CIT/T Assignment 6 — Fetching person images

Complex IT Systems

November 12, 2024

The *Movie Database* (TMDB), found at `https://www.themoviedb.org/`, is an alternative source of information about movies and people that we can use to supplement the database handed out as part of the project portfolio. In particular, it has images of actors and directors that are not available in the database handed out. The information in TMDB is free and available online, so we need not download anything to get access to that information.

TMDB has a public API that is documented at

`https://developer.themoviedb.org/`.

(and which is accessed via various URLs starting with `api.themoviedb.org/`). In this assignment, you must use that API to implement a React app in JavaScript that fetches and displays information about and images of persons.

Task 1. Sign up.

Go to `https://www.themoviedb.org/` and sign up.

Task 2. Register an API key.

Go to `https://www.themoviedb.org/settings/api/request/` and request an API key.

(a) Select “Developer”.

(b) When asked, select “Education” and fill out fields.

You can see your keys at `https://www.themoviedb.org/settings/api`. We will use the API Key, not the API Read Access Token.

Let K be your API key and let Q be a query for a person (for example a name or part of a name of a person). Then the URL

`https://api.themoviedb.org/3/search/person?query=`Q`&api_key=`K

returns a list of persons matching the query. This is documented at `https://developer.themoviedb.org/reference/search-person`, where you can also execute sample queries and see their results.

For example, with Q equal to `spielberg`, we get a response in JSON format containing something like

```
{
  "page": 1,
  "results": [
    {
      "adult": false,
      "gender": 2,
      "id": 488,
      "known_for_department": "Directing",
      "name": "Steven Spielberg",
      "original_name": "Steven Spielberg",
      "popularity": 32.776,
      "profile_path": "/tZxcg19YQ3e8fJ0pOs7hjlnmmr6.jpg",
      "known_for": [
        {
          "backdrop_path": "/mzFjAVLdZAD6UDT5BMRItHL5IVf.jpg",
          "id": 329,
          "title": "Jurassic Park",
          "original_title": "Jurassic Park",
          "overview": "A wealthy entrepreneur secretly creates …",
          "poster_path": "/oU7Oq2kFAAlGqbU4VoAE36g4hoI.jpg",
          "media_type": "movie",
          "adult": false,
          "original_language": "en",
          "genre_ids": [12, 878],
          "popularity": 41.116,
          "release_date": "1993-06-11",
          "video": false,
          "vote_average": 8,
          "vote_count": 16218
        },
        ... and more
      ]
    },
    ... and more
  ]
}
```


Let us call a single element from the `results` list a *person object.* (When searching for `spielberg`, we get an array of 20 such person objects.)

Task 3. Fetch persons.

Implement a component that performs such a search for *a hardcoded* string constant, `QUERY`, once, when the component starts (mounts). Let's call this the *main* component (but you may simply implement this in the top-level component `App`). For testing, give the string `QUERY` a value such that the request returns several

results, for example `"spielberg"`.

[Hints: see slide 31 from Session 01. To test that the call to fetch works, log the result to the console.]

Task 4. Implement a component for persons.

Implement a component `Person` that receives a person object as a prop `person` and that renders (at least) the fields `name` and `known_for_department` for that person.

To test the system, let the main component render all results (i.e., all persons that matched the query) using `Person` components.

[Hint: see slide 19 from Session 01.]

Task 5. Add buttons controlling which person to render.

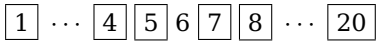
Modify the main component so that it shows only one element from the `results` array, but gives the user the ability to chose which, using one or more buttons. You decide how this is done, but here are some options: (You should implement only one of these, or something similar if you have a better idea.)

(a) Make one button for each entry in the `results` array. When clicking on the Nth button, person N should be displayed. (This is an easy way to solve this task.)

(b) Make two buttons: one for going back through the list of persons, and one for going forward. Make sure that we cannot go off the ends (which would result in an array out of bounds error).

(c) Make buttons for entries "near" the one currently being displayed. For example, when displaying the person with index N, show buttons for jumping to persons N–2, N–1, N+1, and N+2. Avoid making buttons for non-existing entries in the array, and don't make a button for the person currently being displayed.

If you also always show buttons for the first and the last persons (and display some ellipses "$\cdots$" when needed to indicate missing indices), then the result is similar to well-known web shop pagination systems (except that each "page" shows only one element). Such an arrangement of buttons may look as follows (where we are currently seeing person 6).



(This is a slightly more challenging solution, but it helps if you think of it as a variant of (a) where some buttons are not displyed.)

[Hint: No matter how you solve this task, you need a state containing the index of the entry to display and you need one or more `<button>`s with `onClick` handlers for updating or setting this state. See slides 20 and 28 from Session 01 and

remember that to get the element at index i from array a in JavaScript, write $a[i]$.]

Task 6. Render "known for"s.

Each person has a short list, `known_for`, of movies that this person is known for. Make a component that displays (at least) `title`, `release_date`, and `overview` of a `known_for` entry. Use this component to display the list of movies that a person is known for when rendering that person.

The `id` field of a person can be used to look up additional information about that person. We will use this to get additional images of a person, as follows. [See `https://developer.themoviedb.org/reference/person-images`, where you can also run a query and see the result.]

Let *ID* be the `id` of a person and *K* your API key. Then the URL

`https://api.themoviedb.org/3/person/`*ID*`/images?api_key=`*K*

returns a list of *profiles* for that person. For example, with id 488 (Steven Spielberg) we get something like

```
{
  "id": 488,
  "profiles": [
    {
      "aspect_ratio": 0.667,
      "height": 898,
      "iso_639_1": null,
      "file_path": "/tZxcg19YQ3e8fJ0pOs7hjlnmmr6.jpg",
      "vote_average": 6.058,
      "vote_count": 59,
      "width": 599
    },
    ... and more
  ]
}
```

The `profiles` array contains information about images. The `file_path` field holds a path to an image. (There are 17 profiles associated with Steven Spielberg. The paths are different, but some images are identical.) The complete URL of that image consists of (1) a `base_url` (which is "`https://image.tmdb.org/t/p/`"), (2) a `file_size` (for example, "`w45`"), and (3) the `file_path`, concatenated in that order.

[See `https://developer.themoviedb.org/docs/image-basics`.]

For example, here is a URL to a small image of Steven Spielberg:

`https://image.tmdb.org/t/p/w45/tZxcg19YQ3e8fJ0pOs7hjlnmmr6.jpg`

Task 7. Implement a component that renders pictures of a person.

Implement a component `ImagesFor` that takes an id of a person as a prop, fetches a list of `profiles` for the person with that id, extracts the `file_paths` of those profiles, and renders an array of `<img>` elements using those file paths.

Render an `ImagesFor` component under the `Person` component. (Pass the correct `id`.) The result should be that the profile images for the person with that id are rendered.