React Query

Hooks for managing, caching and syncing asynchronous and remote data in React

Day 2 - Fundamentals of Frontend Development

Setup

```
npx create-react-app react-query-crud
npm i json-server
```

Create `data/db.json`

```
npx json-server data/db.json --port 3500
curl http://localhost:3500/todos
```

Setup

axios

```
import axios from "axios";

// CRUD methods with axios
const todosApi = axios.create({
  baseURL: "http://localhost:3500",
});

export const getTodos = async () => {
  const response = await todosApi.get("/todos");
  return response.data;
};


....
```

Without React Query

TodoList.jsx: stale

```
// fetch on first call
useEffect(() => {
  getTodos().then((response) => {
    console.log("🚀 ~ file: TodoList.jsx:15 ~ useEffect ~ output", response);
    setCurrentTodos(response);
  });
}, []);
 // This does NOT trigger GET
 const handleSubmit = (e) => {
  e.preventDefault();
  if (newTodo) {
    addTodo({ userId: 1, title: newTodo, completed: false });

    setNewTodo("");

 .... Need to manually GET again here....
  } else {
    console.log("No todo found");
  }
};
```

What for?

Efficient **query caching** client

Assume `cacheTime = 5 minutes, staleTime = 0`

1. A **new** instance of `useQuery('todos', fetchTodos)` mounts
    - *Goes stale immediately*
2. A **second** instance of `useQuery('todos', fetchTodos)` mounts elsewhere
    - *immediate from cache*
3. A background **refetch** is triggered for both queries (but only one request), since a new instance appeared on screen.
    - *because staleTime is 0*

What for?

Efficient query caching client

1. Both instances of the `useQuery('todos', fetchTodos)` query are **unmounted** and no longer in use.
   - *cache counts down from now*
2. Before the cache timeout has completed **another** instance of `useQuery('todos', fetchTodos)` mounts.
   - *immediate return from cache, background fetch dispatched*
3. The **final** instance of `useQuery('todos', fetchTodos)` unmounts.
4. **If no more** instances of `useQuery('todos', fetchTodos)` appear within 5 minutes.
   - *query + data are garbage collected*

useQueryClient

index.js: declare provider

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";

import { QueryClient, QueryClientProvider } from "react-query";
// only available to us in development mode
import { ReactQueryDevtools } from "react-query/devtools";

const queryClient = new QueryClient();
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <QueryClientProvider client={queryClient}>
      <App />
      <ReactQueryDevtools initialIsOpen />
    </QueryClientProvider>
  </React.StrictMode>
);
```

useQueryClient

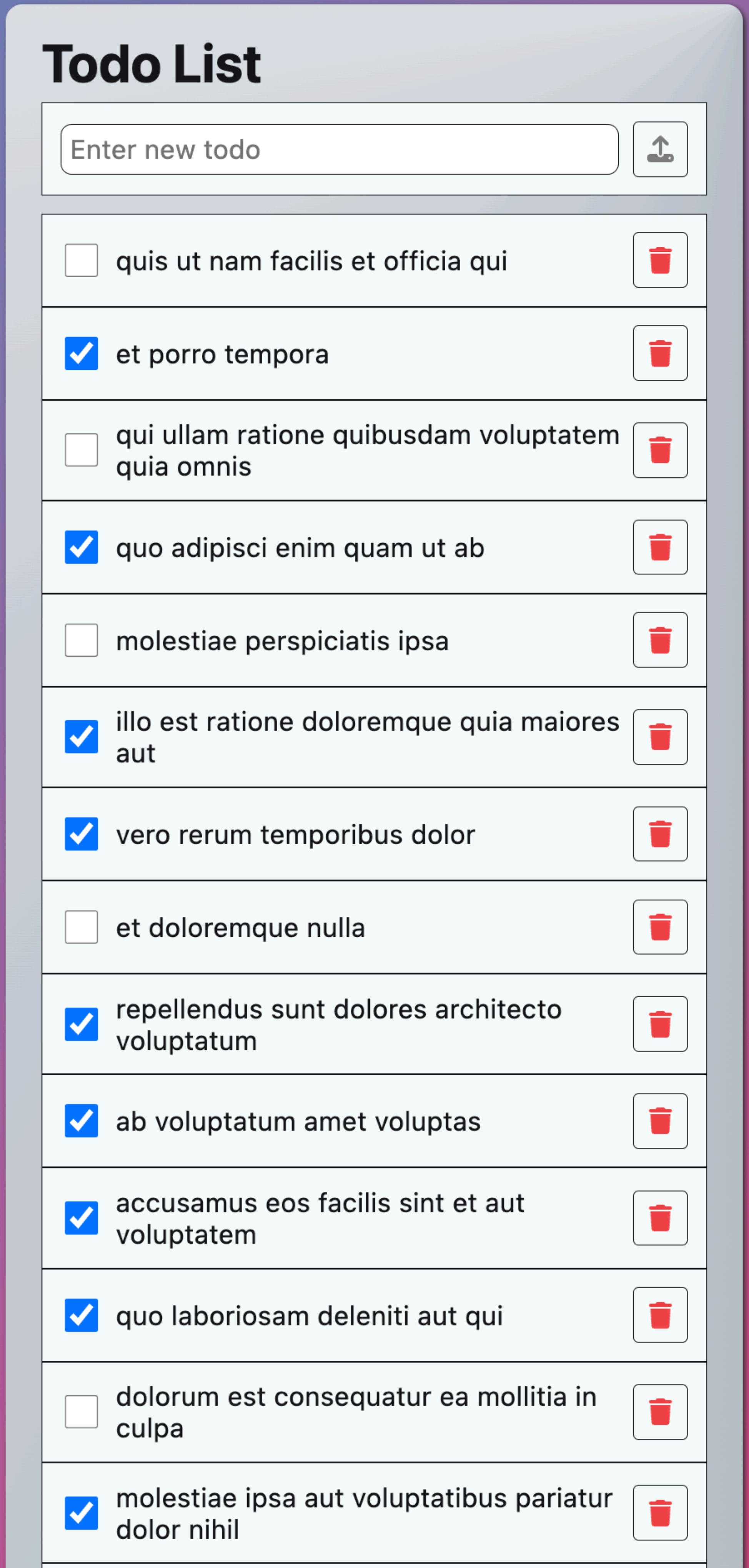
```
const {
  // destructure
  isLoading,
  isError,
  error,
  data: todos, // rename data as todos
} = useQuery("todos", getTodos); // cache name is todos

const addTodoMutation = useMutation(addTodo, {
  onSuccess: () => {
    // Invalidates the todos cache and then triggers a refetch
    queryClient.invalidateQueries("todos");
  },
});


 const handleSubmit = (e) => {
   e.preventDefault(); // dont want the form to reload the page
   addTodoMutation.mutate({ userId: 1, title: newTodo, completed: false });
   setNewTodo("");
 };
```

Demo

new todo added below



Todo List

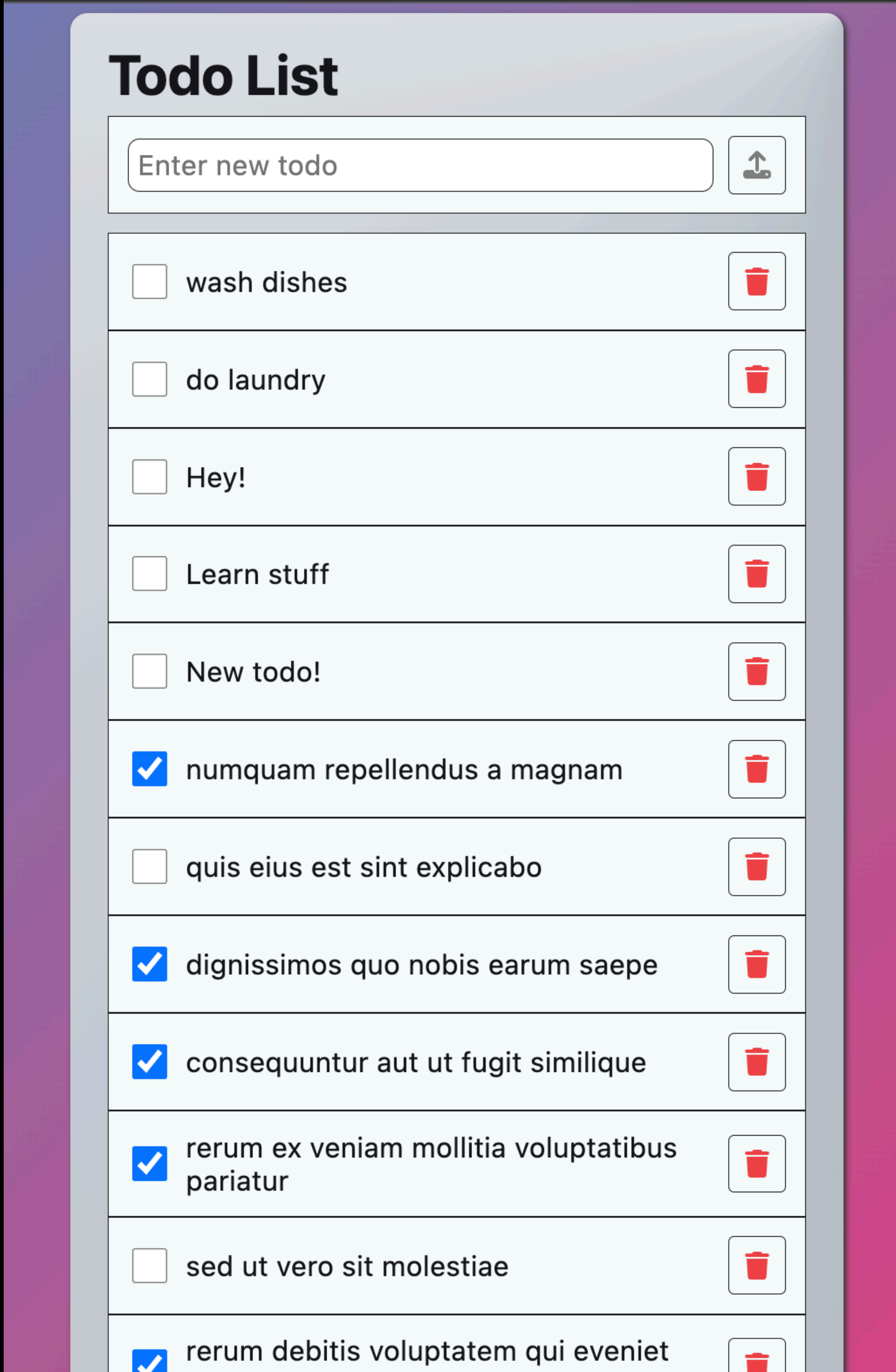
Enter new todo

- [ ] quis ut nam facilis et officia qui
- [x] et porro tempora
- [ ] qui ullam ratione quibusdam voluptatem quia omnis
- [x] quo adipisci enim quam ut ab
- [ ] molestiae perspiciatis ipsa
- [x] illo est ratione doloremque quia maiores aut
- [x] vero rerum temporibus dolor
- [ ] et doloremque nulla
- [x] repellendus sunt dolores architecto voluptatum
- [x] ab voluptatum amet voluptas
- [x] accusamus eos facilis sint et aut voluptatem
- [x] quo laboriosam deleniti aut qui
- [ ] dolorum est consequatur ea mollitia in culpa
- [x] molestiae ipsa aut voluptatibus pariatur dolor nihil

Transform data

Reverse sort after fetch

```
  const {
    data: todos,
  } = useQuery("todos", getTodos, {
    select: (data) => data.sort((a,
b) => b.id - a.id),
    retry: 2,
    onError: (error) => {
      console.log(`Something went
wrong: ${error.message}`);
    },
  });
```



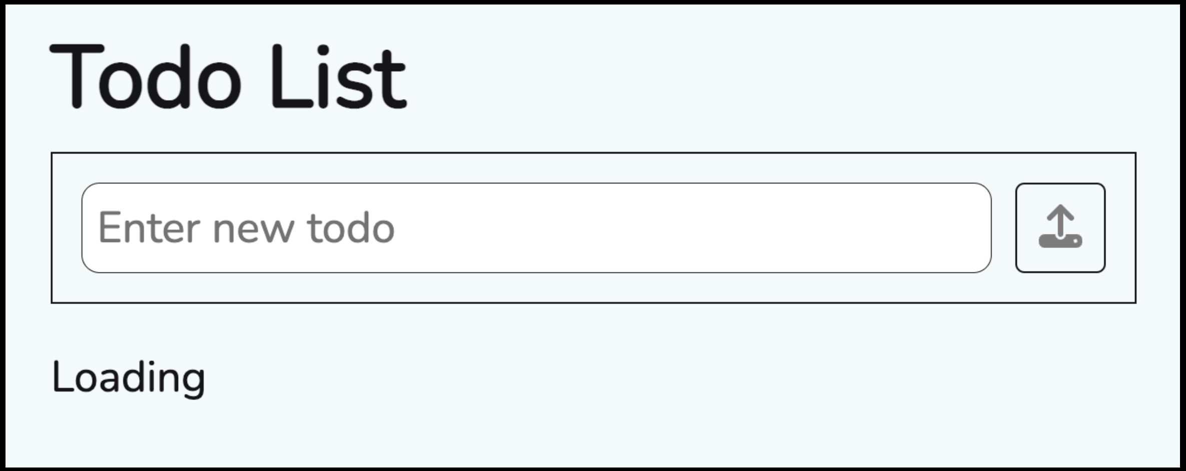
Todo List

Enter new todo

- [ ] wash dishes
- [ ] do laundry
- [ ] Hey!
- [ ] Learn stuff
- [ ] New todo!
- [x] numquam repellendus a magnam
- [ ] quis eius est sint explicabo
- [x] dignissimos quo nobis earum saepe
- [x] consequuntur aut ut fugit similique
- [x] rerum ex veniam mollitia voluptatibus pariatur
- [ ] sed ut vero sit molestiae
- [x] rerum debitis voluptatem qui eveniet

isLoading & isError

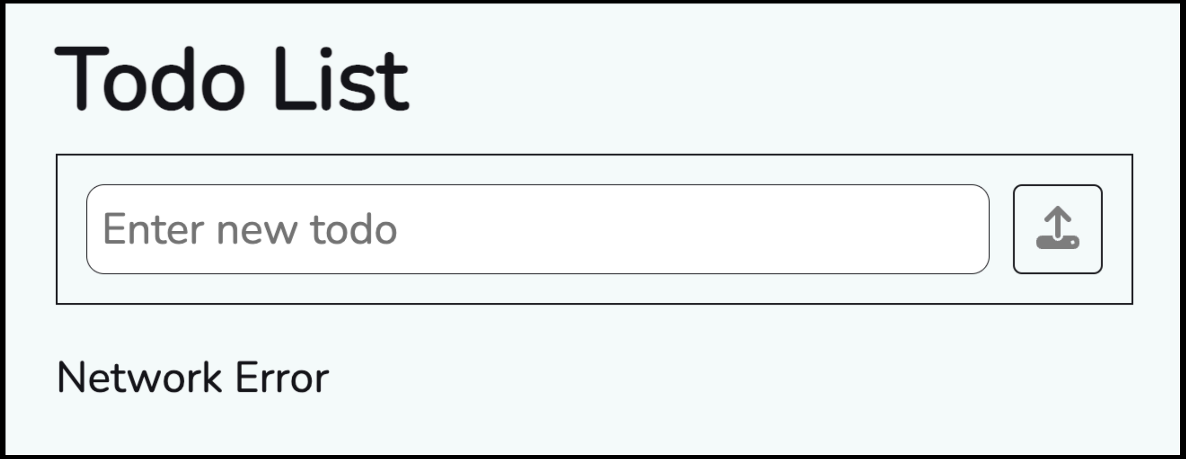
conditional rendering



Todo List

Enter new todo

Loading



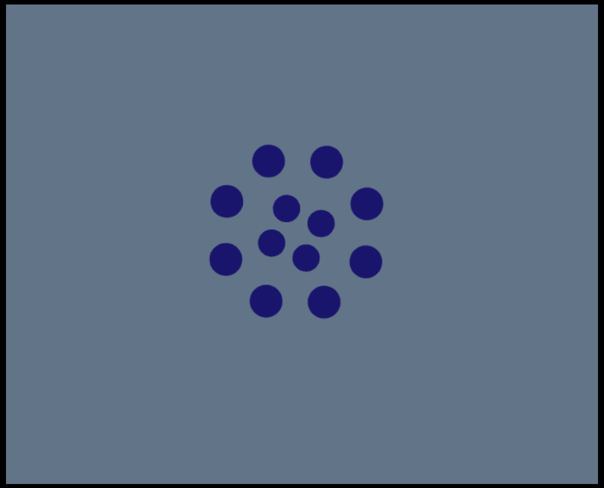
Todo List

Enter new todo

Network Error

Suspense

Display a fallback until its children have finished loading



```
const Loading = () => {
  return (
    <Circles
      height="100"
      width="100"
      color="midnightblue"
      ariaLabel="circles-loading"
      wrapperClass=""
      visible={true}
      wrapperStyle={{ position: "fixed", top: "50%", left: "50%" }}
    />
  );
};


<Suspense fallback={<Loading />}>
  <TodoList />
</Suspense>
```

Suspense

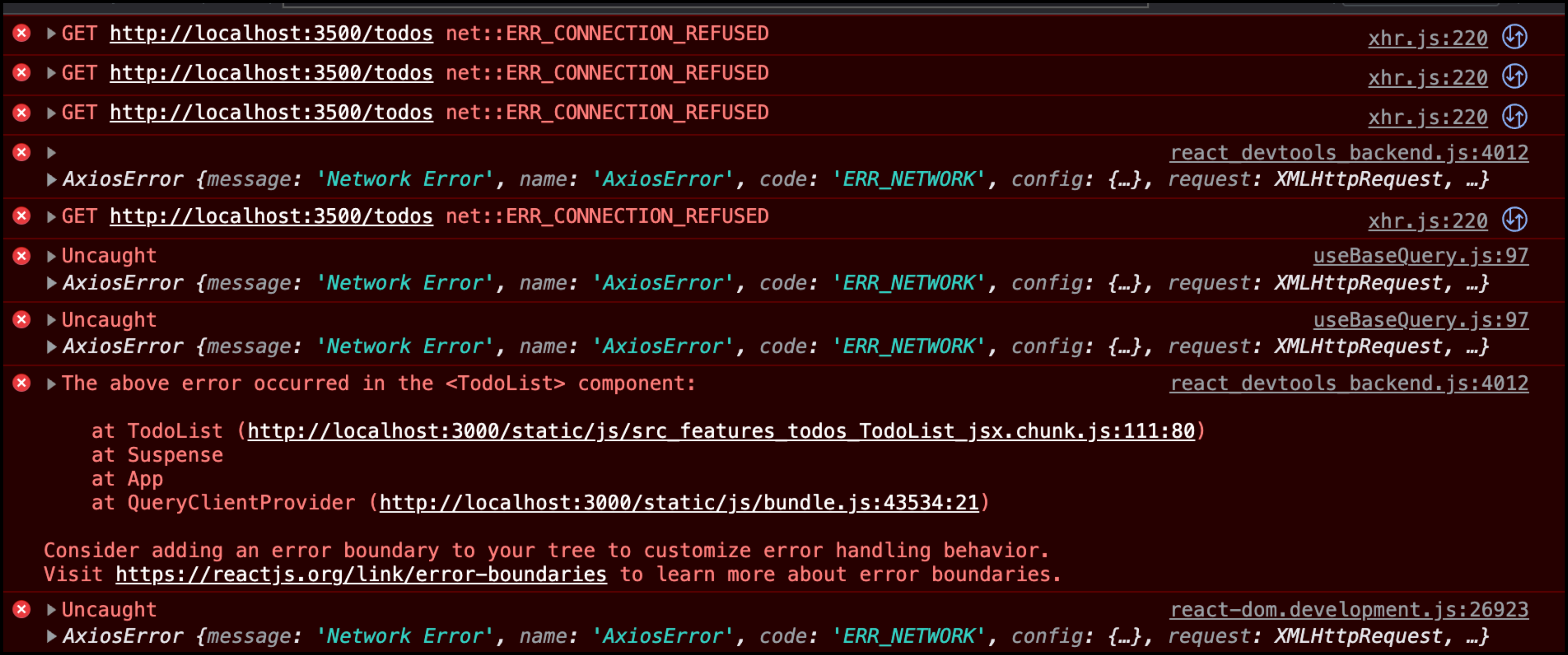
When can we do it?

- Data **fetching** with **Suspense-enabled frameworks** like Relay and Next.js
- **Lazy-loading** component code with `lazy`
- Suspense does **not** detect when data is fetched inside an Effect or event handler.

```
const TodoList = lazy(() => import("./features/todos/TodoList"));
```

Error?

Try to stop the server and refresh



```
▶ GET http://localhost:3500/todos net::ERR_CONNECTION_REFUSED                                    xhr.js:220
▶ GET http://localhost:3500/todos net::ERR_CONNECTION_REFUSED                                    xhr.js:220
▶ GET http://localhost:3500/todos net::ERR_CONNECTION_REFUSED                                    xhr.js:220
▶                                                                        react_devtools_backend.js:4012
▶ AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {…}, request: XMLHttpRequest, …}
▶ GET http://localhost:3500/todos net::ERR_CONNECTION_REFUSED                                    xhr.js:220
▶ Uncaught                                                                          useBaseQuery.js:97
▶ AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {…}, request: XMLHttpRequest, …}
▶ Uncaught                                                                          useBaseQuery.js:97
▶ AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {…}, request: XMLHttpRequest, …}
▶ The above error occurred in the <TodoList> component:                  react_devtools_backend.js:4012

    at TodoList (http://localhost:3000/static/js/src_features_todos_TodoList_jsx.chunk.js:111:80)
    at Suspense
    at App
    at QueryClientProvider (http://localhost:3000/static/js/bundle.js:43534:21)

Consider adding an error boundary to your tree to customize error handling behavior.
Visit https://reactjs.org/link/error-boundaries to learn more about error boundaries.
▶ Uncaught                                                              react-dom.development.js:26923
▶ AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {…}, request: XMLHttpRequest, …}
```

ErrorBoundary

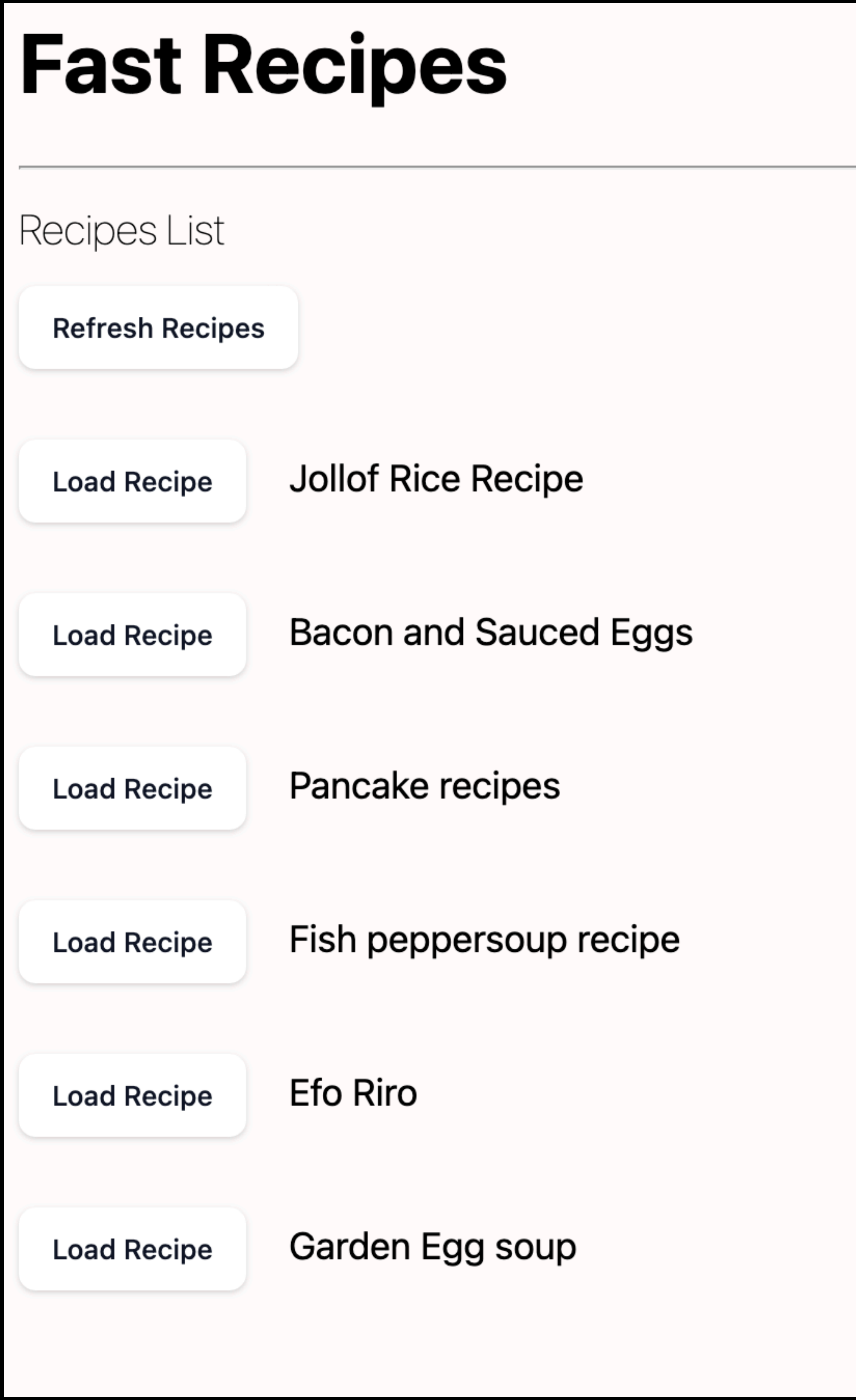
Wrap components with an error boundary

```
// note that resetErrorBoundary is already a function, we are not calling it
const ErrorFallback = ({ error, resetErrorBoundary }) => (
  <div>
    <h1>An error has occured</h1>
    <Button onClick={resetErrorBoundary}>Try again</Button>
    <pre style={{ whiteSpace: "normal" }}>
      An error has occurred: {error.message}
    </pre>
  </div>
);

function App() {
  const { reset } = useQueryErrorResetBoundary();
  return (
    <ErrorBoundary onReset={reset} fallbackRender={ErrorFallback}>
      <Suspense fallback={<Loading />}>
        <TodoList />
      </Suspense>
    </ErrorBoundary>
  );
}
```

Demo: Fast Recipes

```
curl -X POST \
  'http://localhost:8081' \
  --header 'Accept: */*' \
  --header 'Content-Type:
application/json' \
  --data-raw '{
    "id": 7,
    "title": "Stir Fry
Brocolli",
    "content": "Make sure to use
olive oil"
}'
```



Fast Recipes

Recipes List

Load Recipe	Jollof Rice Recipe
Load Recipe	Bacon and Sauced Eggs
Load Recipe	Pancake recipes
Load Recipe	Fish peppersoup recipe
Load Recipe	Efo Riro
Load Recipe	Garden Egg soup