', email);
    setValue('designationId', designation.id);
    setValue('departmentId', department.id);
    setValue('managerId', manager ? manager.id : '');
  }
  setLoading(false);
};
fetchData();
}, [id, isEditing, setValue]);

const onSubmit = async (formData) => {
  const variables = {
    name: formData.name,
    email: formData.email,
    designationId: parseInt(formData.designationId),
    departmentId: parseInt(formData.departmentId),
    managerId: formData.managerId ? parseInt(formData.managerId) : null,
  };
  if (isEditing) {
    await employeeService.updateEmployee({ id: parseInt(id), ...variables });
  } else {
    await employeeService.addEmployee(variables);
  }
  navigate('/employees');
};

if (loading) return <p>Loading...</p>;

return (
  <div className='row'>
    <div className='col-4'></div>
    <div className="col-12 col-md-4 col-xl-4">
      <h2>{isEditing ? 'Edit Employee' : 'Add Employee'}</h2>
      <form onSubmit={handleSubmit(onSubmit)}>
        <div className="form-group">
          <label>Name:</label>
          <input className="form-control" {...register('name', { required: true })} />
        </div>
        <div className="form-group">
          <label>Email:</label>
          <input className="form-control" {...register('email', { required: true })} />

```

```

</div>
<div className="form-group">
  <label>Designation:</label>
  <select className="form-control" {...register('designationId', { required: true })}>
    {designations.map((designation) => (
      <option key={designation.id} value={designation.id}>{designation.title}</option>
    ))}
  </select>
</div>
<div className="form-group">
  <label>Department:</label>
  <select className="form-control" {...register('departmentId', { required: true })}>
    {departments.map((department) => (
      <option key={department.id} value={department.id}>{department.name}</option>
    ))}
  </select>
</div>
<div className="form-group">
  <label>Manager ID:</label>
  <input className="form-control" {...register('managerId')} />
</div>
<button type="submit" className="btn btn-primary mt-3">{isEditing ? 'Update
Employee' : 'Add Employee'}</button>
</form>
</div>
</div>
);
};

```

export default EmployeeForm;

- **Component Setup:**
 - The EmployeeForm component handles both adding and editing employees.
 - useParams is used to get the employee ID from the URL if editing.
 - useForm from react-hook-form manages the form state and validation.
- **Data Fetching:**
 - employeeService.getDepartments() and employeeService.getDesignations(): Fetch departments and designations for dropdowns.
 - If editing, fetch employee data and set form values using setValue.
- **Form Submission:**
 - onSubmit: Handles form submission to either add or update an employee.
- **UI Structure:**

- The form contains fields for Name, Email, Designation, Department, and Manager ID.
- The form is styled with Bootstrap classes.

5. AppRoutes.js

```
// src/AppRoutes.js
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Employees from './components/Employees';
import EmployeeForm from './components/EmployeeForm';
import Departments from './components/Departments';
import DepartmentForm from './components/DepartmentForm';
import Designations from './components/Designations';
import DesignationForm from './components/DesignationForm';

function AppRoutes() {
  return (
    <Routes>
      <Route path="/" element={<div>Welcome to GraphQL Client</div>} />
      <Route path="/employees" element={<Employees />} />
      <Route path="/add-employee" element={<EmployeeForm />} />
      <Route path="/edit-employee/:id" element={<EmployeeForm />} />
      <Route path="/departments" element={<Departments />} />
      <Route path="/add-department" element={<DepartmentForm />} />
      <Route path="/edit-department/:id" element={<DepartmentForm />} />
      <Route path="/designations" element={<Designations />} />
      <Route path="/add-designation" element={<DesignationForm />} />
      <Route path="/edit-designation/:id" element={<DesignationForm />} />
    </Routes>
  );
}

export default AppRoutes;
```

6. Update App.js for Routing

- **App.js:**

```
// src/App.js
import React from 'react';
import { BrowserRouter as Router } from 'react-router-dom';
import Navbar from './components/Navbar';
import './App.css';
import AppRoutes from './AppRoutes';
```

```

function App() {
  return (
    <Router>
      <div className="App">
        <Navbar />
        <main className="container mt-3">
          <AppRoutes />
        </main>
      </div>
    </Router>
  );
}
export default App;

```

7. Departments.js

This component is responsible for displaying a list of departments, allowing users to edit or delete each department. It uses Apollo Client's `useQuery` and `useMutation` hooks to interact with the GraphQL API.

```

// Import necessary modules and components
import React from 'react';
import { useQuery, useMutation } from '@apollo/client';
import { Link } from 'react-router-dom';
import { GET_DEPARTMENTS, DELETE_DEPARTMENT } from '../services/employeeService';

const Departments = () => {
  // Execute the GET_DEPARTMENTS query to fetch the list of departments
  const { loading, error, data } = useQuery(GET_DEPARTMENTS);

  // Define the deleteDepartment mutation and refetch the departments query after
  // deletion
  const [deleteDepartment] = useMutation(DELETE_DEPARTMENT, {
    refetchQueries: [{ query: GET_DEPARTMENTS }],
  });

  // Handle the delete action
  const handleDelete = async (id) => {
    try {
      await deleteDepartment({ variables: { id } });
    } catch (error) {
      console.error('Error:', error);
    }
  }
}

```

```

    }
  };

  // Display loading message while data is being fetched
  if (loading) return <p>Loading...</p>;

  // Display error message if there is an error in fetching data
  if (error) return <p>Error: {error.message}</p>;

  // Render the list of departments
  return (
    <div>
      <h2>Departments</h2>
      <table className="table table-striped">
        <thead>
          <tr>
            <th>Name</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {data.departments.map((department) => (
            <tr key={department.id}>
              <td>{department.name}</td>
              <td>
                <Link className="btn btn-primary btn-sm mr-2" to={` /edit-
department/${department.id}`}>
                  Edit
                </Link>
                <button
                  className="btn btn-danger btn-sm"
                  style={{ margin: 5 }}
                  onClick={() => handleDelete(department.id)}
                >
                  Delete
                </button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

```


export default Departments;

Detailed Explanation

1. Imports:

- React: The core React library.
- `useQuery` and `useMutation` from `@apollo/client`: Hooks for executing GraphQL queries and mutations.
- Link from `react-router-dom`: A component for creating navigational links.
- `GET_DEPARTMENTS` and `DELETE_DEPARTMENT` from `../services/employeeService`: GraphQL queries and mutations defined in a separate service file.

2. `useQuery` Hook:

- `const { loading, error, data } = useQuery(GET_DEPARTMENTS);`
 - Executes the `GET_DEPARTMENTS` query to fetch the list of departments.
 - `loading`, `error`, and `data` are destructured from the result of `useQuery`.

3. `useMutation` Hook:

- `const [deleteDepartment] = useMutation(DELETE_DEPARTMENT, { refetchQueries: [{ query: GET_DEPARTMENTS }] });`
 - Defines the `deleteDepartment` mutation using the `DELETE_DEPARTMENT` GraphQL mutation.
 - `refetchQueries`: Automatically refetches the `GET_DEPARTMENTS` query after a successful mutation to update the list of departments.

4. `handleDelete` Function:

- `const handleDelete = async (id) => { ... };`
 - A function to handle the deletion of a department.
 - Calls the `deleteDepartment` mutation with the department's id.
 - Catches and logs any errors that occur during the mutation.

5. Conditional Rendering:

- `if (loading) return <p>Loading...</p>;`
 - Displays a loading message while the data is being fetched.
- `if (error) return <p>Error: {error.message}</p>;`
 - Displays an error message if there is an error in fetching the data.

6. Render the Departments List:

- `return (...);`
 - Renders a table with the list of departments.
 - Maps over the `data.departments` array to create table rows for each department.
 - Each row has an "Edit" link and a "Delete" button.
 - The "Delete" button calls the `handleDelete` function when clicked.

GraphQL Queries and Mutations

`GET_DEPARTMENTS`

This query fetches the list of departments. It should be defined in your `employeeService.js` file.

```
import { gql } from '@apollo/client';
```

```
export const GET_DEPARTMENTS = gql`
  query GetDepartments {
    departments {
      id
      name
    }
  }
`;
```

DELETE_DEPARTMENT

This mutation deletes a department by its id. It should also be defined in your employeeService.js file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const DELETE_DEPARTMENT = gql`
  mutation DeleteDepartment($id: ID!) {
    deleteDepartment(id: $id) {
      id
    }
  }
`;
```

Summary

- The Departments component uses Apollo Client's useQuery and useMutation hooks to fetch and manipulate department data via GraphQL.
- It conditionally renders loading and error states.
- It provides functionality to delete departments and automatically refetches the department list after a deletion.
- It uses react-router-dom's Link component for navigation to the edit department page.

7. DepartmentForm.js : DepartmentForm Component

This component is responsible for both adding a new department and editing an existing one. It uses React Hook Form for form handling, Apollo Client for interacting with the GraphQL API, and React Router for navigation.

// Import necessary modules and components

```
import React, { useEffect } from 'react';
import { useForm } from 'react-hook-form';
import { useNavigate, useParams } from 'react-router-dom';
import { useQuery, useMutation } from '@apollo/client';
```

```
import { GET_DEPARTMENT, ADD_DEPARTMENT, UPDATE_DEPARTMENT, GET_DEPARTMENTS }  
from '../services/employeeService';
```

```
const DepartmentForm = () => {  
  const { id } = useParams();  
  const isEditing = !!id; // Determine if the form is in editing mode based on the presence of an  
  id  
  const navigate = useNavigate();  
  const { register, handleSubmit, setValue } = useForm();  
  const { data } = useQuery(GET_DEPARTMENT, {  
    variables: { id: parseInt(id) },  
    skip: !isEditing, // Skip the query if not editing  
  });
```

```
  const [addDepartment] = useMutation(ADD_DEPARTMENT, {  
    refetchQueries: [{ query: GET_DEPARTMENTS }],  
    onCompleted: () => navigate('/departments'), // Redirect after successful mutation  
  });
```

```
  const [updateDepartment] = useMutation(UPDATE_DEPARTMENT, {  
    refetchQueries: [{ query: GET_DEPARTMENTS }],  
    onCompleted: () => navigate('/departments'), // Redirect after successful mutation  
  });
```

```
  useEffect(() => {  
    if (isEditing && data) {  
      setValue('name', data.department.name); // Pre-fill the form with existing department data  
    }  
  }, [isEditing, data, setValue]);
```

```
  const onSubmit = async (formData) => {  
    try {  
      if (isEditing) {  
        await updateDepartment({ variables: { id: parseInt(id), name: formData.name } });  
      } else {  
        await addDepartment({ variables: { name: formData.name } });  
      }  
    } catch (error) {  
      console.error('Error:', error);  
    }  
  };
```

```
  return (  
    <div className='row'>
```

```

<div className='col-4'></div>
<div className="col-12 col-md-4 col-xl-4">
  <h2>{isEditing ? 'Edit Department' : 'Add Department'}</h2>
  <form onSubmit={handleSubmit(onSubmit)}>
    <div className="form-group">
      <label>Name:</label>
      <input className="form-control" {...register('name', { required: true })} />
    </div>
    <button type="submit" className="btn btn-primary mt-3">{isEditing ? 'Update
Department' : 'Add Department'}</button>
  </form>
</div>
</div>
);
};

```

```
export default DepartmentForm;
```

Detailed Explanation

- **Imports:**
 - **React:** Core library for building user interfaces.
 - **useForm:** React Hook Form library for handling form state and validation.
 - **useNavigate** and **useParams:** Hooks from React Router for navigation and accessing URL parameters.
 - **useQuery** and **useMutation:** Apollo Client hooks for fetching and mutating data.
 - **GraphQL Queries and Mutations:** Imported from a service file where GraphQL operations are defined.
- **Component Setup:**
 - `const { id } = useParams();` Extracts the id parameter from the URL.
 - `const isEditing = !!id;` Determines if the form is in edit mode based on whether an id is present.
 - `const navigate = useNavigate();` Hook for programmatic navigation.
- **Form Setup with React Hook Form:**
 - `const { register, handleSubmit, setValue } = useForm();` Initializes form handling methods.
- **Fetch Department Data for Editing:**
 - `const { data } = useQuery(GET_DEPARTMENT, { variables: { id: parseInt(id) }, skip: !isEditing });`
 - Fetches department data if `isEditing` is true.
 - Uses the `skip` option to avoid executing the query when adding a new department.
- **Mutations:**
 - `const [addDepartment] = useMutation(ADD_DEPARTMENT, { refetchQueries: [{ query: GET_DEPARTMENTS }], onCompleted: () => navigate('/departments') });`

- Mutation to add a new department.
 - Refetches the department list after mutation.
 - Navigates to the department list page upon completion.
- `const [updateDepartment] = useMutation(UPDATE_DEPARTMENT, { refetchQueries: [{ query: GET_DEPARTMENTS }], onCompleted: () => navigate('/departments') });`
 - Mutation to update an existing department.
 - Refetches the department list after mutation.
 - Navigates to the department list page upon completion.
- **Effect Hook for Setting Form Values:**
 - `useEffect(() => { if (isEditing && data) { setValue('name', data.department.name); } }, [isEditing, data, setValue]);`
 - Prefills the form with the existing department's name if in edit mode and data is available.
- **Form Submission Handler:**
 - `const onSubmit = async (formData) => { ... }`
 - Handles form submission.
 - Calls the appropriate mutation (`addDepartment` or `updateDepartment`) based on `isEditing`.
- **Form Rendering:**
 - `return (<div className='row'> ... </div>);`
 - Renders the form with a conditionally rendered heading and submit button text based on `isEditing`.
 - Uses Bootstrap classes for styling.

GraphQL Queries and Mutations

GET_DEPARTMENT

Fetches the details of a single department.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const GET_DEPARTMENT = gql`
  query GetDepartment($id: ID!) {
    department(id: $id) {
      id
      name
    }
  }
`;
```

ADD_DEPARTMENT

Mutation to add a new department.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const ADD_DEPARTMENT = gql`
  mutation AddDepartment($name: String!) {
    addDepartment(name: $name) {
      id
      name
    }
  }
`;
```

UPDATE_DEPARTMENT

Mutation to update an existing department.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const UPDATE_DEPARTMENT = gql`
  mutation UpdateDepartment($id: ID!, $name: String!) {
    updateDepartment(id: $id, name: $name) {
      id
      name
    }
  }
`;
```

GET_DEPARTMENTS

Fetches the list of all departments.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const GET_DEPARTMENTS = gql`
  query GetDepartments {
    departments {
      id
      name
    }
  }
`;
```

Summary

The DepartmentForm component provides a form for both adding and editing departments. It uses React Hook Form for form state management, Apollo Client for interacting with the GraphQL API, and React Router for handling navigation. The component fetches data conditionally based on whether it is in edit mode and handles form submission by calling the appropriate GraphQL

8. Designation Component : Designations.js

This component is responsible for displaying a list of designations, allowing users to edit or delete each designation. It uses Apollo Client's useQuery and useMutation hooks to interact with the GraphQL API.

```
// Import necessary modules and components
import React from 'react';
import { useQuery, useMutation } from '@apollo/client';
import { Link } from 'react-router-dom';
import { GET_DESIGNATIONS, DELETE_DESIGNATION } from '../services/employeeService';

const Designations = () => {
  // Execute the GET_DESIGNATIONS query to fetch the list of designations
  const { loading, error, data } = useQuery(GET_DESIGNATIONS);

  // Define the deleteDesignation mutation and refetch the designations query after deletion
  const [deleteDesignation] = useMutation(DELETE_DESIGNATION, {
    refetchQueries: [{ query: GET_DESIGNATIONS }],
  });

  // Handle the delete action
  const handleDelete = async (id) => {
    try {
      await deleteDesignation({ variables: { id } });
    } catch (error) {
      console.error('Error:', error);
    }
  };

  // Display loading message while data is being fetched
  if (loading) return <p>Loading...</p>;

  // Display error message if there is an error in fetching data
  if (error) return <p>Error: {error.message}</p>;

  // Render the list of designations
  return (
    <div>
      <h2>Designations</h2>
      <table className="table table-striped">
        <thead>
          <tr>
            <th>Title</th>
```

```

        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      {data.designations.map((designation) => (
        <tr key={designation.id}>
          <td>{designation.title}</td>
          <td>
            <Link className="btn btn-primary btn-sm mr-2" to={`/edit-
designation/${designation.id}`}>
              Edit
            </Link>
            <button
              className="btn btn-danger btn-sm"
              style={{ margin: 5 }}
              onClick={() => handleDelete(designation.id)}
            >
              Delete
            </button>
          </td>
        </tr>
      )})}
    </tbody>
  </table>
</div>
);
};

```

export default Designations;

Detailed Explanation

1. Imports:

- **React**: The core React library.
- **useQuery** and **useMutation** from **@apollo/client**: Hooks for executing GraphQL queries and mutations.
- **Link** from **react-router-dom**: A component for creating navigational links.
- **GET_DESIGNATIONS** and **DELETE_DESIGNATION** from **../services/employeeService**: GraphQL queries and mutations defined in a separate service file.

2. useQuery Hook:

- `const { loading, error, data } = useQuery(GET_DESIGNATIONS);`
 - Executes the **GET_DESIGNATIONS** query to fetch the list of designations.
 - `loading`, `error`, and `data` are destructured from the result of `useQuery`.

3. useMutation Hook:

- `const [deleteDesignation] = useMutation(DELETE_DEPARTMENT, { refetchQueries: [{ query: GET_DEPARTMENTS }] });`
 - Defines the `deleteDesignation` mutation using the `DELETE_DEPARTMENT` GraphQL mutation.
 - `refetchQueries`: Automatically refetches the `GET_DESIGNATIONS` query after a successful mutation to update the list of designations.
- 4. **handleDelete Function:**
 - `const handleDelete = async (id) => { ... }`
 - A function to handle the deletion of a designation.
 - Calls the `deleteDesignation` mutation with the designation's id.
 - Catches and logs any errors that occur during the mutation.
- 5. **Conditional Rendering:**
 - `if (loading) return <p>Loading...</p>;`
 - Displays a loading message while the data is being fetched.
 - `if (error) return <p>Error: {error.message}</p>;`
 - Displays an error message if there is an error in fetching the data.
- 6. **Render the Designations List:**
 - `return (<div> ... </div>);`
 - Renders a table with the list of designations.
 - Maps over the `data.designations` array to create table rows for each designation.
 - Each row has an "Edit" link and a "Delete" button.
 - The "Delete" button calls the `handleDelete` function when clicked.

GraphQL Queries and Mutations

GET_DESIGNATIONS

This query fetches the list of designations. It should be defined in your `employeeService.js` file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const GET_DESIGNATIONS = gql`
  query GetDesignations {
    designations {
      id
      title
    }
  }
`;
```

DELETE_DEPARTMENT

This mutation deletes a department by its id. It should also be defined in your `employeeService.js` file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const DELETE_DEPARTMENT = gql`
  mutation DeleteDepartment($id: ID!) {
    deleteDepartment(id: $id) {
      id
    }
  }
`;
```

Summary

- The Designations component uses Apollo Client's useQuery and useMutation hooks to fetch and manipulate designation data via GraphQL.
- It conditionally renders loading and error states.
- It provides functionality to delete designations and automatically refetches the designation list after a deletion.
- It uses react-router-dom's Link component for navigation to the edit designation page.

9. DesignationForm Component : DesignationForm.js

This component handles both adding a new designation and editing an existing one. It uses React Hook Form for managing form state, Apollo Client for interacting with a GraphQL API, and React Router for navigation.

```
// Import necessary modules and components
import React, { useEffect } from 'react';
import { useForm } from 'react-hook-form';
import { useNavigate, useParams } from 'react-router-dom';
import { useQuery, useMutation } from '@apollo/client';
import { GET_DESIGNATION, ADD_DESIGNATION, UPDATE_DESIGNATION, GET_DESIGNATIONS }
from '../services/employeeService';
```

```
const DesignationForm = () => {
  const { id } = useParams(); // Get the designation ID from the URL
  const isEditing = !!id; // Check if the form is in edit mode based on the presence of an ID
  const navigate = useNavigate(); // Hook for navigation
  const { register, handleSubmit, setValue } = useForm(); // React Hook Form hooks
  const { data } = useQuery(GET_DESIGNATION, {
    variables: { id: parseInt(id) },
    skip: !isEditing, // Skip the query if not editing
  });
```

```
// Mutation to add a new designation
const [addDesignation, { error: addError }] = useMutation(ADD_DESIGNATION, {
  refetchQueries: [{ query: GET_DESIGNATIONS }],
```

```

    onCompleted: () => navigate('/designations'), // Navigate to the designations list upon
completion
  });

  // Mutation to update an existing designation
  const [updateDesignation, { error: updateError }] = useMutation(UPDATE_DESIGNATION, {
    refetchQueries: [{ query: GET_DESIGNATIONS }],
    onCompleted: () => navigate('/designations'), // Navigate to the designations list upon
completion
  });

  // Effect to pre-fill the form if editing
  useEffect(() => {
    if (isEditing && data) {
      setValue('title', data.designation.title); // Set the form value to the existing designation's
title
    }
  }, [isEditing, data, setValue]);

  // Handle form submission
  const onSubmit = async (formData) => {
    try {
      if (isEditing) {
        await updateDesignation({ variables: { id: parseInt(id), title: formData.title } }); // Update
existing designation
      } else {
        await addDesignation({ variables: { title: formData.title } }); // Add new designation
      }
    } catch (error) {
      console.error('Error:', error); // Log any errors
    }
  };

  return (
    <div className='row'>
      <div className='col-4'></div>
      <div className="col-12 col-md-4 col-xl-4">
        <h2>{isEditing ? 'Edit Designation' : 'Add Designation'}</h2>
        <form onSubmit={handleSubmit(onSubmit)}>
          <div className="form-group">
            <label>Title:</label>
            <input className="form-control" {...register('title', { required: true })} /> // Input for
designation title
          </div>

```

```

    <button type="submit" className="btn btn-primary mt-3">
      {isEditing ? 'Update Designation' : 'Add Designation'}
    </button>
  </form>
  {addError && <p>Error adding designation: {addError.message}</p>}
  {updateError && <p>Error updating designation: {updateError.message}</p>}
</div>
</div>
);
};

```

export default DesignationForm;

Detailed Explanation

1. Imports:

- **React:** Core React library.
- **useForm:** Hook from React Hook Form for managing form state and validation.
- **useNavigate** and **useParams:** Hooks from React Router for navigation and accessing URL parameters.
- **useQuery** and **useMutation:** Hooks from Apollo Client for executing GraphQL queries and mutations.
- **GraphQL Queries and Mutations:** Imported from the service file where GraphQL operations are defined.

2. Component Setup:

- `const { id } = useParams();` Extracts the id parameter from the URL.
- `const isEditing = !!id;` Determines if the form is in edit mode based on whether an id is present.
- `const navigate = useNavigate();` Hook for programmatic navigation.
- `const { register, handleSubmit, setValue } = useForm();` Initializes form handling methods.

3. Fetch Designation Data for Editing:

- `const { data } = useQuery(GET_DESIGNATION, { variables: { id: parseInt(id) }, skip: !isEditing });`
 - Fetches designation data if isEditing is true.
 - Uses the skip option to avoid executing the query when adding a new designation.

4. Mutations:

- `const [addDesignation, { error: addError }] = useMutation(ADD_DEPARTMENT, { refetchQueries: [{ query: GET_DEPARTMENTS }], onComplete: () => navigate('/departments') });`
 - Defines the addDesignation mutation using the ADD_DEPARTMENT GraphQL mutation.
 - `refetchQueries:` Automatically refetches the GET_DESIGNATIONS query after a successful mutation to update the list of designations.

- onCompleted: Navigates to the designations list page upon successful completion of the mutation.
 - const [updateDesignation, { error: updateError }] = useMutation(UPDATE_DEPARTMENT, { refetchQueries: [{ query: GET_DEPARTMENTS }], onCompleted: () => navigate('/departments') });
 - Defines the updateDesignation mutation using the UPDATE_DEPARTMENT GraphQL mutation.
 - refetchQueries: Automatically refetches the GET_DESIGNATIONS query after a successful mutation to update the list of designations.
 - onCompleted: Navigates to the designations list page upon successful completion of the mutation.
- 5. Effect Hook for Setting Form Values:**
- useEffect(() => { if (isEditing && data) { setValue('title', data.designation.title); } }, [isEditing, data, setValue]);
 - Prefills the form with the existing designation's title if in edit mode and data is available.
- 6. Form Submission Handler:**
- const onSubmit = async (formData) => { ... }
 - Handles form submission.
 - Calls the appropriate mutation (addDesignation or updateDesignation) based on isEditing.
- 7. Form Rendering:**
- return (<div className='row'> ... </div>);
 - Renders the form with a conditionally rendered heading and submit button text based on isEditing.
 - Uses Bootstrap classes for styling.
 - Displays any errors that occur during the mutation.

GraphQL Queries and Mutations

GET_DESIGNATION

This query fetches the details of a single designation. It should be defined in your employeeService.js file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const GET_DESIGNATION = gql`
  query GetDesignation($id: ID!) {
    designation(id: $id) {
      id
      title
    }
  }
`;
```

ADD_DESIGNATION

This mutation adds a new designation. It should also be defined in your employeeService.js file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const ADD_DESIGNATION = gql`  
  mutation AddDesignation($title: String!) {  
    addDesignation(title: $title) {  
      id  
      title  
    }  
  }  
`;
```

UPDATE_DESIGNATION

This mutation updates an existing designation. It should also be defined in your employeeService.js file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const UPDATE_DESIGNATION = gql`  
  mutation UpdateDesignation($id: ID!, $title: String!) {  
    updateDesignation(id: $id, title: $title) {  
      id  
      title  
    }  
  }  
`;
```

GET_DESIGNATIONS

This query fetches the list of all designations. It should also be defined in your employeeService.js file.

javascript

Copy code

```
import { gql } from '@apollo/client';
```

```
export const GET_DESIGNATIONS = gql`  
  query GetDesignations {  
    designations {  
      id  
      title  
    }  
  }  
`;
```

Summary

The DesignationForm component handles both adding and editing designations. It uses React Hook Form for managing form state, Apollo Client for interacting with the GraphQL API, and React Router for navigation. The component fetches data conditionally based on whether it is in edit mode and handles form submission by calling the appropriate GraphQL mutations. Errors during mutations are handled and displayed to the user.

10. **employeeService.js**

```
// src/services/employeeService.js
import { gql } from '@apollo/client';
import client from '../apolloClient'; // Import the Apollo Client instance
```

```
// Employee Queries and Mutations
```

```
export const GET_EMPLOYEES = gql`
  query GetEmployees {
    employees {
      id
      name
      email
      designation {
        title
      }
      department {
        name
      }
      manager {
        name
      }
    }
  }
`;
```

```
export const GET_EMPLOYEE = gql`
  query GetEmployee($id: Int!) {
    employee(id: $id) {
      id
      name
      email
      designation {
        id
        title
      }
      department {
        id
        name
      }
    }
  }
`;
```

```

    }
    manager {
      id
      name
    }
  }
}
`;

```

```

export const ADD_EMPLOYEE = gql`
  mutation AddEmployee($name: String!, $email: String!, $designationId: Int!, $departmentId:
Int!, $managerId: Int) {
    addEmployee(name: $name, email: $email, designationId: $designationId, departmentId:
$departmentId, managerId: $managerId) {
      id
      name
      email
      designation {
        title
      }
      department {
        name
      }
      manager {
        name
      }
    }
  }
`;

```

```

export const UPDATE_EMPLOYEE = gql`
  mutation UpdateEmployee($id: Int!, $name: String!, $email: String!, $designationId: Int!,
$departmentId: Int!, $managerId: Int) {
    updateEmployee(id: $id, name: $name, email: $email, designationId: $designationId,
departmentId: $departmentId, managerId: $managerId) {
      id
      name
      email
      designation {
        title
      }
      department {
        name
      }
    }
  }
`;

```



```
    manager {  
      name  
    }  
  }  
}  
`;  
`;
```

```
export const DELETE_EMPLOYEE = gql`  
  mutation DeleteEmployee($id: Int!) {  
    deleteEmployee(id: $id) {  
      id  
    }  
  }  
`;  
`;
```

```
// Department Queries and Mutations  
export const GET_DEPARTMENTS = gql`  
  query GetDepartments {  
    departments {  
      id  
      name  
    }  
  }  
`;  
`;
```

```
export const GET_DEPARTMENT = gql`  
  query GetDepartment($id: Int!) {  
    department(id: $id) {  
      id  
      name  
    }  
  }  
`;  
`;
```

```
export const ADD_DEPARTMENT = gql`  
  mutation AddDepartment($name: String!) {  
    addDepartment(name: $name) {  
      id  
      name  
    }  
  }  
`;  
`;
```

```
export const UPDATE_DEPARTMENT = gql`
```

```
mutation UpdateDepartment($id: Int!, $name: String!) {  
  updateDepartment(id: $id, name: $name) {  
    id  
    name  
  }  
}  
`;  
`;
```

```
export const DELETE_DEPARTMENT = gql`  
mutation DeleteDepartment($id: Int!) {  
  deleteDepartment(id: $id) {  
    id  
  }  
}  
`;  
`;
```

```
// Designation Queries and Mutations  
export const GET_DESIGNATIONS = gql`  
query GetDesignations {  
  designations {  
    id  
    title  
  }  
}  
`;  
`;
```

```
export const GET_DESIGNATION = gql`  
query GetDesignation($id: Int!) {  
  designation(id: $id) {  
    id  
    title  
  }  
}  
`;  
`;
```

```
export const ADD_DESIGNATION = gql`  
mutation AddDesignation($title: String!) {  
  addDesignation(title: $title) {  
    id  
    title  
  }  
}  
`;  
`;
```

```

export const UPDATE_DESIGNATION = gql`
  mutation UpdateDesignation($id: Int!, $title: String!) {
    updateDesignation(id: $id, title: $title) {
      id
      title
    }
  }
`;

```

```

export const DELETE_DESIGNATION = gql`
  mutation DeleteDesignation($id: Int!) {
    deleteDesignation(id: $id) {
      id
    }
  }
`;

```

```

const employeeService = {
  // Employee services
  getEmployees: async () => {
    const response = await client.query({ query: GET_EMPLOYEES });
    return response.data.employees;
  },
  getEmployee: async (id) => {
    const response = await client.query({ query: GET_EMPLOYEE, variables: { id } });
    return response.data.employee;
  },
  addEmployee: async (employee) => {
    const response = await client.mutate({ mutation: ADD_EMPLOYEE, variables: employee });
    return response.data.addEmployee;
  },
  updateEmployee: async (employee) => {
    const response = await client.mutate({ mutation: UPDATE_EMPLOYEE, variables: employee });
    return response.data.updateEmployee;
  },
  deleteEmployee: async (id) => {
    const response = await client.mutate({ mutation: DELETE_EMPLOYEE, variables: { id } });
    return response.data.deleteEmployee;
  },

  // Department services
  getDepartments: async () => {
    const response = await client.query({ query: GET_DEPARTMENTS });
    return response.data.departments;
  },

```

```

    },
    getDepartment: async (id) => {
      const response = await client.query({ query: GET_DEPARTMENT, variables: { id } });
      return response.data.department;
    },
    addDepartment: async (department) => {
      const response = await client.mutate({ mutation: ADD_DEPARTMENT, variables: department });
      return response.data.addDepartment;
    },
    updateDepartment: async (department) => {
      const response = await client.mutate({ mutation: UPDATE_DEPARTMENT, variables: department
});
      return response.data.updateDepartment;
    },
    deleteDepartment: async (id) => {
      const response = await client.mutate({ mutation: DELETE_DEPARTMENT, variables: { id } });
      return response.data.deleteDepartment;
    },

    // Designation services
    getDesignations: async () => {
      const response = await client.query({ query: GET_DESIGNATIONS });
      return response.data.designations;
    },
    getDesignation: async (id) => {
      const response = await client.query({ query: GET_DESIGNATION, variables: { id } });
      return response.data.designation;
    },
    addDesignation: async (designation) => {
      const response = await client.mutate({ mutation: ADD_DESIGNATION, variables: designation });
      return response.data.addDesignation;
    },
    updateDesignation: async (designation) => {
      const response = await client.mutate({ mutation: UPDATE_DESIGNATION, variables: designation
});
      return response.data.updateDesignation;
    },
    deleteDesignation: async (id) => {
      const response = await client.mutate({ mutation: DELETE_DESIGNATION, variables: { id } });
      return response.data.deleteDesignation;
    },
  };

export default employeeService;

```

2. Backend: Apollo Server with Sequelize and MySQL

1. Project Setup

1. Create a new project directory and initialize it:

```
mkdir graphql-server
```

```
cd graphql-server
```

```
npm init -y
```

2. Install necessary dependencies:

```
npm install apollo-server graphql sequelize mysql2
```

- apollo-server: For setting up the Apollo Server.
- graphql: Required for using GraphQL.
- sequelize: ORM for managing MySQL database.
- mysql2: MySQL client for Node.js.

2. Sequelize Configuration

1. Create config directory and config.json for Sequelize configuration:

```
mkdir config
```

```
touch config/config.json
```

2. Add the following content to config/config.json:

```
{
  "development": {
    "username": "root",
    "password": "root1234",
    "database": "employeeedb",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": "root1234",
    "database": "employeeedb",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": "root1234",
    "database": "employeeedb",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

```
}  
}
```

3. Define Sequelize Models

1. Create models directory and define models:

○ index.js:

```
// models/index.js  
const Sequelize = require('sequelize');  
const config = require('../config/config.json');  
  
const env = process.env.NODE_ENV || 'development';  
const dbConfig = config[env];  
const sequelize = new Sequelize(dbConfig.database, dbConfig.username, dbConfig.password,  
dbConfig);  
  
const db = {};  
  
db.Sequelize = Sequelize;  
db.sequelize = sequelize;  
  
db.Department = require('./department')(sequelize, Sequelize);  
db.Designation = require('./designation')(sequelize, Sequelize);  
db.Employee = require('./employee')(sequelize, Sequelize);  
  
// Define associations  
db.Department.hasMany(db.Employee, { foreignKey: 'department_id' });  
db.Employee.belongsTo(db.Department, { foreignKey: 'department_id' });  
  
db.Designation.hasMany(db.Employee, { foreignKey: 'designation_id' });  
db.Employee.belongsTo(db.Designation, { foreignKey: 'designation_id' });  
  
db.Employee.belongsTo(db.Employee, { as: 'Manager', foreignKey: 'manager_id' });  
db.Employee.hasMany(db.Employee, { as: 'Subordinates', foreignKey: 'manager_id' });  
  
module.exports = db;  
○ department.js:  
// models/department.js  
module.exports = (sequelize, DataTypes) => {  
  const Department = sequelize.define('Department', {  
    id: {  
      type: DataTypes.INTEGER,  
      autoIncrement: true,  
      primaryKey: true,  
    },  
    name: {
```

```

    type: DataTypes.STRING,
    allowNull: false,
  },
});
return Department;
};

```

- **designation.js:**

javascript

Copy code

```

// models/designation.js
module.exports = (sequelize, DataTypes) => {
  const Designation = sequelize.define('Designation', {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false,
    },
  });
  return Designation;
};

```

- **employee.js:**

```

// models/employee.js
module.exports = (sequelize, DataTypes) => {
  const Employee = sequelize.define('Employee', {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true,
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    designation_id: {
      type: DataTypes.INTEGER,
      references: {

```

```

        model: 'Designations',
        key: 'id',
      },
    },
    department_id: {
      type: DataTypes.INTEGER,
      references: {
        model: 'Departments',
        key: 'id',
      },
    },
    manager_id: {
      type: DataTypes.INTEGER,
      references: {
        model: 'Employees',
        key: 'id',
      },
    },
  });
  return Employee;
};

```

4. Define GraphQL Schema and Resolvers

1. Create graphql directory and define schema and resolvers:

o typeDefs.js:

```

// graphql/typeDefs.js
const { gql } = require('apollo-server');

const typeDefs = gql`
  type Department {
    id: Int!
    name: String!
  }

  type Designation {
    id: Int!
    name: String!
  }

  type Employee {
    id: Int!
    name: String!
    email: String!
    designation: Designation
    department: Department
  }
`

```



```
  manager: Employee
  subordinates: [Employee]
}
```

```
type Query {
  departments: [Department]
  department(id: Int!): Department
  designations: [Designation]
  designation(id: Int!): Designation
  employees: [Employee]
  employee(id: Int!): Employee
}
```

```
type Mutation {
  addDepartment(name: String!): Department
  updateDepartment(id: Int!, name: String!): Department
  deleteDepartment(id: Int!): Department
  addDesignation(name: String!): Designation
  updateDesignation(id: Int!, name: String!): Designation
  deleteDesignation(id: Int!): Designation
  addEmployee(name: String!, email: String!, designationId: Int!, departmentId: Int!,
managerId: Int): Employee
  updateEmployee(id: Int!, name: String!, email: String!, designationId: Int!, departmentId: Int!,
managerId: Int): Employee
  deleteEmployee(id: Int!): Employee
}
`;
```

```
module.exports = typeDefs;
```

```
  ○ resolvers.js:
```

```
// graphql/resolvers.js
```

```
const { Department, Designation, Employee } = require('../models');
```

```
const resolvers = {
```

```
  Query: {
    departments: async () => await Department.findAll(),
    department: async (_, { id }) => await Department.findByPk(id),
    designations: async () => await Designation.findAll(),
    designation: async (_, { id }) => await Designation.findByPk(id),
    employees: async () => await Employee.findAll({
      include: [
        { model: Designation },
        { model: Department },
        { model: Employee, as: 'Manager' },
      ],
    })
  },
```

```

    { model: Employee, as: 'Subordinates' }
  ]
}),
employee: async (_, { id }) => await Employee.findByPk(id, {
  include: [
    { model: Designation },
    { model: Department },
    { model: Employee, as: 'Manager' },
    { model: Employee, as: 'Subordinates' }
  ]
})
},
Mutation: {
  addDepartment: async (_, { name }) => await Department.create({ name }),
  updateDepartment: async (_, { id, name }) => {
    const department = await Department.findByPk(id);
    department.name = name;
    await department.save();
    return department;
  },
  deleteDepartment: async (_, { id }) => {
    const department = await Department.findByPk(id);
    await department.destroy();
    return department;
  },
  addDesignation: async (_, { name }) => await Designation.create({ name }),
  updateDesignation: async (_, { id, name }) => {
    const designation = await Designation.findByPk(id);
    designation.name = name;
    await designation.save();
    return designation;
  },
  deleteDesignation: async (_, { id }) => {
    const designation = await Designation.findByPk(id);
    await designation.destroy();
    return designation;
  },
  addEmployee: async (_, { name, email, designationId, departmentId, managerId }) => await
Employee.create({
  name,
  email,
  designation_id: designationId,
  department_id: departmentId,
  manager_id: managerId

```

```

    }},
    updateEmployee: async (_, { id, name, email, designationId, departmentId, managerId }) => {
      const employee = await Employee.findByPk(id);
      employee.name = name;
      employee.email = email;
      employee.designation_id = designationId;
      employee.department_id = departmentId;
      employee.manager_id = managerId;
      await employee.save();
      return employee;
    },
    deleteEmployee: async (_, { id }) => {
      const employee = await Employee.findByPk(id);
      await employee.destroy();
      return employee;
    }
  },
  Employee: {
    designation: async (employee) => await Designation.findByPk(employee.designation_id),
    department: async (employee) => await Department.findByPk(employee.department_id),
    manager: async (employee) => await Employee.findByPk(employee.manager_id),
    subordinates: async (employee) => await Employee.findAll({ where: { manager_id:
employee.id } })
  }
};

```

module.exports = resolvers;

5. Set Up Apollo Server

1. Create server.js:

```

// server.js
const { ApolloServer } = require('apollo-server');
const typeDefs = require('./graphql/typeDefs');
const resolvers = require('./graphql/resolvers');
const db = require('./models');

const server = new ApolloServer({ typeDefs, resolvers });

db.sequelize.sync().then(() => {
  server.listen().then(({ url }) => {
    console.log(`🚀 Server ready at ${url}`);
  });
});

```

- **ApolloServer Setup:**

- typeDefs and resolvers are imported and passed to the ApolloServer constructor.
- db.sequelize.sync() ensures the database schema is up-to-date before starting the server.
- The server listens on the default port and logs the URL.

6. Initialize the Database

1. Start your MySQL server and create the employeedb database:

```
create database employeedb;
use employeedb;
-- Create Departments table
CREATE TABLE departments (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Create Designations table
CREATE TABLE designations (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Create Employees table
CREATE TABLE employees (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  designation_id INT,
  department_id INT,
  manager_id INT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (designation_id) REFERENCES designations(id),
  FOREIGN KEY (department_id) REFERENCES departments(id),
  FOREIGN KEY (manager_id) REFERENCES employees(id)
);

-- Insert sample departments
INSERT INTO departments (name) VALUES ('HR'), ('Engineering'), ('Sales');
```

-- Insert sample designations

```
INSERT INTO designations (title) VALUES ('Manager'), ('Developer'), ('Salesperson');
```

-- Insert sample employees

```
INSERT INTO employees(name, email, designation_id, department_id, manager_id, createdAt, updatedAt) VALUES
```

```
('Alice Johnson', 'alice@example.com', 1, 1, NULL, NOW(), NOW()),
```

```
('Bob Smith', 'bob@example.com', 2, 2, 1, NOW(), NOW()),
```

```
('Charlie Brown', 'charlie@example.com', 3, 3, 1, NOW(), NOW()),
```

```
('Diana Prince', 'diana@example.com', 2, 2, 1, NOW(), NOW()),
```

```
('Eve Adams', 'eve@example.com', 3, 3, 1, NOW(), NOW());
```

7. Run the Apollo Server

1. Start your Apollo Server:

node server.js

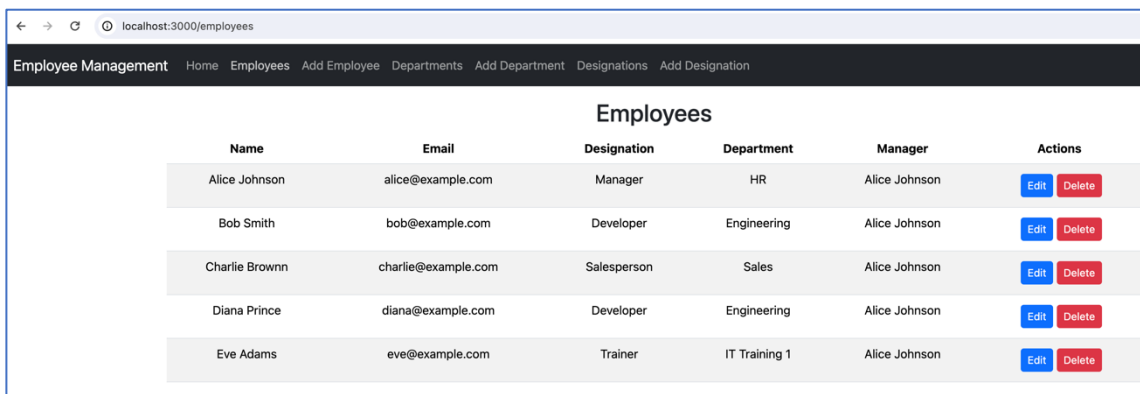
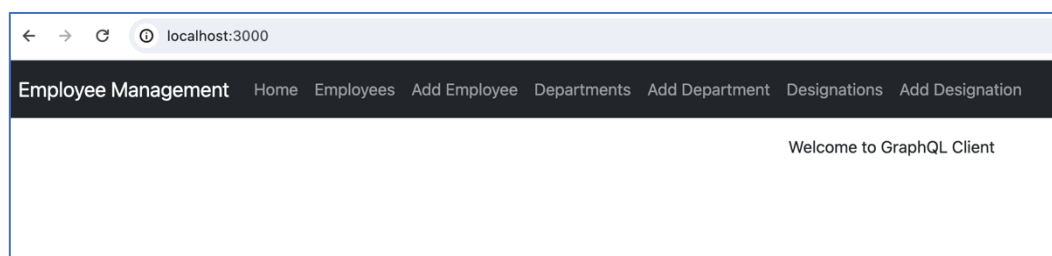
- You should see a message indicating that the server is running:

 Server ready at <http://localhost:4000/>

Final Thoughts

With these steps, you have set up a fully functional employee management system with a React frontend and an Apollo Server backend using Sequelize and MySQL. The frontend includes components for managing employees, departments, and designations, and the backend provides the necessary GraphQL endpoints to support these operations.

UI Final screens

A screenshot of the 'Employees' page in the application. The address bar shows 'localhost:3000/employees'. The navigation bar is the same as the previous screenshot. The page title is 'Employees'. Below the title is a table with columns: Name, Email, Designation, Department, Manager, and Actions. The table contains five rows of employee data.

Name	Email	Designation	Department	Manager	Actions
Alice Johnson	alice@example.com	Manager	HR	Alice Johnson	Edit Delete
Bob Smith	bob@example.com	Developer	Engineering	Alice Johnson	Edit Delete
Charlie Brown	charlie@example.com	Salesperson	Sales	Alice Johnson	Edit Delete
Diana Prince	diana@example.com	Developer	Engineering	Alice Johnson	Edit Delete
Eve Adams	eve@example.com	Trainer	IT Training 1	Alice Johnson	Edit Delete

← → ↻ 🌐 localhost:3000/add-employee

Employee Management Home Employees Add Employee Departments Add Department Designations Add Designation

Add Employee

Name:

Email:

Designation:

Manager

Department:

HR

Manager ID:

Add Employee

← → ↻ 🌐 localhost:3000/departments

Employee Management Home Employees Add Employee Departments Add Department Designations Add Designation

Departments

Name	Actions
HR	<div>EditDelete</div>
Engineering	<div>EditDelete</div>
Sales	<div>EditDelete</div>
IT Training 1	<div>EditDelete</div>
Accounting	<div>EditDelete</div>

← → ↻ 🌐 localhost:3000/add-department

Employee Management Home Employees Add Employee Departments Add Department Designations Add Designation

Add Department

Name:

Add Department

← → ↻ 🌐 localhost:3000/designations

Employee Management Home Employees Add Employee Departments Add Department Designations Add Designation

Designations

Title	Actions
Manager	<div>EditDelete</div>
Developer	<div>EditDelete</div>
Salesperson	<div>EditDelete</div>
Trainer	<div>EditDelete</div>

← → ↻ 🌐 localhost:3000/add-designation

Employee Management Home Employees Add Employee Departments Add Department Designations Add Designation

Add Designation

Title:

Add Designation