IMY 220
Assignment 5

Async & React

General Instructions

- This assignment must be completed and submitted by the due date which is available on ClickUP.
- This assignment is a take-home assignment and may take one or two days to complete.
- This assignment should be completed on your own, but you may come to the tutorial session or email the assistant lecturer if you need further assistance.
- No late / email submissions will be accepted.

This assignment focuses on **AJAX, ES6 Promises and Async in React**. You will be required to create a simple React application that queries the Star Wars API and updates the components based on user input.

Note**: For this assignment you are required to set up a React project using webpack and babel as is taught in the lecture videos. You must make use of the correct file structure, and each component should be in a **separate file** in the correct folder as taught in the lecture videos. **You may not use a builder such as Create-react-app or Vite. Doing so will result in -10% from your assignment mark.

Part 1 - React App With Webpack and Babel

Create a **simple React App using webpack and babel** as taught in the lecture 19 videos (NodeJs & React).

You are more than welcome to copy and paste in a different project that you have already set up from another practical, assignment or your semester project, just make sure that you have a **working project**, and that you **only** have what is required from you for this assignment (i.e., please remove any extra files when copying over).

You do not have to set up the folder structure like you need to for your project (i.e., with `frontend` and `backend` folders, etc.) but you may do so if you like.

The rest of your assignment depends on this step working, so make sure you have set everything up correctly before moving on to the next step.

Part 2 - API Requests

In a file called ***api.js*** (this can be in the **root** of your project). Export two functions that query the Star Wars API (SWAPI): https://swapi.dev/ (Read their documentation on how to structure each query). Keep in mind these functions will be **asynchronous**.

1. The first function should get **a character** based on an **id** passed in.
2. The second should query the SWAPI for a **specific** character by **name**.

For each of the queries you should do adequate **error handling**. For example, if the API request fails or if the response returned is empty, you should return and display appropriate and informative error messages to the user (not just in the console).

The API requests should be made by using one of the methods taught in the lecture videos and slides which return a **Promise**.

Note: *You should not get a CORS error upon using the SWAPI. If you do, something else in your project is likely incorrect.*

Part 3 - Person Component

Create a Person class component (***Person.js***) that displays the results of the API request (passed into the component) as follows:

Person

R2-D2

Birth Year: 33BBY

Eye Color: red

Gender: n/a

Homeworld: https://swapi.dev/api/planets/8/

You do not need to include more of the response data you get from the SWAPI, but you may do so if you like. You must include at least the data above.

Part 4 - Search Component

Create a Search class component (***Search.js***) that handles the user input for a search. The user can search for a specific Star Wars character (person) by name. The search should **not execute** if the input is empty.

The search should look something like this:

You should use the method for grabbing values from HTML inputs, and passing out search output (i.e., passing the search value to the parent component) as discussed in previous assignments, and in the lecture videos/slides. This component should not make API calls, but should just handle user input of a search value that is then passed to the parent component.

Part 5 - StarWars Component

Create a StarWars class component (***StarWars.js***). This component should act as the parent component for the *Person* and *Search* components, and should handle making requests to the SWAPI using the two functions you defined in ***api.js*** above.

This component should also include two **buttons**, a **previous** and a **next** button, that will fetch the previous/next Star Wars character from the API respectively and display it using the *Person* component. To do this, you will need to track and update the component state while handling the async SWAPI queries from within the **StarWars** component.

For this, you will need to make use of the methods for handling asynchrony in components as taught in the lecture videos. **Having an infinite loop / any React synchronisation errors will result in zero for this section, as you are handling async updates to the component state incorrectly**.

1. The component should initially query the API for a character with an id of **1** (using the first function you defined)
2. When the user clicks the **next** button, SWAPI should be queried for a character with an id of **2**. If the user clicks the button again, SWAPI should be queried for a character with an id of **3**, and so on.
3. The same thing happens when the user presses the **previous** button, except that if the *last* query was for a character with an id of **3**, SWAPI should be queried for a character with an id of **2** (i.e., moving backwards).

4. You will need to handle SWAPI queries with ids <=0 as the API will bring back nothing / an error. For example, disabling the button or showing an error.

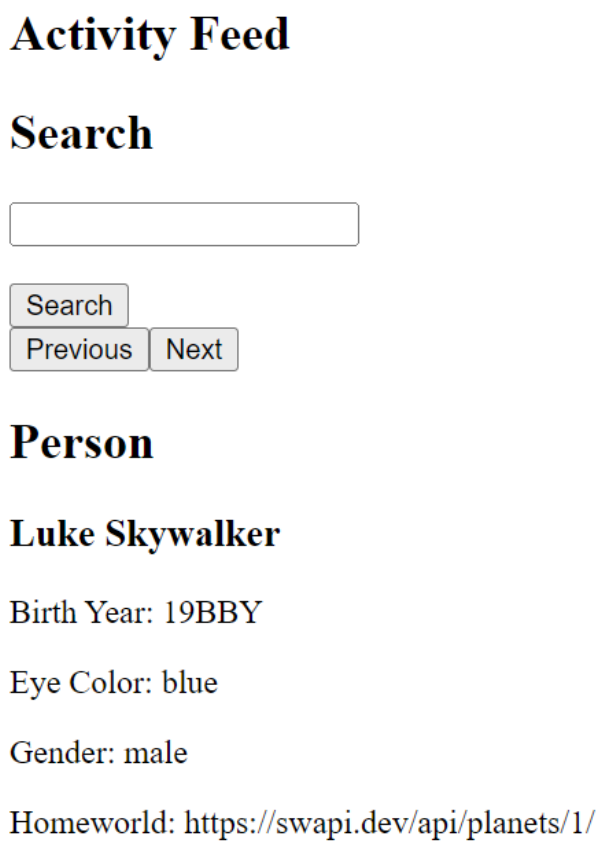
When a user **searches** for a specific Star Wars character, a **search** query to SWAPI should be made (using the second function you defined) and the Person component updated with the result of the search. The page should not be reloaded.

Hint: For searching, you can update the component regularly without requiring using a lifecycle method.

Remember to display appropriate error messages to the user as described in Part 1.

Render this component in the "**#root**" div in the browser. This should be the root of your application.

The component should look something like this (when the page has initially loaded):



If the user presses the next button…

…once:

Activity Feed

Search

Search

Previous Next

Person

C-3PO

Birth Year: 112BBY

Eye Color: yellow

Gender: n/a

Homeworld: https://swapi.dev/api/planets/1/

…twice:

Activity Feed

Search

Search

Previous Next

Person

R2-D2

Birth Year: 33BBY

Eye Color: red

Gender: n/a

Homeworld: https://swapi.dev/api/planets/8/

If the user presses the previous button (without reloading the page):

Activity Feed

Search

Search
Previous Next

Person

C-3PO

Birth Year: 112BBY

Eye Color: yellow

Gender: n/a

Homeworld: https://swapi.dev/api/planets/1/

If the user searches for Leia:

Activity Feed

Search

Search
Previous Next

Person

Leia Organa

Birth Year: 19BBY

Eye Color: brown

Gender: female

Homeworld: https://swapi.dev/api/planets/2/

Bonus: Tailwind Styling

As a bonus, import tailwind into your project and style your different Person, Search and StarWars components using tailwind classes. ***Note*** *that in React, the HTML class is replaced with* ***className***.

The design should still be readable / usable. You should style the text (font-weight, font-colour, etc.), padding / margin, background colour and anything else that seems appropriate. You may make use of the default tailwind fonts, colours, etc. **No** default HTML/CSS styling (like what is in the screenshots above) may be visible to get the marks.

Submission Instructions

Place these in a folder named `A5_u12345678` where `12345678` is your student number.

- Submit *all* the files required for this assignment to work (including the config files for babel and webpack) **except** for your node_modules folder and its contents.
- **Do not submit your node_modules folder. This will result in -10% from your assignment mark.**

Zip the **folder** and upload this to ClickUP in the relevant submission slot before the deadline.