RTK Query

📘 Как работает RTK Query

Полное пошаговое объяснение (Query + Mutation + Cache + Tags)

📌 1. Что такое RTK Query

RTK Query — это мощный инструмент внутри Redux Toolkit, созданный для того, чтобы упростить работу с API:

GET-запросы, POST, обработку ошибок, кеширование, автоматическое обновление данных.

Основные преимущества:

- ✨ минимум кода
- ⚡ автоматическое кеширование
- 🔄 автоматическое обновление данных после мутаций
- 🧩 хуки создаются автоматически
- 🪄 не нужно писать Thunk, Reducer, Actions

📐 2. Основные элементы RTK Query

🎯 2.1. API Slice

API slice — это «база» всего RTK Query.

Создаётся так:

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

export const api = createApi({
  reducerPath: 'api',
```

```
  baseQuery: fetchBaseQuery({ baseUrl: 'https://example.com/api' }),
  endpoints: () => ({}),
});
```

Что делает `createApi`:

- создаёт reducer для хранения кеша запросов;
- создаёт middleware для управления запросами;
- генерирует хуки;
- умеет инвалидировать кеш (invalidateTags).

🌐 2.2. baseQuery

Это функция, которая отправляет запросы на сервер.

Чаще всего — `fetchBaseQuery`.

RTK Query использует её для отправки:

- GET
- POST
- PUT
- PATCH
- DELETE

🔍 3. Query — запросы на получение данных

3.1. Создание Query Endpoint

```
endpoints: (builder) => ({
  getPosts: builder.query({
```

```
    query: () => '/posts',
  }),
})
```

🔧 RTK Query автоматически создаёт хук:

```
const { data, error, isLoading, isFetching, refetch } = useGetPostsQuery();
```

3.2. Как работает Query под капотом

Когда компонент вызывает хук:

1. RTK Query проверяет, есть ли данные в кеше.
2. Если данных нет → отправляет запрос.
3. Полученные данные сохраняет в store в формате кеша.
4. Компонент автоматически перерисовывается.
5. При повторном вызове хука данные берутся из кеша (запрос не отправляется).
6. Если кеш устарел — RTK Query обновляет данные.

3.3. Дополнительные параметры Query

👉 Параметры запроса

```
getPostById: builder.query({
  query: (id) => `/posts/${id}`,
});
```

Использование:

```
const { data } = useGetPostByIdQuery(10);
```

👉 Время жизни кеша

```
keepUnusedDataFor: 60, // 60 секунд
```

Если компонент размонтирован, данные удалятся через 60 секунд.

👉 Автоперезапросы (рефетчинг)

```
refetchOnFocus: true,       // при возвращении к вкладке
refetchOnReconnect: true,   // при восстановлении сети
refetchOnMountOrArgChange: true,
```

🛠️ 4. Mutation — запросы на изменение данных

Мутации используются для:

- добавления (POST)
- обновления (PUT/PATCH)
- удаления (DELETE)

4.1. Создание мутации

```
addPost: builder.mutation({
  query: (body) => ({
    url: '/posts',
```

```
    method: 'POST',
    body,
  }),
})
```

RTK Query генерирует хук:

```
const [addPost, { data, error, isLoading }] = useAddPostMutation();
```

4.2. Как работает Mutation под капотом

Когда вызывается `addPost(...)`:

1. RTK Query отправляет запрос на сервер.
2. Ожидает ответа.
3. Обновляет состояние мутации.
4. По желанию — *инвалидирует кеш* (invalidateTags), чтобы обновить связанные Query.
5. Все Query, завязанные на тег, автоматически перезапросятся.

4.3. Выполнение мутации

```
const [addPost] = useAddPostMutation();

const handleSubmit = async () => {
  await addPost({ title: 'Hello world', content: '...' });
};
```

Мутация всегда вызывается вручную → в отличие от Query, она **не запускается автоматически**.

🧩 5. Tags — механизм обновления данных

Самая сильная часть RTK Query — система тегов.

5.1. providesTags в Query

```
getPosts: builder.query({
  query: () => '/posts',
  providesTags: ['Posts'],
});
```

Query сообщает RTK Query:

⭐ «Я кэширую данные, которые относятся к тегу 'Posts'».

5.2. invalidatesTags в Mutation

```
addPost: builder.mutation({
  query: (body) => ({
    url: '/posts',
    method: 'POST',
    body,
  }),
  invalidatesTags: ['Posts'],
});
```

После успешной мутации RTK Query:

1. помечает тег 'Posts' как устаревший;
2. автоматически перезапускает все Query, у которых есть `providesTags: ['Posts']`.

🔄 Пример полного взаимодействия

1) Query кеширует данные

```
getPosts → providesTags: ['Posts']
```

2) Mutation изменяет данные

```
addPost → invalidatesTags: ['Posts']
```

3) RTK Query делает refetch

- автоматический повторный запрос `/posts`
- список обновляется без ручного вмешательства

🧬 6. Полный пример API Slice

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

export const api = createApi({
  reducerPath: 'api',
  baseQuery: fetchBaseQuery({ baseUrl: 'https://example.com/api' }),
  tagTypes: ['Posts'],
  endpoints: (builder) => ({
    getPosts: builder.query({
      query: () => '/posts',
      providesTags: ['Posts'],
    }),

    getPostById: builder.query({
      query: (id) => `/posts/${id}`,
```

```
    providesTags: (result, error, id) => [{ type: 'Posts', id }],
  }),

  addPost: builder.mutation({
    query: (body) => ({
      url: '/posts',
      method: 'POST',
      body,
    }),
    invalidatesTags: ['Posts'],
  }),

  deletePost: builder.mutation({
    query: (id) => ({
      url: `/posts/${id}`,
      method: 'DELETE',
    }),
    invalidatesTags: ['Posts'],
  }),
 }),
});

export const {
 useGetPostsQuery,
 useGetPostByIdQuery,
 useAddPostMutation,
 useDeletePostMutation,
} = api;
```

🔄 7. Жизненный цикл Query

1. Компонент вызывает хук.
2. RTK Query проверяет кеш.

3. Если данных нет → отправляет запрос.
4. Полученные данные сохраняет.
5. Возвращает:
    - data
    - isLoading
    - error
6. Если тег инвалидирован — делает новый запрос.

🛠️ 8. Жизненный цикл Mutation

1. Пользователь вызывает мутацию.
2. RTK Query отправляет запрос.
3. Возвращает:
    - data
    - isLoading
    - isError
4. Проверяет `invalidatesTags`.
5. Обновляет кеш.
6. RTK Query начинает рефетч связанных Query.

⚙️ 9. Особые возможности

🔁 9.1. Ручной refetch

```
const { refetch } = useGetPostsQuery();
refetch();
```

🗄️ 9.2. Prefetch (предзагрузка данных)

```
api.util.prefetch('getPosts', undefined, { force: true });
```

🧪 9.3. Ручное обновление кеша

```
api.util.updateQueryData('getPosts', undefined, (draft) => {
  draft.push(newPost);
});
```

Это похоже на `immer` — можно мутировать draft.

🔥 10. Главное, что нужно запомнить

Query

- автоматически выполняется
- автоматически кешируется
- автоматически обновляется при invalidatesTags

Mutation

- выполняется вручную
- обновляет сервер
- автоматически вызывает refetch нужных Query

Tags

- Query → `provideTags`
- Mutation → `invalidateTags`
- RTK Query сам следит за данными

Преимущества

- меньше кода
- меньше ручного Redux
- очень чистая архитектура
- автоматическое обновление UI