(Remaining) Agenda

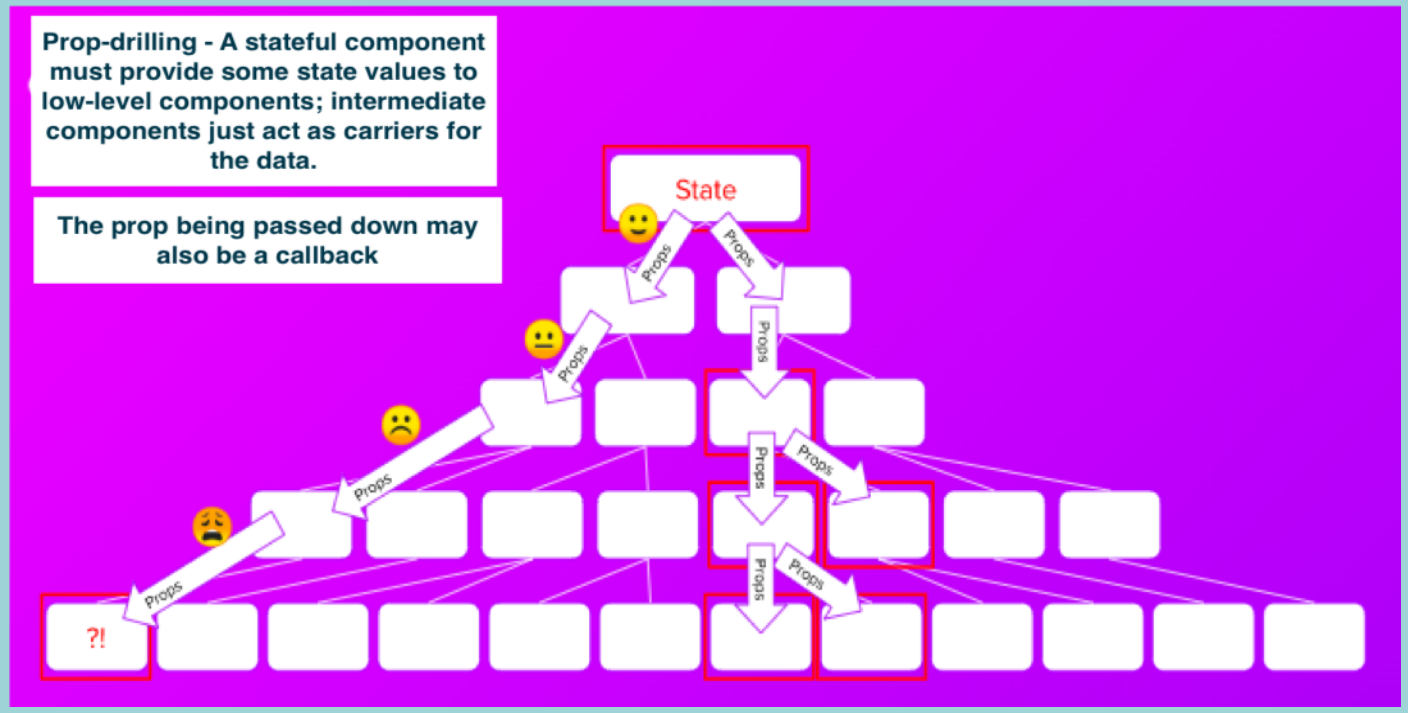
- **Design patterns (Contd.)**
  - **The Provider Pattern**
- **Data Fetching and Caching**
  - **The react-query library**
- **Routing.**
  - **Protected routes and authentication.**

React Context.

The Provider pattern

The Provider pattern – When?

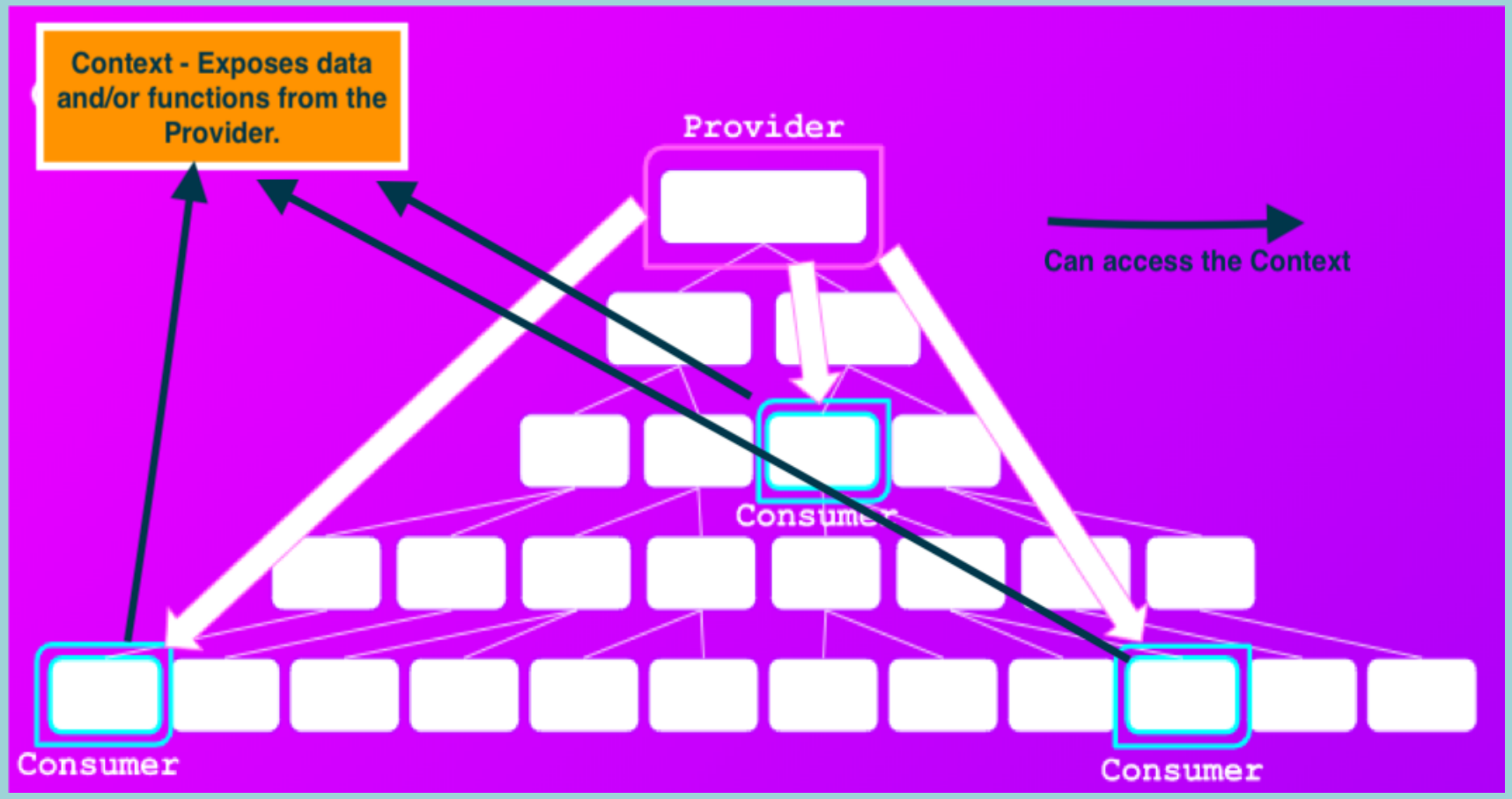
- **Use cases:**
  1. Sharing data/state **with multiple components**, i.e. global data**, e.g. favourite movies.**
  2. **To avoid** prop-drilling**.**



The Provider pattern – How?

- **React Implementation steps:**
  1. **Declare a component for managing the shared data – the** Provider **component.**
  2. **Create a** context **construct and link it to the Provider.**
  3. **Place the shared data inside the context.**
  4. Compose **the Provider with other components – allow other components access the context.**
- Context **– the glue for the Provider pattern in React.**
  - Wraps **the data (and behaviour) to be shared.**
  - **Avoids** prop drilling**.**
  - **Provider component manages the context.**
  - **Consumer accesses the context with the** useContext **hook**

The Provider pattern – React Context.



The Provider pattern – Implementation

- **Declare the Provider component:**

```
export const SomeContext = React.createContext(null)

const ContextProvider = props => {
  . . . Use useState and useEffect hooks to
  . . . initialize global state variables
  return (
    <SomeContext.Provider
          value={{ key1: value1,. . . . }} >
      {props.children}
    </SomeContext.Provider>
  );
};
export default ContextProvider
```

- **We associate the** Context **with the** Provider **using** <contextName.Provider>.
- **The** values object **declares the context's content. .**
  - **Can be functions as well as (state) data.**

The Provider pattern – Implementation.

- **Integrate (Compose) the Provider with the rest of the app, using the Container pattern**

```
const App = () => {
    return (
        <BrowserRouter>
          <ContextProvider>
              . . . . . . . . .
          </ContextProvider>
        </BrowserRouter>
    );
  };

  ReactDOM.render(<App />, document.getElementById("root"));
```

- **The Provider's** subordinate components **can now 'request' access to the context.**

The Provider pattern – Implementation.

- useContext **hook – givss a component access to a context/**

  Const contextRef = useContext(ContextName)

  // contextRef points at context's values object.

```
import React, { useContext } from "react";
import {SomeContext} from '......'

const ConsumerComponent = props => {
  const context = useContext(SomeContext);

  . . . access context values with 'context.keyX'

};
```

```
export const SomeContext = React.createContext(null)

const ContextProvider = props => {
  . . . Use useState and useEffect hooks to
  . . . initialize global state variables
  return (
    <SomeContext.Provider
        value={{ key1: value1,. . . . }} >
      {props.children}
    </SomeContext.Provider>
  );
};
export default ContextProvider
```

The Provider pattern – Implementation.

- **For improved** separation of concerns**, use multiple context instead of a 'catch all' context.**

```
const App = () => {
    return (
        <BrowserRouter>
          <ContextProviderA>
            <ContextProviderB>
              . . . . . . . .
            </ContextProviderB>
          </ContextProviderA>
        </BrowserRouter>
    );
  };
```

The Provider pattern.

- **When NOT to use a Context:**
  1. **To avoid ‘shallow’ prop drilling.**
     - **Prop drilling is faster for ‘shallow’ cases.**
  2. **For state that should be kept local to a component, e.g. web form inputs.**
  3. **For large object - monitor performance and refactor as necessary.**

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

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 o
- Mortal
- Godzill

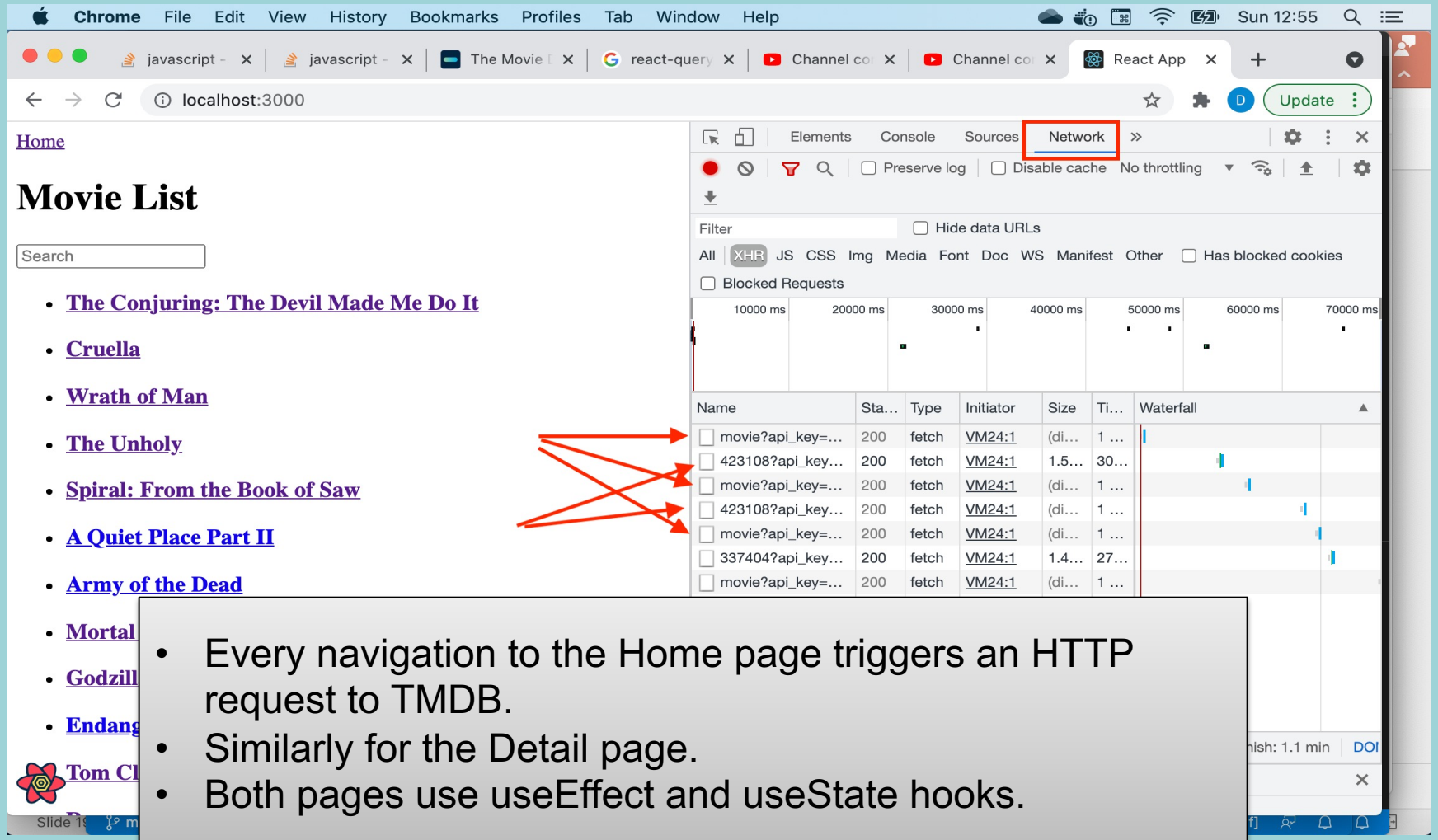
Home

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",
```


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

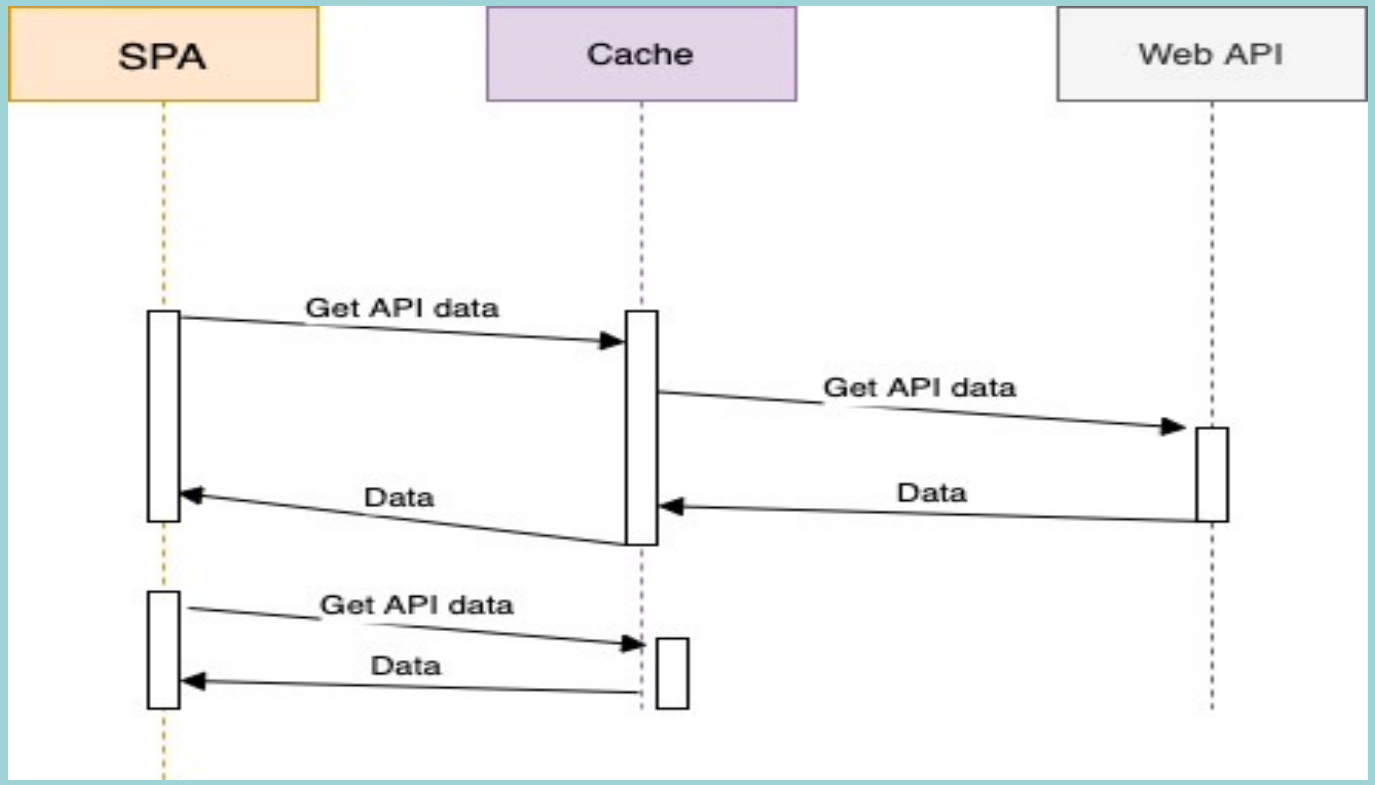
Sample App – The Problem.



- 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 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-valu**e datastores structure.**
  - **Keys must be <u>unique</u>.**
  - **Value can be any <u>serializable</u> 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.**

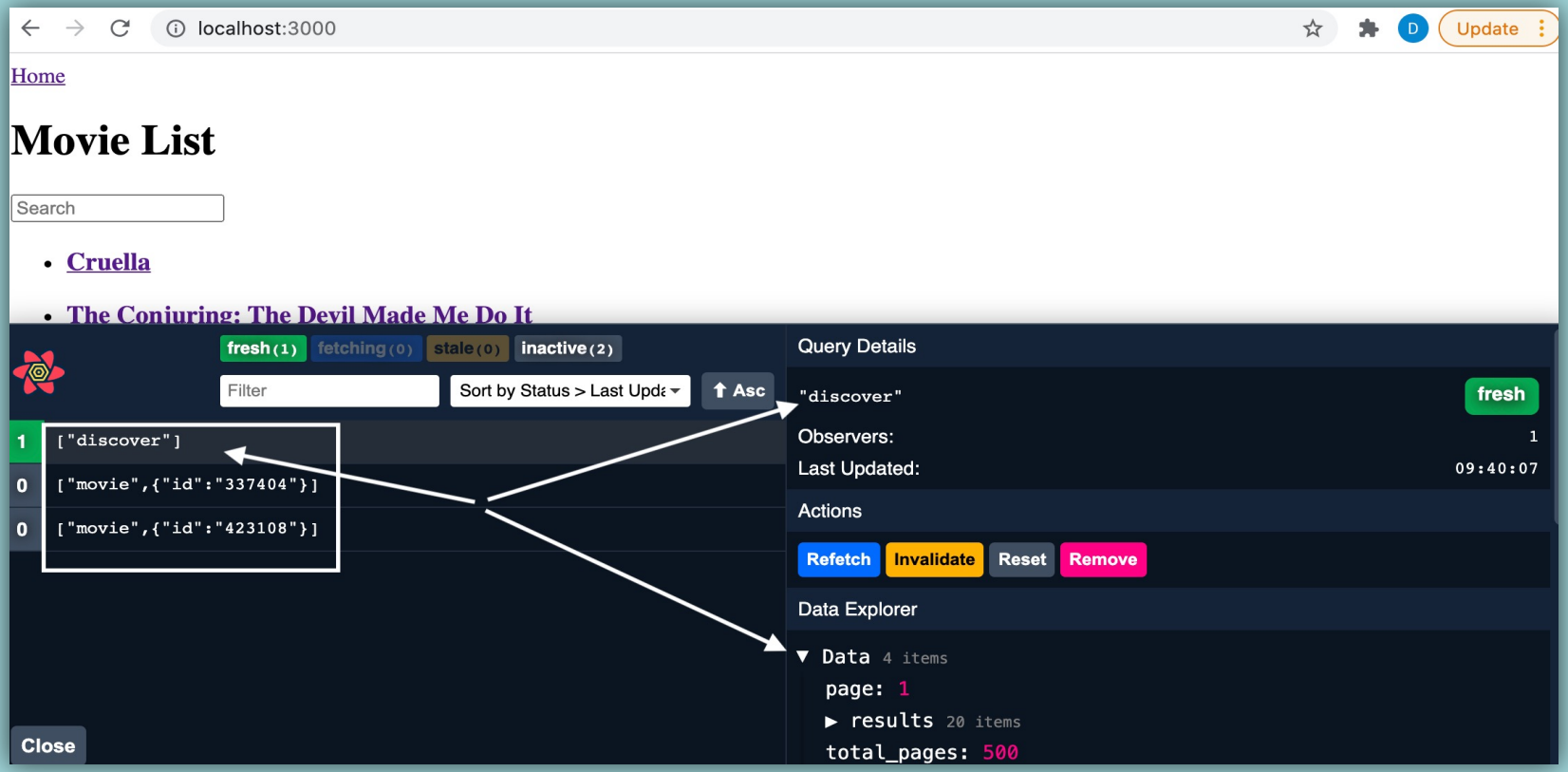
The query key.

- *“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
```

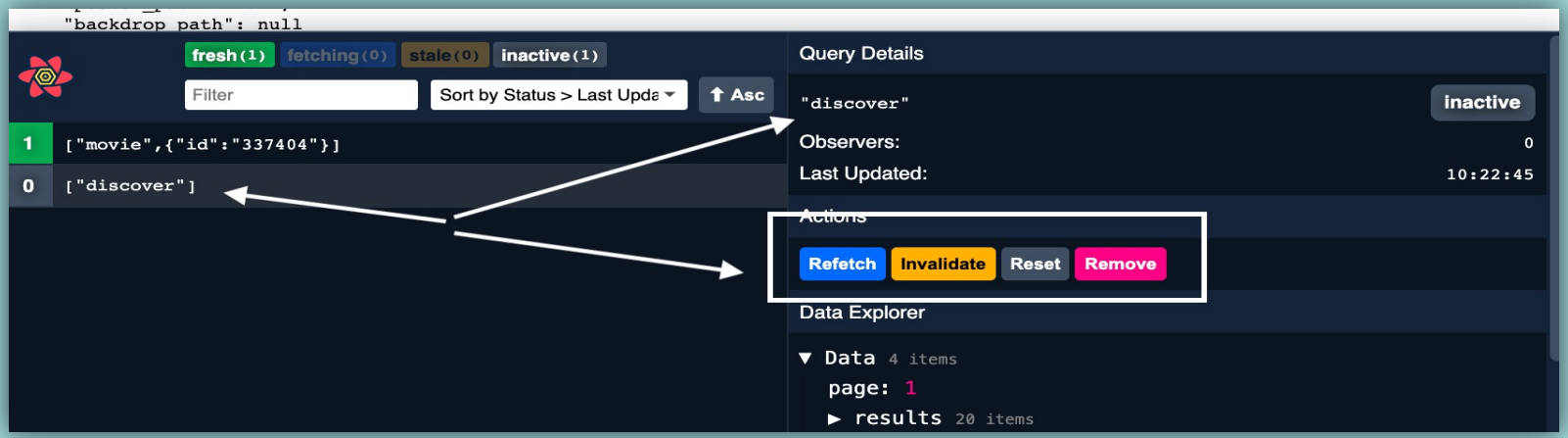
react-query DevTools.

- **Allows us to <u>observe</u> the current state of the cache datastore – great for debugging.**



react-query DevTools.

- **Allows us to <u>manipulate</u> 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.**

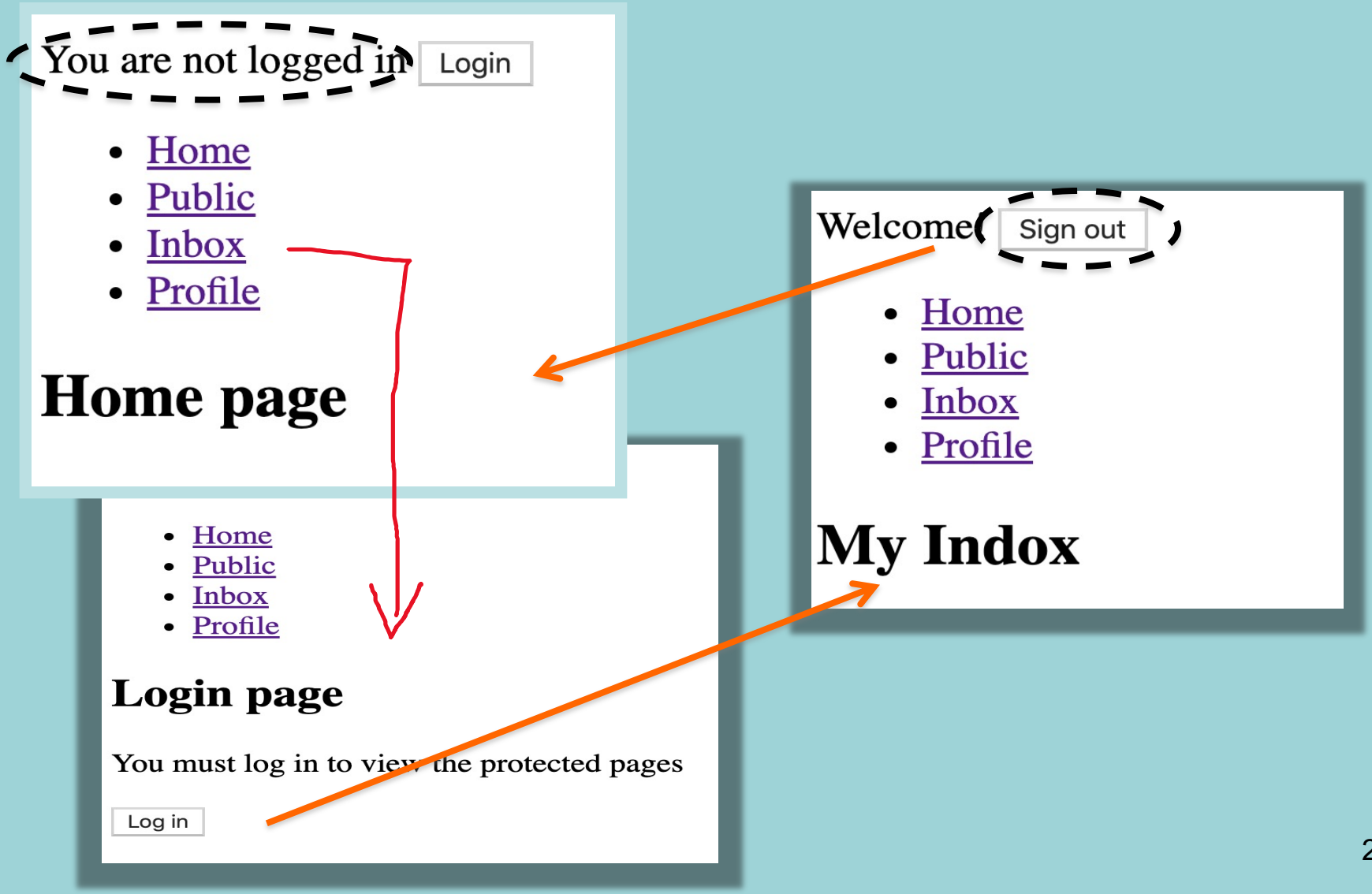
- **Cache server state locally in the browser.**
  - **Avoid unnecessary HTTP traffic → Reduce page load time**
  - **Be aware of cache entry** staleness **→ Use TTL to minimize staleness.**

- **The react-query library**
  - **A set of hooks for cache interaction.**

Authentication and Protected/Private Routes

(See Routing samples Archive)

Objective


You are not logged in
Login
Home
Public
Inbox
Profile
Home page
Welcome
Sign out
Home
Public
Inbox
Profile
My Indox
Home
Public
Inbox
Profile
Login page
You must log in to view the protected pages
Log in

Protected Routes.

- **Not native to React Router.**
- **We need a custom solution.**
- **Solution objective: Clear, declarative style for declare views/pages requiring authentication:**

```
<Switch>
  <Route path="/public" component={PublicPage} />
  <Route path="/login" component={LoginPage} />
  <Route exact path="/" component={HomePage} />
  <PrivateRoute path="/inbox" component={Inbox} />
  <PrivateRoute path="/profile" component={Profile} />
  <Redirect from="*" to="/" />
</Switch>
```

Protected Routes.

- **Solution features:**

1. React Context - **store user's authentication status.**
2. Programmatic navigation – **e.g. redirect unauthenticated user to login page.**
3. **Remember user's intent before forced authentication.**

Protected Routes - Implementation

- **Solution elements: The AuthContext.**

```
import React, { useState, useEffect, createContext } from "react";

export const AuthContext = createContext(null);

const AuthContextProvider = (props) => {
  const [user, setUser] = useState({ username: null, password: null });

  const authenticate = (username, password) => {
    // .... Validation user credentials somehow ......
    setUser({ username, password });
  };
  const isAuthenticated = user.username === null ? false : true
  const signout = () => {
    setTimeout(() => setUser( { username: null, password: null } ), 100);
  };

  return (
    <AuthContext.Provider
      value={{
        isAuthenticated,
        authenticate,
        signout,
      }}
    >
      {props.children}
    </AuthContext.Provider>
  );
};
export default AuthContextProvider;
```

Protected Routes - Implementation

- **Solution elements (Contd.): <PrivateRoute />**

```
<PrivateRoute path="/inbox" component={Inbox} />
```

```
5   const PrivateRoute = (props) => {
6     const context = useContext(AuthContext);
7     const { pathname } = useLocation()
8
9     const { component: Component, ...rest } = props;
10
11    return context.isAuthenticated ? (        You, 9 months ago • Initial
12      <Route {...rest} render={(props) => <Component {...props} />} />
13    ) : (
14      <Redirect
15        to={{
16          pathname: "/login",
17          state: { from: pathname },
18        }}
19      />
20    );
21  };
```

```
{pathname: "/inbox", sear
key: "0pfafo"}
  hash: ""
  key: "0pfafo"
  pathname: "/inbox"
  search: ""
  state: undefined
  __proto__: Object
```

Protected Routes - Implementation

- **Solution elements (Contd.): <LoginPage>**

```
5   const LoginPage = (props) => {
6     const context = useContext(AuthContext);
7     const { state } = useLocation()
8     const login = () => {
9       const username = Math.random().toString(36).substring(7);
10      context.authenticate(username, "pass1");
11    };
12        You, 9 months ago • Initial structure …
13    const { from } = state || { from: { pathname: "/" } };
14
15    return context.isAuthenticated ? (
16      <Redirect to={from} />
17    ) : (
18      <>
19        <h2>Login page</h2>
20        <p>You must log in to view the protected pages </p>
21        {/* Login web form  */}
22        <button onClick={login}>Submit</button>
23      </>
24    );
25  };
```

Protected Routes - Implementation

- **See** src/sample9 **from routing samples archive**