

CAB230 Assignment 1, 2023

1. Introduction

This assignment requires you to develop a React-based web application to allow users to view and analyse data about movies which we have exposed via a REST API. The assignment builds upon the lecture and practical material covered in the first half of CAB230. We will talk about the REST API in detail below.

The main aims of this assignment are to:

- Allow you to build a sophisticated client web application using modern approaches
- Provide experience in querying REST APIs and presenting the results on a web page
- Provide experience with modern web technologies such as React

2. The Dataset

The movies dataset we are using is based on publicly available data from IMDB and other sources. The data describes a subset of movies released since 1990.

It is important to appreciate that you will not work with this database directly when completing this assignment. Instead, all interactions should take place via our published REST API. You will be able to fetch the movie data using HTTP GET operations. You will also need to handle a registration and login process from your React application to the API through appropriate HTTP POST requests. We will now consider the provided REST API in more detail.

3. The REST API

Accessing the documentation

The REST API is published and documented at: <http://sefdb02.qut.edu.au:3000> (see the note below about the QUT VPN if you are off-campus). The documentation on the index page was created using Swagger (<https://swagger.io/>). We will teach you more about Swagger in the coming weeks. For this assignment, you only need to read and interact with the documentation that we have created for you.

Note that this may not be a permanent location for this API – we will see how it goes, and if necessary, relocate the API to a better home if there are problems.

The Swagger Docs are your primary source for anything to do with the API during this assignment. They will be maintained more regularly than this specification, and they have executable examples. If there is any conflict in the information we provide, or if something is unclear, the Swagger Docs win.

Our REST API is hosted on a QUT server, which is behind the QUT firewall. If you are off-campus, you will need to install and use the QUT VPN to access it:

<https://qutvirtual4.qut.edu.au/group/student/it-and-printing/wi-fi-and-internet-access/accessing-resources-off-campus>

The structure of the endpoints

The API endpoints are split into two types: data and authentication.

Data endpoints

HTTP Request	Route
GET	/movies/search
GET	/movies/data/{imdbID}
GET	/people/{id}

Authentication endpoints

HTTP Request	Route
POST	/user/register
POST	/user/login
POST	/user/refresh
POST	/user/logout

The first two data endpoints are publicly accessible. However, the */people/{id}* is restricted to authenticated users. The best way to think about this is to assume some difference between “free” content and content which is only available to registered members of your website that have logged in.

Users wanting to access the members-only content must first interact with the authentication endpoints. New users may register to become members (*/user/register*) and they can then login to the system (relying on */user/login*) to gain access to the authenticated information (accessible via */people/{id}*).

We will now consider each of the endpoints in turn.

Data endpoints

All data endpoints can be accessed via HTTP GET requests. You can discover the error conditions and precise usage in the Swagger docs. Here we are primarily concerned with the types of data that are available.

/movies/search

```
{  
  "data": [  

```

```

{
  "title": "Kate & Leopold",
  "year": 2001,
  "imdbID": "tt0035423",
  "imdbRating": "6.4",
  "rottenTomatoesRating": "52",
  "metacriticRating": "44",
  "classification": "PG-13"
},
{
  "title": "The Other Side of the Wind",
  "year": 2018,
  "imdbID": "tt0069049",
  "imdbRating": "6.7",
  "rottenTomatoesRating": "83",
  "metacriticRating": "78",
  "classification": "R"
},
{
  "title": "A Tale of Springtime",
  "year": 1990,
  "imdbID": "tt0097106",
  "imdbRating": "7.1",
  "rottenTomatoesRating": "86",
  "metacriticRating": null,
  "classification": "PG"
},
...
],
"pagination": {
  "total": 12184,
  "lastPage": 122,
  "prevPage": null,
  "nextPage": 2,
  "perPage": 100,
  "currentPage": 1,
  "from": 0,
  "to": 100
}
}

```

The `/movies/search` endpoint returns a list of movies, listed in order of their `imdbID`. In the excerpt above we have edited the results to show a manageable list of three movies. In reality, searches will yield 100 results at a time. This endpoint has three optional query parameters: `title`, `year` and `page`.

/movies/data/{imdbID}

```
{
  "year": 1998,
  "runtime": 83,
  "genres": [
    "Adventure",
    "Animation",
    "Comedy"
  ],
  "country": "United States",
  "principals": [
    {
      "id": "nm0506977",
      "category": "producer",
      "name": "Bradford Lewis",
      "characters": []
    },
    {
      "id": "nm0000095",
      "category": "actor",
      "name": "Woody Allen",
      "characters": [
        "Z"
      ]
    },
    ...
  ],
  "ratings": [
    {
      "source": "Internet Movie Database",
      "value": 6.5
    },
    {
      "source": "Rotten Tomatoes",
      "value": 92
    },
    {
      "source": "Metacritic",
      "value": 72
    }
  ],
  "boxoffice": 90757863,
  "poster": "https://m.media-
amazon.com/images/M/MV5BMzMzMd1MDktODg4OC00Y2E5LTk1ZjMtNmZMzIxZGQ0ZGI3XkEyXk
FqcGdeQXVyMTQxNzMzNDI@._V1_SX300.jpg",
  "plot": "A rather neurotic ant tries to break from his totalitarian society
while trying to win the affection of the princess he loves."
}
```

The `/movies/data` endpoint has a mandatory path parameter: `imdbID`. The endpoint returns more detailed information about the queried movie. In the above excerpt we have queried `/movies/data/tt0120587`. Once again, we have edited the results to show a manageable list of two principals (the most notable people that worked on the film). In practice, this query returns 10 such principals.

`/people/{id}`

The final data endpoint, `/people/{id}`, has a mandatory ID path parameter. This person ID can be obtained from the `/movies/data/{imdbID}` endpoint. The `/people/{id}` endpoint returns information about the queried actor, including movies that they are listed as a principal of. In the excerpt below we have queried for the person with the ID of `nm0001772` (`/people/nm0001772`).

This is an authorised endpoint, and can only be accessed by logged-in users. Without authentication, the following data is returned:

```
{
  "error": true,
  "message": "Authorization header ('Bearer token') not found"
}
```

Once authenticated, more useful information is returned:

```
{
  "name": "Patrick Stewart",
  "birthYear": 1940,
  "deathYear": null,
  "roles": [
    {
      "movieName": "Star Trek: Generations",
      "movieId": "tt0111280",
      "category": "actor",
      "characters": [
        "Picard"
      ],
      "imdbRating": 6.6
    },
    {
      "movieName": "Star Trek: First Contact",
      "movieId": "tt0117731",
      "category": "actor",
      "characters": [
        "Picard"
      ],
      "imdbRating": 7.6
    }
  ]
}
```

```

    },
    {
      "movieName": "Conspiracy Theory",
      "movieId": "tt0118883",
      "category": "actor",
      "characters": [
        "Dr. Jonas"
      ],
      "imdbRating": 6.7
    },
    ...
  ]
}

```

Authentication endpoints

In Week 7 of CAB230 we will discuss authentication and JSON Web Tokens (JWTs). We will also provide a practical worksheet on client-side authentication that we anticipate will be of great assistance in completing this part of the assignment. If you are reading this specification prior to Week 7, then it is expected that most of this section won't make a lot of sense right now.

The authentication endpoints (*/user/register*, */user/login*, */user/refresh* and */user/logout*) all need to be interacted with via HTTP POST requests. The body of the first two requests should include an email and password.

If a user is successfully registered, the returned response will be:

```

{
  "message": "User created"
}

```

If a user successfully logs in, the returned response will be:

```

{
  "bearerToken": {
    "token": "<<some token here>>",
    "token_type": "Bearer",
    "expires_in": 600
  },
  "refreshToken": {
    "token": "<<some other token here>>",
    "token_type": "Refresh",
    "expires_in": 86400
  }
}

```

The key pieces of information in the successful logged in response are the token. Your application needs to store the tokens and use the bearer token to prove that the user is authorised when accessing the authenticated route.

As mentioned earlier, we will show you how to use the authenticated routes with React in Week 7. However, it is also possible to interact with these routes via the Swagger docs. We will cover this process now.

First, register a new user. Then login with the same credentials. The response from the login route will be very similar to the screenshot above, but it should now include an actual token. At the top right of the Swagger page, you should see a green Authorize button:



Click on this button and you should now see the Authorisation dialogue:

A dialog box titled 'Available authorizations' with a close button (x) in the top right corner. Inside the dialog, it shows 'bearerAuth (http, Bearer)' and a prompt 'Enter JWT Bearer token only'. Below this, there is a label 'Value:' followed by a text input field containing 'JWTHere'. At the bottom of the dialog, there are two buttons: a green 'Authorize' button and a grey 'Close' button.

Copy the token (the contents of the 'token' field from the 'bearerToken' object) from your login response and paste it where I have "JWTHere" in the image. Click on the green Authorize button and close the dialogue. You should now see that the Authorize button lock is shut:



You are now able to use the authenticated route in Swagger.

For brevity we haven't included the possible error responses for the authenticated routes in this specification. However, they are all detailed in the Swagger docs.

4. Application Breakdown

This assignment requires you to make a lot of choices. These decisions are more obvious after you have created a few apps, but they can be hard if you are doing this for the first time. At the most basic level, you must develop a React application that allows the user to interact with all the available endpoints, but without the user ever being aware of the underlying calls to the API.

In this section we will give you some guidance on what we want to see. You should think of this assignment as a series of increasingly challenging steps, with each step allowing you the opportunity to achieve a higher grade level. You should think very carefully about how users will input things into your app, the components that you will use to display the API data, and how users will navigate between pages.

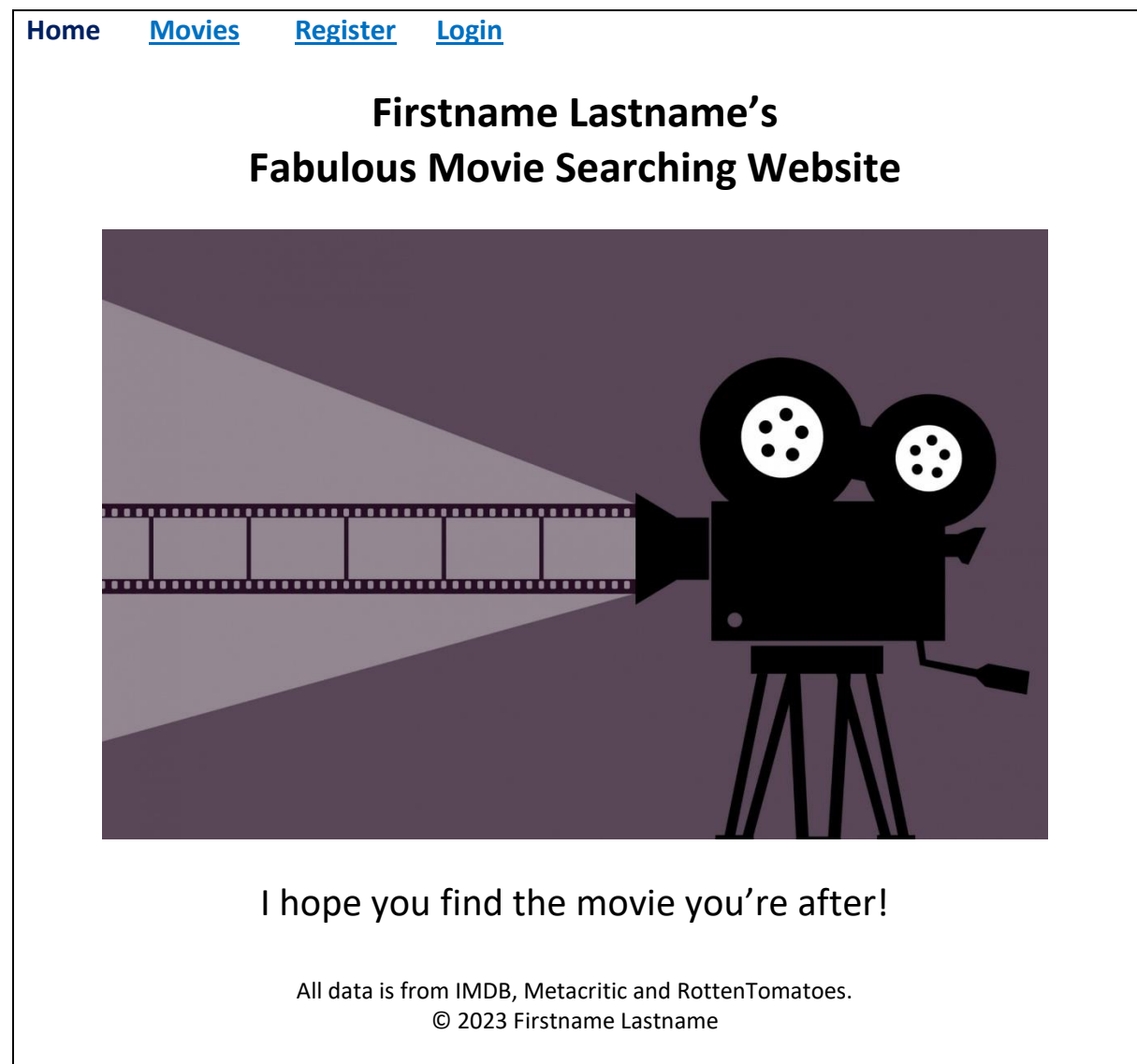
There is some latitude in all of this, and we have purposely left the rest of this section somewhat open ended. The screenshots that we show below are of an example application with almost no styling or consideration to design. We hope that it will serve as a very useful starting point, but it is not intended to be a model solution and nor do we wish for every student's assignment to look and function identically – your personal take is warmly encouraged.

If at any point you are wondering whether you are on the right track, consider whether your application utilises all the available endpoints and works with a range of components. If it does, you are likely heading in the right direction. Talk to us in the practicals if you are unsure. Towards the end of this specification, we will discuss the grade levels for this assignment in greater detail.

Landing Page

What will your users see when they first launch your app? Often there will be a hero image (just find some public domain image online and use it) and a welcome message. The navigation choices available to the user should be clear. What can they access in your app and how? Your layout should be chosen to facilitate the information that is available at the endpoints.


Something approximately like (and this does not mean you should make your page look like this- make it look much nicer!):



Movies Page

The Movie page utilises data available from the `/movies/search` endpoint. The purpose of this page is to allow the user to search for films with a text search of the film name and/or by the year the film came out. There are multiple ways you could go about doing this: for example, you could initially just show the full list of films* and provide a search bar and maybe a dropdown for the year (the dataset only has films between 1990 and 2023, so that gives you a range to start with.)

[Home](#) [Movies](#) [Register](#) [Login](#)

Movies containing the text from 

Title	Year	IMDB rating	RottenTomatoes	Metacritic	Rated
Star Wars: Episode I - The Phantom Menace	1999	6.5	51	51	PG
Star Wars: Episode II - Attack of the Clones	2002	6.6	65	54	PG
Star Wars: Episode III - Revenge of the Sith	2005	7.6	79	68	PG-13
Star Wars: The Clone Wars	2008	5.9	18	35	PG
Star Wars: Episode VII - The Force Awakens	2015	7.8	93	80	PG-13
Star Wars: Episode VIII - The Last Jedi	2017	6.9	91	84	PG-13
Star Wars: Episode IX - The Rise of Skywalker	2019	6.5	52	53	PG-13
Rogue One: A Star Wars Story	2016	7.8	84	65	PG-13
Solo: A Star Wars Story	2018	6.9	69	62	PG-13

Showing 1-9 of 9 results << 1 >>

All data is from IMDB, Metacritic and RottenTomatoes.
© 2023 Firstname Lastname

You should also provide sorting and filtering support, as many of these columns contain useful things a user may want to sort/filter by.

You also need to decide what to do if the number of films being returned is more than 100, and therefore multiple requests need to be made to retrieve all of them. Unless you are aiming for a 7, it is acceptable to simply show the first 100 results (though your interface should make clear that there are more and you are merely showing the first 100). If you are aiming for a 7, please note that the API will not react kindly if you simply send multiple requests in order to retrieve all of the results at once and you will quickly find yourself rate-limited. In this case you will need to work out something else to do that is reasonable.

You will also need to decide how the user will select an individual movie. For example, can you just click on a movie to see more information about it?

Individual Movie Page

The individual movie page utilises data from the `/movies/data/{imdbID}` endpoint. There is quite a bit of information here, including a list of prominent people that worked on the film, as well as a link to a poster that you should be able to make use of:

Star Wars: Episode I - The Phantom Menace

Released in: 1999

Runtime: 136 minutes

Genres: **Action** **Adventure** **Fantasy**

Country:  United States

Box Office: \$474,544,677

Two Jedi escape a hostile blockade to find allies and come across a young boy who may bring balance to the Force, but the long dormant Sith resurface to claim their original glory.



Role	Name	Characters
Editor	Paul Martin Smith	
Actor	Ewan McGregor	Obi-Wan Kenobi
Actor	Liam Neeson	Qui-Gon Jinn
Actress	Natalie Portman	Queen Amidala, Padmé
Actor	Jake Lloyd	Anakin Skywalker
Director	George Lucas	
Producer	Rick McCallum	
Composer	John Williams	
Cinematographer	David Tattersall	
Editor	Ben Burtt	

Internet Movie Database:
6.5/10

Rotten Tomatoes:
51%

Metacritic:
51/100

All data is from IMDB, Metacritic and RottenTomatoes.

© 2023 Firstname Lastname

Individual Person Page

This page utilises data from the `/people/{id}` endpoint to show information about a particular person, including their birth year (if known) and death year (if applicable), as well as what movies they've worked on.

Ewan McGregor

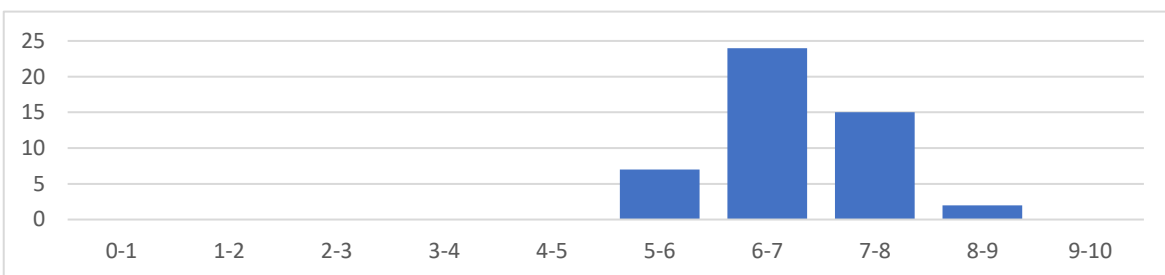
1971—

Role	Movie	Characters	Rating
Actor	Shallow Grave	Alex Law	7.3
Actor	The Pillow Book	Jerome	6.5
Actor	Brassed Off	Andy	7.2
Actor	Trainspotting	Renton	8.1
Actor	A Life Less Ordinary	Robert	6.3
Actor	Nightwatch	Martin Bells	6.2
Actor	Eye of the Beholder	Eye	5
Actor	Velvet Goldmine	Curt Wild	6.9
Actor	Star Wars: Episode I - The Phantom Menace	Obi-Wan Kenobi	6.5
Actor	Star Wars: Episode II - Attack of the Clones	Obi-Wan Kenobi	6.6

Showing 1-10 of 48 results

<< [1](#) [2](#) [3](#) [4](#) [5](#) >>

IMDB ratings at a glance



All data is from IMDB, Metacritic and RottenTomatoes.

© 2023 Firstname Lastname

If the user is not authenticated, you should display a message here telling them to log in (maybe giving a helpful link to the login page instead). You might also just not have any links to this page from the movie page if the user is not logged in – but keep in mind that you need to account for users that directly type a URL in.

If you are aiming for a 5 of greater you will want to include a chart component here (using chart.js or similar.) In this example I am just counting the number of IMDB ratings in each range and plotting a vertical bar chart (so users can get at a glance an idea of how highly rated this user's film career has been), but you may find something else that is useful to do with the data that can be plotted in such a way.

Register and Login pages

[Home](#) [Movies](#) [Register](#) [Login](#)

Registration

Email:

Password:

Confirm password:

Submit

All data is from IMDB, Metacritic and RottenTomatoes.
© 2023 Firstname Lastname

[Home](#) [Movies](#) [Register](#) [Login](#)

Login

Email:

Password:

Login

All data is from IMDB, Metacritic and RottenTomatoes.
© 2023 Firstname Lastname

The login and register pages should utilise the two POST authentication endpoints: `/user/login` and `/user/register`. You can combine the login and register pages into one page should you wish. You will need to build a form component to allow the user to enter their email and password. You should also consider and appropriately handle any error cases - e.g., email is already in use, password is missing etc. See the Swagger docs for the full list of possible error responses. Also, make sure that your password field is of the correct type so the password is hidden from shoulder-surfers.

If you are aiming for a 5 or higher, make sure you handle when the authentication token expires. The ones we generate only last 10 minutes so this will be happening a lot*. In that case, you can either tell the user that they need to log in again (redirecting them to the login page as well) or, if you are aiming for a 6 or 7, you will want to make use of the `/user/refresh` endpoint and pass in the refresh token you got when you logged in. Then make sure you store both the new bearer and new refresh tokens. If you are unable to use the refresh token (possibly because it has expired as well) then you will have to tell the user that they need to log in again.

*The `/user/login` field has a Boolean parameter it can take, `longExpiry`, which can be used for testing purposes and gives you tokens that last a year if set to true. However, the app you finally submit may not use these if you are aiming for above a 4 in the functionality criterion (as you need to implement session handling properly for a 5 and that includes handling token expiry).

You will also need some means for the user to log out. You could reuse the login page and just replace the text and dialog with a button allowing the user to log out, for example. When the user logs out, you must (if you are aiming for a 5) delete the bearer token from local storage. If you are aiming for a 6 or 7 you must additionally pass the refresh token to the `/user/logout` endpoint before deleting it from local storage as well.

Finally, keep in mind the navigation bar. If the user is logged in, instead of showing 'Register' and 'Login' buttons, show the 'Logout' button as well as the user's email address (you can store this in local storage, or decode it from the JWT.) See the 'Individual Person Page' mockup for an example of the navigation bar in an alternate state. In order to do this, you will need to keep track of this login information at a sufficiently high place in your React component hierarchy – you may consider using *context* to assist in this.

5. Some Guidance on Website Design

CSS

We anticipate most students will use a component library that comes with some included styling, such as Reactstrap. Utilising a component library will allow you to design a clean and modern website without having to write much CSS. If you have prior experience with CSS, then you may wish to write most of the styling yourself – this is an acceptable approach too.

Routing

You should think about how you are going to handle each route in your client-side application. Have a look at the React Router examples in the lectures and worksheets. You may use a basic HTML menu as a last resort (if aiming for a 4), but the use of React Router will attract much more credit (and is necessary for a 5 or higher).

Choice of Inputs/Widgets

Some of the endpoints need parameters and you will need to choose how to specify them. Have a look at the example screenshots above and their associated commentary. Are you going to use drop downs? Text boxes? Radio buttons? What makes sense for the choices you are making? Are you making consistent choices across the app? Draw some screens on paper or in a drawing tool before you start any programming.

Table Components

As discussed in the Application Breakdown section, the table component plays a crucial role in displaying the data from the server. Using a sophisticated table component like AG Grid means that you can offer rich functionality on the client-side (e.g., filtering, sorting, and pagination). Pay particular attention to the practical worksheet which walks you through this material.

Charting

To display the population density data, a bar chart or some other form of visualisation will be very well received. You may use standard chart libraries to produce these. We recommend the use of chartJS (<https://www.chartjs.org/>), especially via the widely used React wrapper which you can find here (<https://www.npmjs.com/package/react-chartjs-2>). d3 (<https://d3js.org/>) is also a popular choice, but it does have a steep learning curve and we therefore don't recommend using it unless you have prior experience.

6. The Report

We require a report and user guide to be submitted. For this submission, we have provided a template .docx file on Canvas – use this as a base to create your report. There are no marks for creativity here – we just want it filled in appropriately. Mostly this is just to help us better understand your application, but we will also require you to assess critically the quality of your UI design, and to analyse it from the perspective of accessibility, and the changes that would be needed for the application to be compliant. We give more guidance on this in the report template which sits alongside this specification on Canvas. However, to make one thing clear, you can be as brutal and self-critical as you like in these sections without noticeably affecting your actual UI marks. In fact, this is very much encouraged. The report will also help us get your thoughts on the process, the bits that worked and the bits that didn't, and the usability and correctness of the application. The report will include:

1. Introduction – telling us what was implemented and what wasn't, showing a few screenshots to illustrate the functionality.
2. Use of Endpoints. How you used the API endpoints and their relationship to the functionality described.
3. A list of any external components used, such as AG Grid.
4. Application Design and Usability. This section is a discussion of the choices you made in designing the app, and some frank assessments of its usability and the changes that would be needed to comply with accessibility requirements. This section attracts a lot of the marks allocated for the report and is noted separately in the CRA.
5. Technical Description of the Application. The architecture, test plan, results and technical limitations and compromises. We do not expect automated testing. We do expect you to verify that the application works for each of the use cases. Screen shots are your friend. This section is especially important if something doesn't actually work.

6. References
7. Appendix: brief user guide

The report should be saved as a PDF and submitted with your assignment (see submission details below).

7. Grading

The marking for this assignment will be governed by the CRA rubric, and this will cover several aspects of the assignment process. These include:

1. The overall level of functionality successfully implemented
2. The usability of the application
3. The robustness of the application
4. Evidence of a professional approach to design
5. Evidence of a professional approach to development and code quality
6. The quality of the professional report – including your assessment of your UI design and analysis of changes needed for accessibility (see report section above).

Full details of this split will be found in the separate CRA document accompanying this specification on Canvas. Here we are concerned only with the marks for functionality, application usability and robustness, and UI design. The precise marks awarded may be reduced because of features which are only partially implemented or error-prone or components which are poorly chosen and so on. As a guide, these are our expectations at each grade level:

- **[Grade of 4 level]:** A simple React app with limited styling which uses the data endpoints and presents the data cleanly. Some basic navigation between pages is required. User endpoints and the authenticated data query may not have been implemented successfully. An attempt should be made at implementing a table component, but there may be limited use of pagination, and the user may be unable to filter or sort the data on the client side. A react-strap table component or other simple alternative would suffice here.
- **[Grade of 5 level]:** At this level we would expect successful implementation of the user registration and login endpoints, and the authenticated data endpoint. Table components should utilise the standard functionality provided by a component such as AG Grid, and there should not be excessive querying of the server. The design of the website should be clean and generally uncluttered, although some parts of the website may be clumsy or confusing to use. An attempt may have been made at implementing a chart component, but shortcomings are likely to be present.
- **[Grade of 6]:** Here the expectation is that you have exceeded the grade of 5 level in that all the basics are there and working smoothly. Navigation is handled using React Router, controlled forms are used for inputs and there is evidence of close alignment between your chosen components and the data they are displaying. You are making use of refresh tokens so the user does not need to log in again every 10 minutes. The overall design of the website should be well thought out and it should be easy to use. We would also expect at this level for you to have used a chart component (or similar visualisation) on your 'person' page to show the IMDB scores of the films they worked on.

- **[Grade of 7]:** Our expectations for a grade of 7 are that you have covered everything needed for a grade of 6, and that you have gone beyond that, making full use in exposing the capabilities of the endpoints to users (e.g. with ag-grid infinite scrolling to handle the paginated search data.)

A reminder that the mark levels are based on *successful* implementation of the features mentioned. If they don't work or are substandard, then of course the grade level may be reduced.

8. Submission

Your submission will include the following components:

Code

The code archive should be based on the React application structure that you inherit from create-react-app. We will not be impressed if this has been disrupted badly or there are additional folders with a seemingly random purpose and organisation. You will need to **zip up your code archive and submit it to Canvas.**

Above all, we will need you to pay close attention to your *node_modules*:

Node apps involve installation of packages, and this leads to the installation of *node_modules*, and eventually these take up rather a lot of space. Please delete them. And then look around through the directory structures again and delete any others that you missed the first time. We don't want to see them, we don't want to store them, and you don't want to wait while they upload.

And in case you missed it the first time, please delete your *node_modules*.

Report

The report will be as discussed above and as specified in more detail in the report template. We will get you to submit it as a PDF along with your code (not in the zip, but as a separate file.)