# RTK Query + TypeScript

---

## Overview/learning goals

- Understand how RTK Query fits into Redux Toolkit and React apps.
- Create `createApi` endpoints for CRUD operations.
- Use RTK Query hooks from components (TypeScript-first).
- Implement optimistic UI updates with `updateQueryData` (or equivalent) and rollback on error.
- Learn how to type API requests/responses cleanly with TypeScript.
- Learn how to test optimistic flows (success + failure) manually.

---

## Project brief (mini assignment)

Build a tiny **Todo** feature (single-page; no backend required for the spec — can use json-server or mock server for testing). Features the junior dev will implement and demonstrate:

1. List todos (GET)
2. Add todo (POST)
3. Toggle todo completed state (PATCH)
4. Delete todo (DELETE)

For add / toggle / delete: perform **optimistic updates** so the UI changes immediately and rolls back if the network request fails.

TypeScript is mandatory for all files/types.

---

## Required deliverables

1. A short README explaining architecture and decisions (1–2 pages).

2. Type declarations for the todo model and request/response payloads.

3. A description of the RTK Query slice: list of endpoints, tags used, and reasoning for caching/invalidations (no code, but clear plain-English descriptions).

4. A short write-up (or annotated screenshots) showing optimistic update flows:

- Add: show UI state immediately after add, then final state after server response.
- Toggle: show UI before, optimistic state, and rollback example.
- Delete: show UI immediate removal, and rollback example.

5. A brief test plan describing how to manually simulate network success and failure and the expected behavior.

6. A short demo video (1–3 minutes) or GIF showing the optimistic add/toggle/delete flows in action.

7. A checklist showing acceptance criteria passed/failed and reviewer notes.

---

## Step-by-step task list

1. Define TypeScript types:

    - Todo (id, title, completed, optional timestamps)
    - Create and update payload types

2. Design the API slice in plain English:

    - Base URL chosen (local json-server or mock)
    - Endpoints: getTodos, addTodo, toggleTodo, deleteTodo
    - Which endpoints provide/invalidates which tags and why

3. Implement UI components (React + TS):

    - TodoList: consumes list, shows loading/error states
    - AddTodo: small form for adding
    - TodoItem: toggle checkbox and delete button

4. Implement optimistic updates in each mutation:

    - Add: insert temporary client-side item (unique temp id), then replace with server result on success, or undo on failure.
    - Toggle: immediately flip `completed` in cache, undo if request fails.
    - Delete: optimistically remove from cache, restore on failure.

5. Handle rollback correctly:

    - Use the cache-patching mechanism that supports undo (describe and use it).
    - Ensure error handling shows a user-visible notification on failure.

6. Test by forcing failures:
    - Stop mock server or configure mock to return 500 for one request.
    - Verify optimistic change is applied and then undone.
7. Document everything (README + screenshots/video).

---

## How to test optimistic flows (manual test plan)

- Baseline: start with 5 todos.
- Add flow:
    1. Submit add form — UI should show new todo immediately (temp state).
    2. If server succeeds: temp item replaced with server item (id changes to server id).
    3. If server fails: temp item disappears and an error banner appears.
- Toggle flow:
    1. Click checkbox — UI flips immediately.
    2. If server succeeds: state remains.
    3. If server fails: UI flips back and a message appears.
- Delete flow:
    1. Click delete — item disappears immediately from list.
    2. If server succeeds: item stays removed.
    3. If server fails: item reappears and a message appears.
- Simulate failure options:
    - Temporarily shut down the mock server before clicking.
    - Configure the mock to return 500 on a specific route.
    - Throttle network to show optimistic behaviour more clearly.

---