Agenda

- Data Fetching and Caching
  - The react-query library

- Assignment 1 specification

Data Fetching & Caching.

SPA State (Data) - Client

- 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

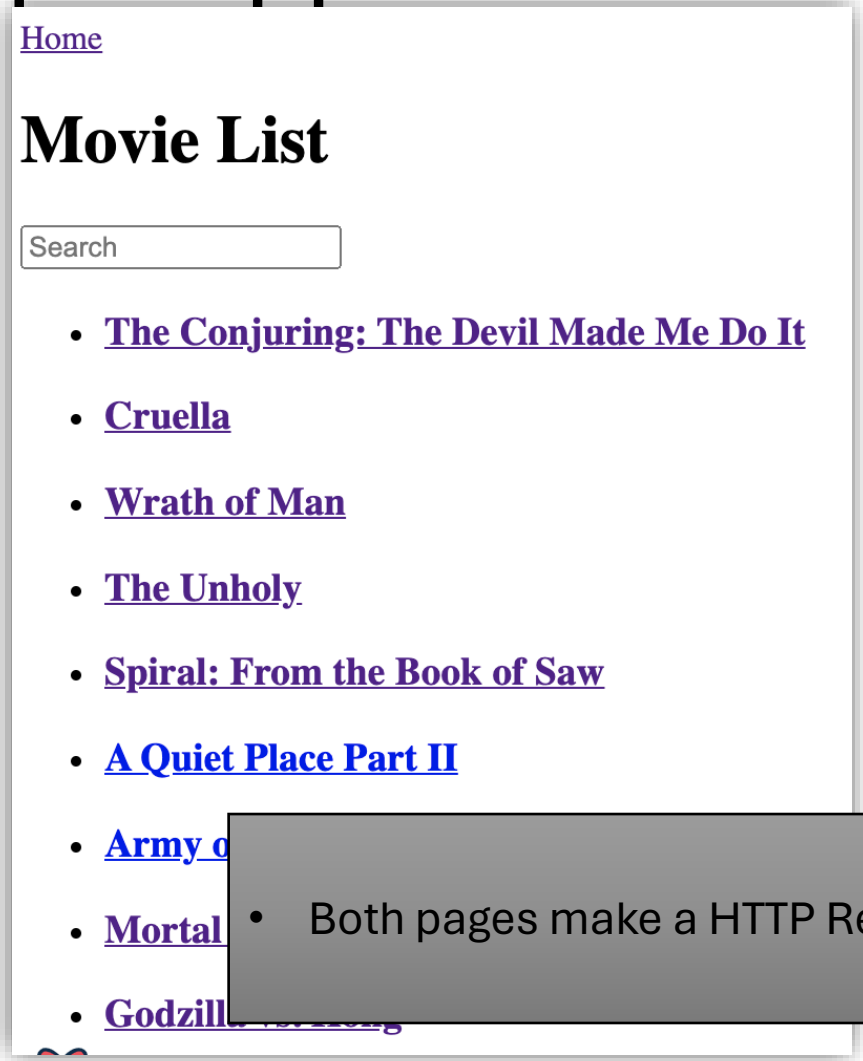
- Server state (The M in MVC).
  - e.g. list of ‘discover’ movies, movie details, friends.

- 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  - Can change without client’s knowledge → Client can be ‘out of date’.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.

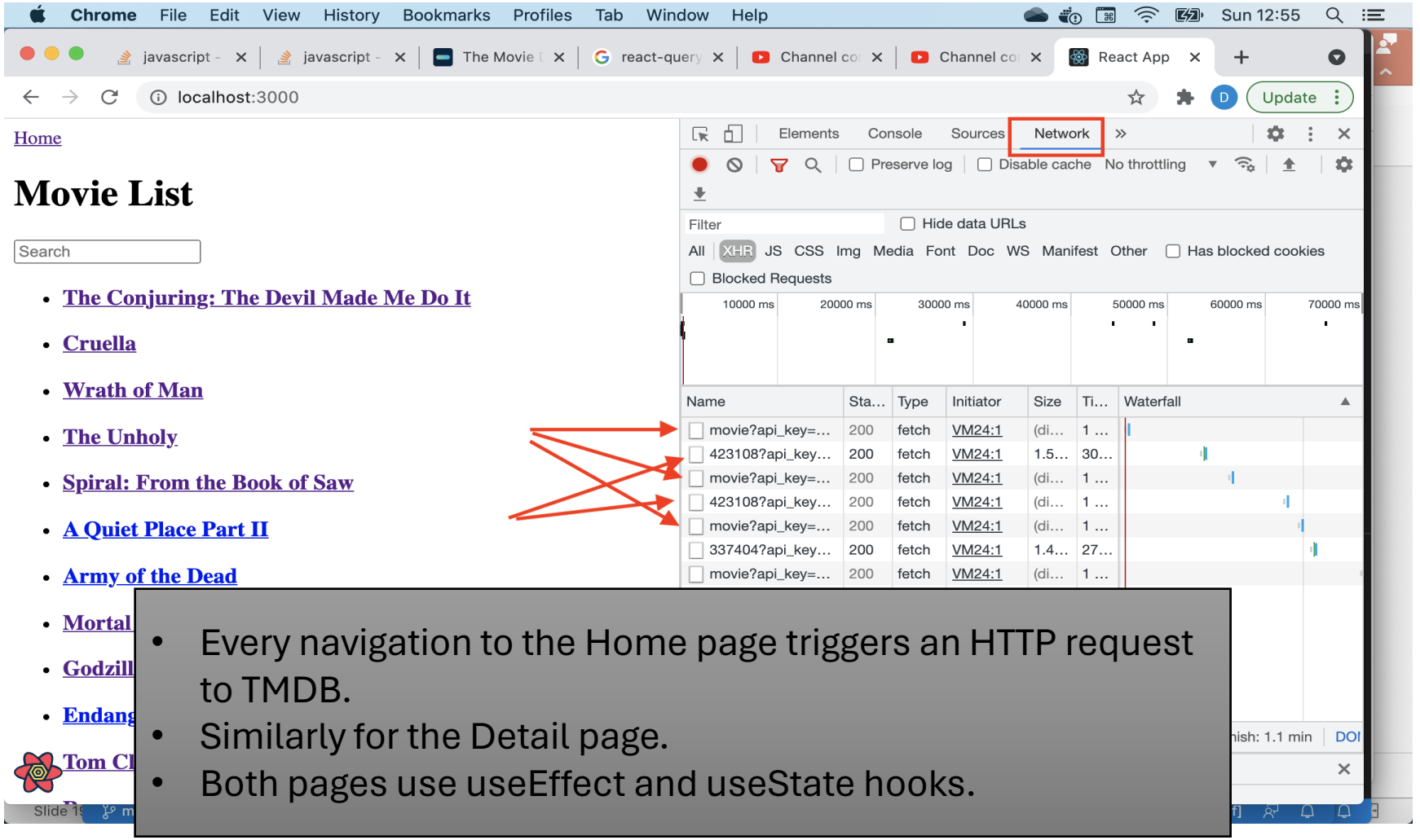
- Server state characteristics (contd.).
  - Management options:
    1. Spread across many component.
       - Good separation of concerns. (+)
       - Unnecessary re-fetching. (-)
    2. Global state (Context).
       - No unnecessary re-fetching. (+)
       - Fetching data before its required. (-)
       - 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.



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

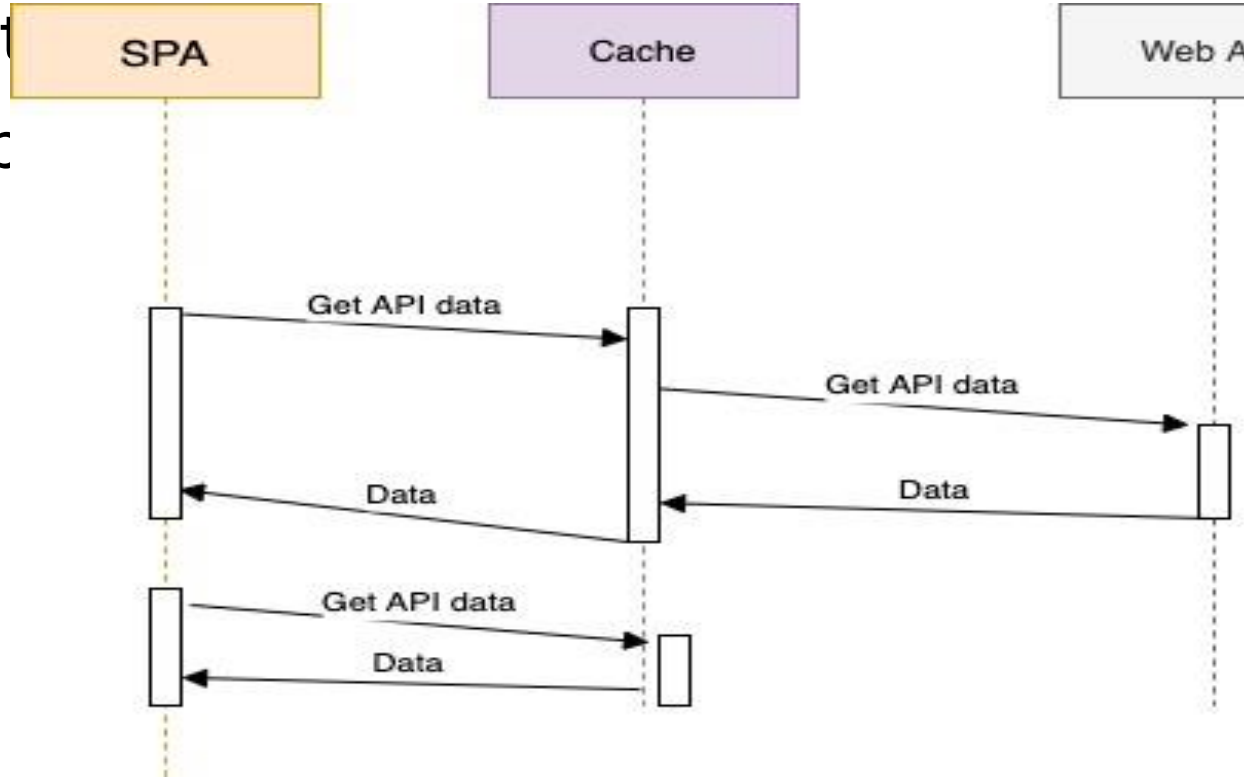
Sample App – The Problem.



- 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 int
- Sp ges.



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 serialisable 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 .....”*

```
const { id } = useParams();
const {…} = useQuery<MovieT, Error>(["movie", id],
                                    () => getMovie(id ||
""));
```

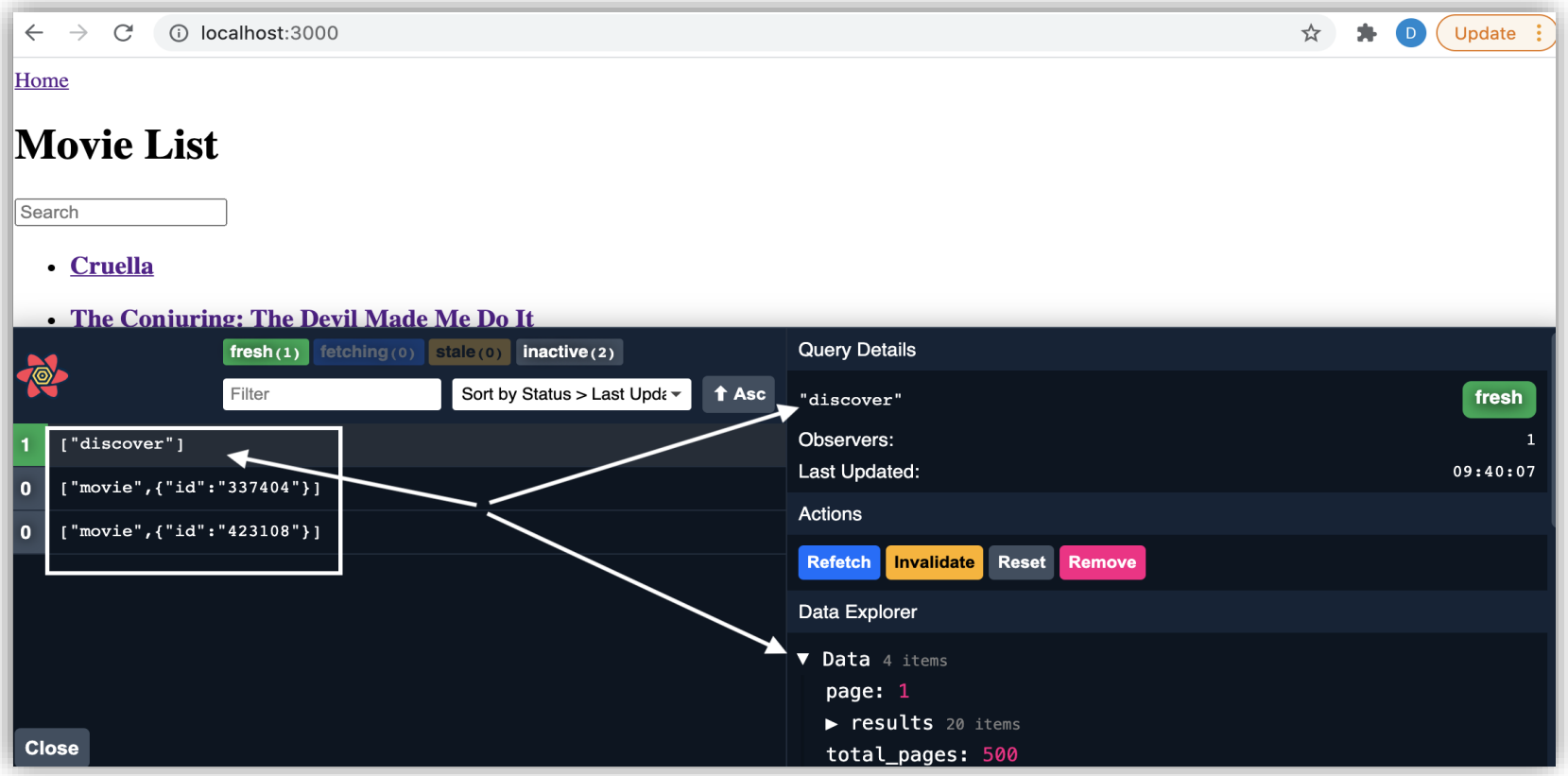
// The query function.

- ```
  export const getMovie = (id: string) => {
  ```

.... Do HTTP GET using a movie id of 1234
  ```

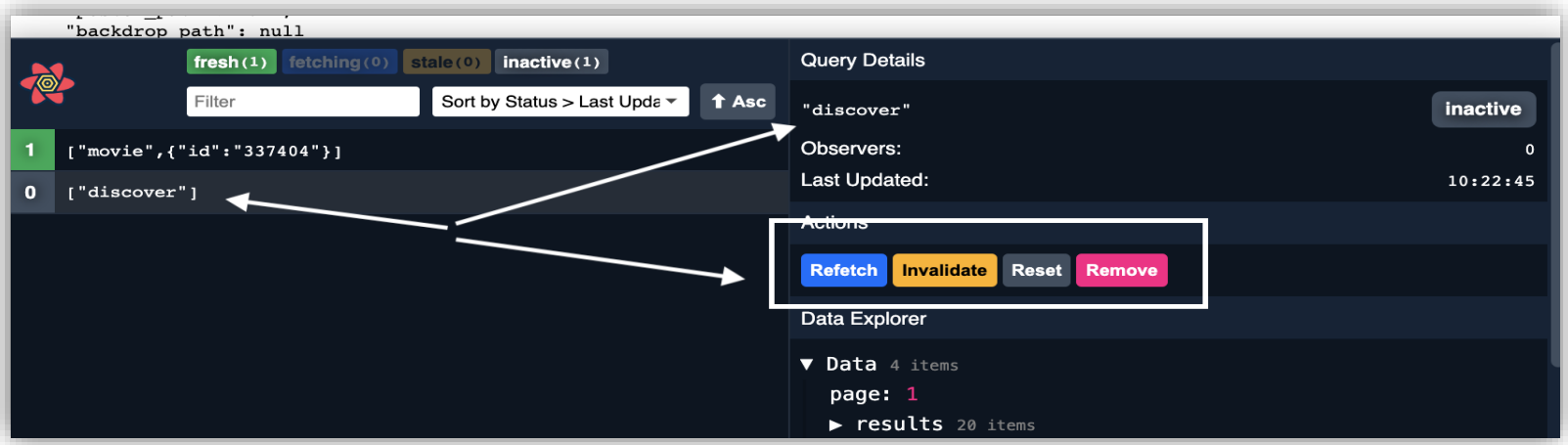
react-query DevTools.

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



react-query DevTools.

- Allows us to manipulate cache entries.



- Refresh – force an immediate re-request of data from theAPI.
- 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.