

(Remaining) Agenda

- **Data Fetching and Caching**
 - The react-query library
- **Assignment 1 specification**

Data Fetching & Caching.

SPA State (Data)

- **Client state (aka App State).**
 - e.g. Menu selection, UI theme, Text input, logged-in user id.
 - **Characteristics:**
 - **Client-owned; Not shared; Not persisted (across sessions); Always up-to-date.**
 - **Accessed synchronously.**
 - **useState() hook**
 - **Management - Private to a component or Global state (Context).**

SPA State (Data)

- **Server state (The M in MVC).**
 - e.g. list of ‘discover’ movies, movie details, friends.
 - **Characteristics:**
 - **Persisted remotely. Shared ownership.**
 - **Accessed asynchronously → Impacts user experience.**
 - **Can change without client’s knowledge → Client can be ‘out of date’.**
 - **useState + useEffect hooks.**

SPA Server State.

- **Server state characteristics (contd.).**
 - **Management options:**
 1. **Spread across many component.**
 - **Good separation of concerns. (+)**
 - **Unnecessary re-fetching. (-)**
 2. **Global state (Context).**
 - **No unnecessary re-fetching. (+)**
 - **Fetching data before its required. (-)**
 - **Poor separation of concerns. (-)_**
 3. **3rd party library – e.g. Redux**
 - **Same as 2 above.**
- **We want the best of 1 and 2, if possible.**

Sample App.

[Home](#)

Movie List

- [The Conjuring: The Devil Made Me Do It](#)
- [Cruella](#)
- [Wrath of Man](#)
- [The Unholy](#)
- [Spiral: From the Book of Saw](#)
- [A Quiet Place Part II](#)
- [Army of Darkness](#)
- [Mortal Kombat](#)
- [Godzilla](#)

[Home](#)

Movie Details

```
{
  "adult": false,
  "backdrop_path": "/6MKr3KgOLmzOP6MSuZERO41Lpkt.jpg",
  "belongs_to_collection": {
    "id": 837007,
    "name": "Cruella Collection",
    "poster_path": null,
    "backdrop_path": null
  },
  "budget": 200000000,
  "genres": [
    {
      "id": 35,
      "name": "Comedy"
    },
    {
      "id": 80,
      "name": "Crime"
    }
  ],
  "homepage": "https://movies.disney.com/cruella",
  "production_companies": [
```

- Both pages make a HTTP Request to a web API (TMDB)

Sample App – The Problem.

The screenshot shows a web browser at localhost:3000 displaying a 'Movie List' application. The application has a search bar and a list of movies. The network tab is open, showing a list of HTTP requests. Red arrows indicate that each movie in the list triggers an HTTP request to TMDB.

Movie List

Search

- [The Conjuring: The Devil Made Me Do It](#)
- [Cruella](#)
- [Wrath of Man](#)
- [The Unholy](#)
- [Spiral: From the Book of Saw](#)
- [A Quiet Place Part II](#)
- [Army of the Dead](#)
- [Mortal](#)
- [Godzill](#)
- [Endang](#)
- [Tom Cl](#)

Slide 15

Network

Filter: XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies

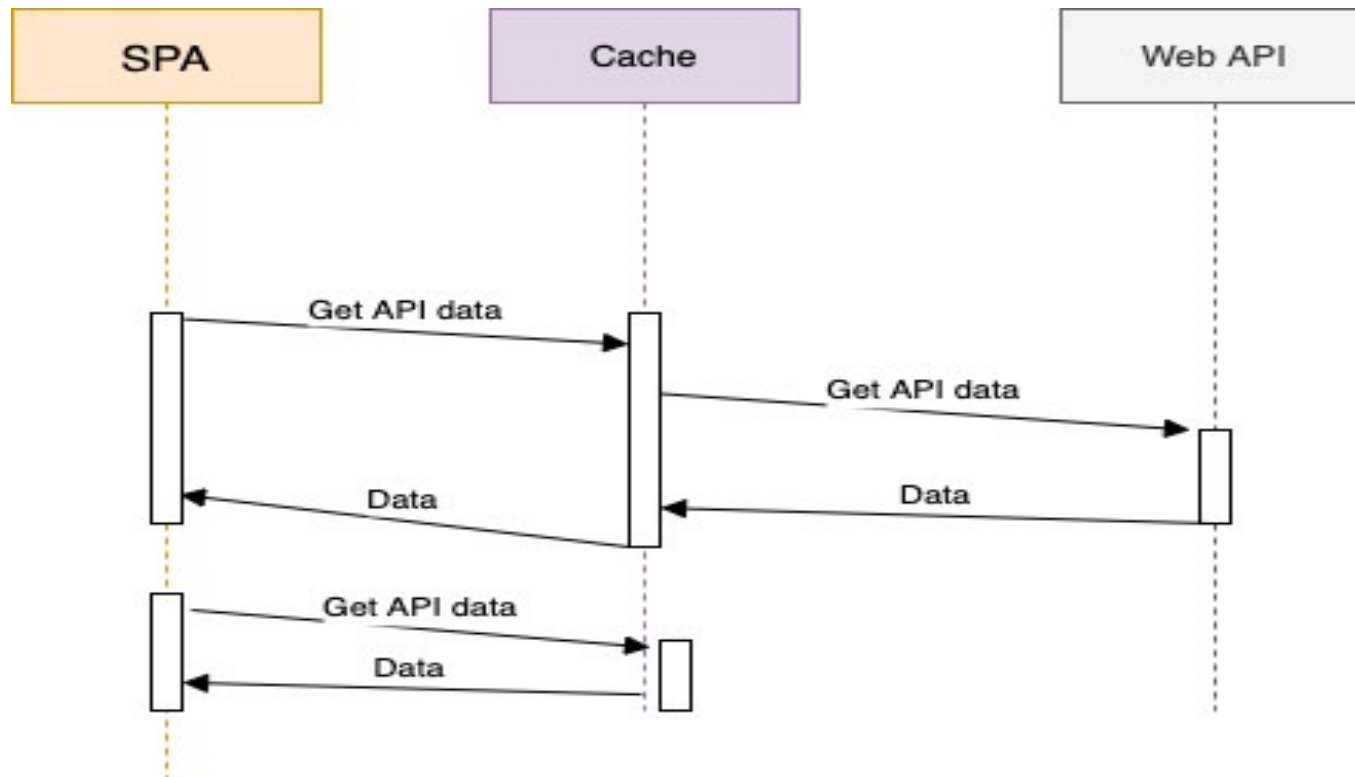
Blocked Requests

Name	Status	Type	Initiator	Size	Time	Waterfall
movie?api_key=...	200	fetch	VM24:1	(di...	1 ...	
423108?api_key...	200	fetch	VM24:1	1.5...	30...	
movie?api_key=...	200	fetch	VM24:1	(di...	1 ...	
423108?api_key...	200	fetch	VM24:1	(di...	1 ...	
movie?api_key=...	200	fetch	VM24:1	(di...	1 ...	
337404?api_key...	200	fetch	VM24:1	1.4...	27...	
movie?api_key=...	200	fetch	VM24:1	(di...	1 ...	

- Every navigation to the Home page triggers an HTTP request to TMDB.
- Similarly for the Detail page.
- Both pages use useEffect and useState hooks.

Sample App – The Solution. .

- *Cache* (store temporarily) the API data locally in the browser.
- **Reduces the workload on the backend** for read intensive apps.
- **Speeds up the rendering time** for revisited pages.



Caching (General).

- **Caches are** in-memory datastores **with high** performance and **low** latency.
- **Simple** key-value **datastores** **structure**.
 - **Keys must be unique**.
 - **Value can be any serializable data type – Object, Array, Primitive.**
- **Cache hit** – The requested data is in the cache.
- **Cache miss** - The requested data is not in the cache.
- **Caches have a simple interface:**
 - `serializedValue = cache.get(key)`
 - `cache.delete(key)`
 - `cache.purge()`
- **Cache entries have a time-to-live (TTL).**



The react-query library

- **3rd party JavaScript (React) caching library.**
 - **Provides a set of hooks.**

e.g. `const { data, error, isLoading, isError } = useQuery(key, queryFunction);`

 - **data – from the cache (hit) or returned by the API (miss).**
 - **error – error response from API.**
 - **isLoading(boolean) – true while waiting for API response.**
 - **isError (boolean) – true when API response is an error status.**
- **Causes a component to re-render on query completion.**
- **Replaces your `useState` and `useEffect` hooks.**

The query key.

- *“Query keys can be as simple as a string, or as complex as an array of many strings and nested objects. As long as the query key is serializable, and **unique to the query's data***”

```
e.g. const { ....., } =  
      useQuery( ["movie", { id: 1234 }], getMovie);  
// The query function.  
export const getMovie = (args) => {  
  const [, idPart] = args.queryKey;  
  const {id} = idPart  
  .... Do HTTP GET using a movie id of 1234
```

react-query DevTools.

- Allows us to observe the current state of the cache datastore – great for debugging.

The screenshot displays the React Query DevTools interface overlaid on a web application. The application, running on localhost:3000, is titled "Movie List" and features a search bar and a list of movies, including "Cruella" and "The Conjuring: The Devil Made Me Do It".

The DevTools interface is divided into several sections:

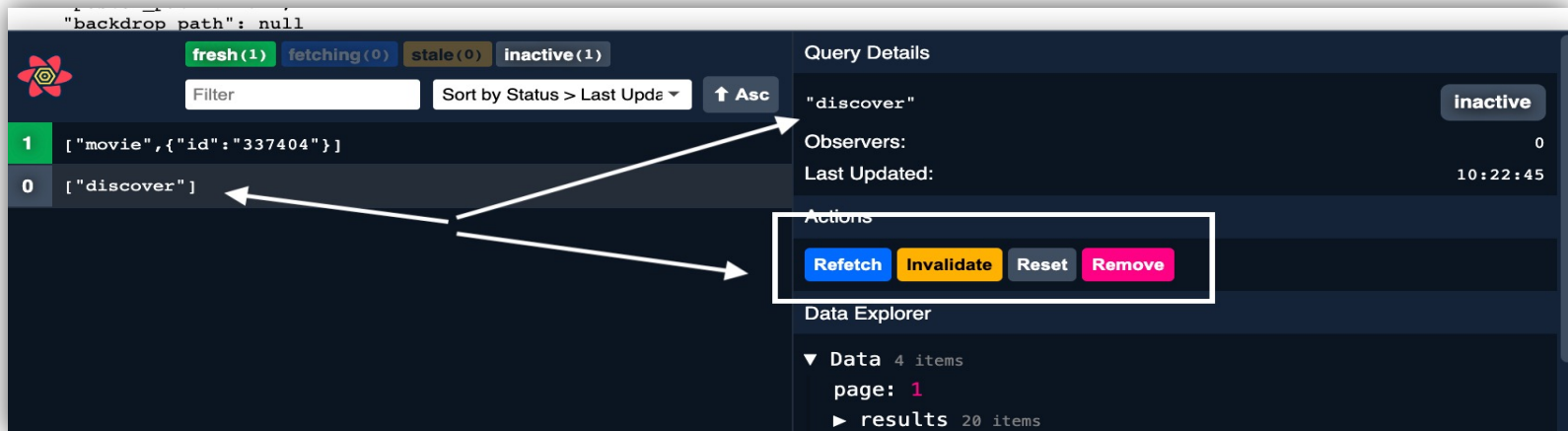
- Cache State:** A table showing the current state of the cache. It includes a status bar with "fresh (1)", "fetching (0)", "stale (0)", and "inactive (2)". The table lists the following queries:

Index	Query
1	["discover"]
0	["movie", {"id": "337404"}]
0	["movie", {"id": "423108"}]
- Query Details:** A panel for the selected query "discover", showing it is "fresh" with 1 observer. It includes a "Last Updated" timestamp of 09:40:07 and a set of actions: "Refetch", "Invalidate", "Reset", and "Remove".
- Data Explorer:** A panel showing the data for the selected query, indicating 4 items. It displays "page: 1", "results: 20 items", and "total_pages: 500".

Arrows indicate the relationship between the cache state and the query details/data explorer panels.

react-query DevTools.

- Allows us to manipulate cache entries.



- Refresh – force an immediate re-request of data from the API.
- Invalidate – set entry as 'stale'. Cache will request update from web API when next required by the SPA.
- Reset – only applies when app can mutate the API's data.
- Remove – remove entry from cache immediately.

Summary

- **State Management - The M in MVC**
- **State:**
 1. **Client/App state.**
 2. **Server state.**
- **Cache server state locally in the browser.**
 - **Avoid unnecessary HTTP traffic → Reduce page load time**
 - **Be aware of cache entry staleness → Use TTL to minimize staleness.**
- **The react-query library**
 - **A set of hooks for cache interaction.**