

**Setup**

**1. Install React Query**

* Open your terminal in the project directory.
* Run the following command to install React Query:

bash

CopyEdit

npm install @tanstack/react-query@4

**2. Create and Configure the Query Client**

* Open your main application file (e.g., App.jsx).
* Import the necessary modules from React Query:

jsx

CopyEdit

import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

* Create a new Query Client instance and set default options (optional):

jsx

CopyEdit

const queryClient = new QueryClient({

defaultOptions: {

queries: {

staleTime: 60 \* 1000, // Cache data stays fresh for 1 minute

},

},

});

**3. Wrap the Application with the QueryClientProvider**

* Wrap your application's component tree with QueryClientProvider and pass the queryClient instance:

jsx

CopyEdit

function App() {

return (

<QueryClientProvider client={queryClient}>

{/\* Your application components go here \*/}

</QueryClientProvider>

);

}

**4. Install and Set Up React Query Dev Tools (Optional)**

* Install the React Query Dev Tools package:

bash

CopyEdit

npm install @tanstack/react-query-devtools

* Import and include the Dev Tools component in your application:

jsx

CopyEdit

import { ReactQueryDevtools } from '@tanstack/react-query-devtools';

function App() {

return (

<QueryClientProvider client={queryClient}>

{/\* Your application components \*/}

<ReactQueryDevtools initialIsOpen={false} />

</QueryClientProvider>

);

}

**5. Verify the Setup**

* Start your application and check for the React Query Dev Tools symbol (typically in the corner of the browser window).
* If you click the symbol, the Dev Tools panel will open. Initially, it will be empty until data is fetched.

**Fetching Using React Query**

1. **Fetching Data with React Query**
   * Use useQuery to fetch data.
   * Example:

javascript

CopyEdit

const { data: cabins, isLoading, isError, error } = useQuery('cabins', getCabins);

* + Handle states:
    - isLoading: Show a spinner during loading.
    - isError: Display an error message.

1. **Custom Hook for Abstraction**
   * Create a custom hook to reuse query logic:

javascript

CopyEdit

export function useCabins() {

const { isLoading, data: cabins, error } = useQuery({

queryKey: ["cabins"],

queryFn: getCabins,

});

return { isLoading, cabins, error };

}

1. **Fresh vs. Stale Data**
   * **Fresh Data**: Served directly from the cache if within the staleTime.
   * **Stale Data**: Refetched automatically when the component mounts after staleTime expires.
2. **Default Behavior**
   * staleTime: Default is 0, meaning data is immediately stale.
   * React Query refetches data:
     + On component mount.
     + When the query becomes active.

 **Create the delete function:** Define an async function deleteCabin that accepts the id of the cabin to delete.

js

CopyEdit

const deleteCabin = async (id) => {

await supabase.from('cabins').delete().eq('id', id);

};

 **Set up the mutation:** Use the useMutation hook from React Query to call deleteCabin on a successful deletion.

js

CopyEdit

const { mutate, isLoading: isDeleting } = useMutation(deleteCabin, {

onSuccess: () => {

queryClient.invalidateQueries('cabins');

},

onError: (error) => {

alert(`Error: ${error.message}`);

}

});

 **Add the Delete button:** Bind the mutate function to the onClick of the Delete button, passing the cabinId.

js

CopyEdit

<button onClick={() => mutate(cabinId)} disabled={isDeleting}>

{isDeleting ? 'Deleting...' : 'Delete'}

</button>

 **Invalidate the cache on success:** Use queryClient.invalidateQueries('cabins') to re-fetch the data and reflect the change immediately in the UI.

 **Handle errors:** In the onError callback, handle any errors by showing an alert or other notification.

Here’s a shorter version for displaying a toast notification in your app:

1. **Install React Hot Toast**:  
   Use the react-hot-toast library for easy toast notifications.
2. **Set up Toaster component**:  
   Add the Toaster component to your app, which is responsible for displaying the toast.
3. **Configure Toast Styles**:  
   Customize the toast appearance by setting options like position, gutter (space), container style, and toast duration (e.g., 3 seconds for success, 5 seconds for errors).
4. **Add Toast Notifications**:  
   Import the toast function and use toast.success or toast.error for success or error messages, respectively. Example:
5. toast.success('Operation successful!', { duration: 3000 });
6. toast.error('Something went wrong!', { duration: 5000 });
7. **Replace Alerts**:  
   Replace any alerts in your app with toast.success or toast.error to improve the user experience with styled notifications.

This process ensures your app has smooth and customizable toast notifications.

**Steps to Set Up React Hook Form:**

1. **Install React Hook Form:** Install the library using npm:

bash

CopyEdit

npm install react-hook-form

1. **Set Up Form Component:**
   * Add a button to toggle the form's visibility.
   * Use useState to manage form visibility.

jsx

CopyEdit

const [showForm, setShowForm] = useState(false);

1. **Create Form with React Hook Form:**
   * Use the useForm hook to initialize the form and get methods like register, handleSubmit, and formState.
   * Register each form field using register, which connects the input to RHF.

jsx

CopyEdit

const { register, handleSubmit, formState: { errors } } = useForm();

1. **Handle Form Submission:**
   * Use handleSubmit to handle form submission and pass a custom function to process the form data.
   * Inside the onSubmit function, log or send data to the backend (e.g., Supabase).

jsx

CopyEdit

const onSubmit = (data) => {

console.log(data);

};

1. **Validation:**
   * Add validation rules inside the register method (e.g., required, pattern).
   * Display error messages based on validation failures.

jsx

CopyEdit

{...register('fieldName', { required: 'This field is required' })}

1. **Reset Form (Optional):**
   * Use reset() to reset form fields if needed (e.g., after submission or cancel action).

jsx

CopyEdit

const { reset } = useForm();

steps to create a cabin, based on the transcript you provided:

**1. Create a createCabin Function in Services:**

* Define a function createCabin that accepts a new cabin object as an input.
* This function interacts with the Supabase API to create a new cabin in the database.

js

CopyEdit

const createCabin = async (newCabin) => {

try {

const { data, error } = await supabase

.from('cabins')

.insert([newCabin]);

if (error) throw error;

return data;

} catch (error) {

console.error("Error creating cabin:", error);

}

};

**2. Set Up Role-Level Security (RLS):**

* Create a policy for inserting cabins. Enable the policy by selecting "Insert" and setting the expression to true to allow all users to create new cabins.
* Similarly, enable the update policy by selecting "Update" and setting the expression to true.

**3. Grab Code from Supabase API Docs:**

* Go to Supabase API docs and grab the code to insert new data into the cabins table.
* The code inserts the new cabin object into the table by matching the field names with the table's columns.

js

CopyEdit

const { data, error } = await supabase

.from('cabins')

.insert([newCabin]);

**4. Use React Query Mutation for Creating Cabin:**

* Use useMutation from React Query to handle the cabin creation process.
* Pass the createCabin function to useMutation to mutate the cabin data.
* Add an onSuccess handler to show a success toast and invalidate the cabins query to update the UI.

jsx

CopyEdit

const { mutate, isLoading } = useMutation(createCabin, {

onSuccess: () => {

toast.success("New cabin successfully created!");

queryClient.invalidateQueries('cabins');

},

onError: (error) => {

toast.error(error.message);

},

});

**5. Handle Form Submission:**

* Use mutate to pass form data into the createCabin function.
* After submission, reset the form and clear the fields using the reset function from React Hook Form.

jsx

CopyEdit

const { reset } = useForm();

const onSubmit = (data) => {

mutate(data);

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

};

**6. Disable Button During Submission:**

* Use the isLoading state from the mutation to disable the submit button during the cabin creation process.

jsx

CopyEdit

<button disabled={isLoading}>

{isLoading ? 'Creating...' : 'Create Cabin'}

</button>