# **Implementing Mutations in Apollo Client**

**Mutations** in GraphQL are used to modify data on the server-side. They are similar to HTTP POST, PUT, DELETE methods in RESTful APIs. Mutations allow you to create, update, or delete data, and they can also return the modified data, allowing your client to update its local state immediately after the mutation is completed.

**Why Mutations are Used**

1. **Data Modification**: Mutations are used to change data on the server, such as creating new records, updating existing ones, or deleting records.
2. **Optimistic UI Updates**: Mutations allow for optimistic updates where the UI is updated immediately, anticipating the server's response.
3. **Immediate Feedback**: They provide immediate feedback to the user by returning the modified data, which can be used to update the UI.
4. **Batch Operations**: Mutations can perform multiple operations in a single request, which is more efficient than making multiple API calls.

**Where to Use Mutations**

1. **Form Submissions**: When a user submits a form to create or update data.
2. **Delete Actions**: When a user performs an action to delete a record.
3. **Complex Transactions**: When multiple related operations need to be performed in a single transaction.
4. **Real-Time Applications**: To push updates to the server that can be broadcasted to other clients in real-time.

**Scenario: Managing Employees in an Application**

Let's consider an example where you manage employee records in an application. You'll need to perform the following operations:

1. Add a new employee.
2. Update an existing employee.
3. Delete an employee.

**Detailed Explanation and Implementation**

**Step 1: Define the Mutations**

Create a file for your GraphQL mutations.

// src/mutations/employeeMutations.js

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

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

}

}

`;

**Step 2: Use Mutations in Components**

**Adding an Employee**

1. **AddEmployeeForm.js**

// src/components/AddEmployeeForm.js

import React from 'react';

import { useForm } from 'react-hook-form';

import { useMutation } from '@apollo/client';

import { ADD\_EMPLOYEE } from '../mutations/employeeMutations';

import { GET\_EMPLOYEES } from '../queries/employeeQueries';

const AddEmployeeForm = () => {

const { register, handleSubmit, reset } = useForm();

const [addEmployee, { loading, error }] = useMutation(ADD\_EMPLOYEE, {

update(cache, { data: { addEmployee } }) {

const { employees } = cache.readQuery({ query: GET\_EMPLOYEES });

cache.writeQuery({

query: GET\_EMPLOYEES,

data: { employees: [...employees, addEmployee] },

});

},

onError: (error) => {

console.error('Error adding employee:', error.message);

}

});

const onSubmit = async (formData) => {

try {

await addEmployee({ variables: { ...formData, designationId: parseInt(formData.designationId), departmentId: parseInt(formData.departmentId), managerId: formData.managerId ? parseInt(formData.managerId) : null } });

reset(); // Reset the form after successful submission

} catch (error) {

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

}

};

return (

<div>

<h2>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" type="email" {...register('email', { required: true })} />

</div>

<div className="form-group">

<label>Designation ID:</label>

<input className="form-control" type="number" {...register('designationId', { required: true })} />

</div>

<div className="form-group">

<label>Department ID:</label>

<input className="form-control" type="number" {...register('departmentId', { required: true })} />

</div>

<div className="form-group">

<label>Manager ID:</label>

<input className="form-control" type="number" {...register('managerId')} />

</div>

<button type="submit" className="btn btn-primary mt-3" disabled={loading}>

{loading ? 'Adding...' : 'Add Employee'}

</button>

{error && <p>Error adding employee: {error.message}</p>}

</form>

</div>

);

};

export default AddEmployeeForm;

**Updating an Employee**

1. **UpdateEmployeeForm.js**

// src/components/UpdateEmployeeForm.js

import React, { useEffect } from 'react';

import { useForm } from 'react-hook-form';

import { useMutation, useQuery } from '@apollo/client';

import { GET\_EMPLOYEE } from '../queries/employeeQueries';

import { UPDATE\_EMPLOYEE } from '../mutations/employeeMutations';

const UpdateEmployeeForm = ({ id }) => {

const { register, handleSubmit, setValue } = useForm();

const { loading: queryLoading, error: queryError, data } = useQuery(GET\_EMPLOYEE, {

variables: { id },

});

const [updateEmployee, { loading: mutationLoading, error: mutationError }] = useMutation(UPDATE\_EMPLOYEE, {

onCompleted: () => {

alert('Employee updated successfully');

},

onError: (error) => {

console.error('Error updating employee:', error.message);

}

});

useEffect(() => {

if (data) {

const { employee } = data;

setValue('name', employee.name);

setValue('email', employee.email);

setValue('designationId', employee.designation.id);

setValue('departmentId', employee.department.id);

setValue('managerId', employee.manager ? employee.manager.id : '');

}

}, [data, setValue]);

const onSubmit = async (formData) => {

try {

await updateEmployee({ variables: { id, ...formData, designationId: parseInt(formData.designationId), departmentId: parseInt(formData.departmentId), managerId: formData.managerId ? parseInt(formData.managerId) : null } });

} catch (error) {

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

}

};

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

if (queryError) return <p>Error loading employee: {queryError.message}</p>;

return (

<div>

<h2>Update 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" type="email" {...register('email', { required: true })} />

</div>

<div className="form-group">

<label>Designation ID:</label>

<input className="form-control" type="number" {...register('designationId', { required: true })} />

</div>

<div className="form-group">

<label>Department ID:</label>

<input className="form-control" type="number" {...register('departmentId', { required: true })} />

</div>

<div className="form-group">

<label>Manager ID:</label>

<input className="form-control" type="number" {...register('managerId')} />

</div>

<button type="submit" className="btn btn-primary mt-3" disabled={mutationLoading}>

{mutationLoading ? 'Updating...' : 'Update Employee'}

</button>

{mutationError && <p>Error updating employee: {mutationError.message}</p>}

</form>

</div>

);

};

export default UpdateEmployeeForm;

**Deleting an Employee**

1. **EmployeeList.js** (with delete functionality)

// src/components/EmployeeList.js

import React from 'react';

import { useQuery, useMutation } from '@apollo/client';

import { GET\_EMPLOYEES } from '../queries/employeeQueries';

import { DELETE\_EMPLOYEE } from '../mutations/employeeMutations';

const EmployeeList = () => {

const { loading, error, data } = useQuery(GET\_EMPLOYEES);

const [deleteEmployee] = useMutation(DELETE\_EMPLOYEE, {

update(cache, { data: { deleteEmployee } }) {

const { employees } = cache.readQuery({ query: GET\_EMPLOYEES });

cache.writeQuery({

query: GET\_EMPLOYEES,

data: { employees: employees.filter(emp => emp.id !== deleteEmployee.id) },

});

},

onError: (error) => {

console.error('Error deleting employee:', error.message);

}

});

const handleDelete = async (id) => {

try {

await deleteEmployee({ variables: { id } });

} catch (error) {

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

}

};

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

if (error) return <p>Error: {error.message}</p>;

return (

<div>

<h2>Employees</h2>

<ul>

{data.employees.map(employee => (

<li key={employee.id}>

{employee.name} - {employee.email}

<button onClick={() => handleDelete(employee.id)}>Delete</button>

</li>

))}

</ul>

</div>

);

};

export default EmployeeList;

**Explanation and Detailed Steps**

1. **Defining the Mutations**:
   * **ADD\_EMPLOYEE**: This mutation creates a new employee record with the provided details.
   * **UPDATE\_EMPLOYEE**: This mutation updates an existing employee's details based on the provided ID.
   * **DELETE\_EMPLOYEE**: This mutation deletes an employee record based on the provided ID.
2. **Using Mutations in Components**:
   * **AddEmployeeForm**: Uses the useMutation hook to call the ADD\_EMPLOYEE mutation when the form is submitted. The cache is updated to include the new employee.
   * **UpdateEmployeeForm**: Uses useMutation for the UPDATE\_EMPLOYEE mutation. The form is pre-filled with the existing data using the useQuery hook.
   * **EmployeeList**: Fetches the list of employees using useQuery and allows deleting an employee with the DELETE\_EMPLOYEE mutation. The cache is updated to remove the deleted employee.
3. **Optimistic Updates**:
   * Optimistic updates can be added to mutations to update the UI immediately before the server response is received. This provides a better user experience.
4. **Error Handling**:
   * Proper error handling is implemented using the onError callback in the useMutation hook to catch and display errors.

**Conclusion**

Mutations in Apollo Client are essential for modifying data on the server. They provide a robust way to handle form submissions, delete actions, and complex transactions. By integrating mutations with caching mechanisms, you can ensure your application's UI stays in sync with the server, providing a seamless and responsive user experience. This detailed implementation shows how to manage employee records in an application, highlighting the use of mutations for adding, updating, and deleting records.