Error Messages

June 21st 2023

OBJECTIVE

The users should be able to see personalized error messages when inserting unallowed characters in an input element and / or when setting less than one track for a playlist. A fallback for API failures should also be provided in case Spotify is down for a couple of hours or even more time.

BACKGROUND

This functionality is important because it avoids erroneous API calls (for example: calling /playlists/{playlist_id}/tracks with no tracks at all) and informs the user of what's wrong in a friendly way, if by any means the API fails to correctly respond to the user, then an error message is provided both for the "Save Playlist" functionality as well as the "Search" functionality.

If we look at any modern-day web form we will find customized error messages for everything: from unsafe passwords to already-in-use usernames. For instance, as quoted from HB Design's article "The Importance of Error Messages": '

Error messages can easily be overlooked when a page or web-based tool is being created. Sometimes the placeholders for error messages are still there when a page or tool has been published—this can lead to confusion. Other times, the message appears at the wrong time or with the wrong information.

[…]

At the Delight Conference in Portland, Jonathan Colman of Facebook suggested that error messages should explain simply and clearly:

1. What just happened

2. Where the problem happened (did the user do something, or did the tool do something)

3. What to do next and who should do it'

TECHNICAL DESIGN

This section should lay out all of the information needed to implement this feature. In the context of React, this could include new components and their functionality, existing component updates, how to address edge cases, and any other information an engineer would need before implementing this feature.

For front-end changes, this section will often include design mocks or wireframes to specify how the design of the application needs to be updated.

This feature should be enabled in 3 different components: SearchBar (display typing errors in input), SearchResults (provide fallback for API errors) and Playlist (display typing errors, not allowing the user to save a playlist with no tracks and providing fallback for API errors).

For the implementation of these features we decided to use the useState hook, React Event Handlers, React props and the HTMLObjectElement: setCustomValidity method to display errors in a more semantic way, we also used CSS pseudo elements for easier styling.
Below, all implementations for the feature:

App

The app components is responsible for passing its properties (error and setError from its state and the method checkInvalidCharacters, which returns true if any invalid characters such as ` or ç are present in its value parameter).

```jsx
import SearchBar from './components/searchBar/SearchBar';
import SearchResults from './components/searchResults/searchResults';
import Playlist from './components/playlist/Playlist';
import React, { useState } from 'react';

export const context = React.createContext()    Fast refresh only works when a file only export components. Use a

function App() {
  const [playlistTracks, setPlaylistTracks] = useState([]);
  const [tracks, setTracks] = useState([]);
  const [error, setError] = useState(false);

  const checkInvalidCharacters = value => !/^[a-zA-Z\s]+$/.test(value);

  return (
    <main>
      <context.Provider value={[tracks, playlistTracks, setPlaylistTracks]}>
        <SearchBar setTracks={setTracks} setError={setError} checkInvalidCharacters={checkInvalidCharacters} />
        <SearchResults error={error} />
        <Playlist checkInvalidCharacters={checkInvalidCharacters} />
      </context.Provider>
    </main>
  );
}

export default App
```

SearchBar

The SearchBar component receives setError and checkInvalidCharacters as its props.

In the handleSubmit method we check if the string is empty or its just whitespace (with the help of the String.prototype.trim function) and set the validity of Event.target.children[0] to false by passing a non-empty string to the setCustomValidity method, after this we return from the function. The next time the user searches for something that passes the previous check, we will make sure that the input is set as valid again (using setCustomValidity with an empty string).

We also prevent a bug where the handleSubmit method won't be called after setting validity to false by setting the <button type="submit"> boolean attribute formNoValidate in our return statement.

Later on the function checks in results.tracks exists and if its length is more than 0, in that case it calls the setError function with false as its parameter and returns, among other things If one of these conditions isn't met then the setError function is called with true as its parameter, when we go at the searchResults component we will take a look at what this piece of state does.

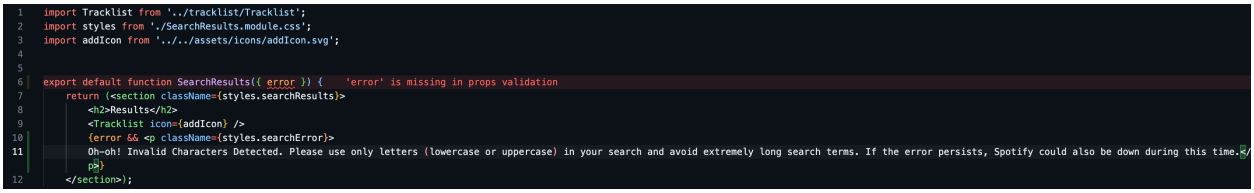
```
import { authenticate, refreshToken, searchTracks } from '../../util/Spotify'
import styles from './SearchBar.module.css';
import { useState } from 'react';


export default function SearchBar({ setTracks, setError, checkInvalidCharacters }) {    'setTracks' is missing in props validation
  const [searchValue, setSearchValue] = useState('');
  const handleChange = ({target: {value}}) => {
    if (checkInvalidCharacters(value) && value.trim() !== '') {
      return;
    }
    setSearchValue(value);
  }
  const handleSubmit = async e => {
    e.preventDefault();
    const formChildren = e.target.children;
    formChildren[1].blur();
    if (searchValue.trim() === '') {
      formChildren[0].setCustomValidity(' ')
      return;
    }
    formChildren[0].setCustomValidity('')
    let accessToken = JSON.parse(localStorage.getItem('access_token'));
    if (accessToken === null) {
      accessToken = await authenticate()
    }
    else if (accessToken.expDate + 3600000 < (new Date()).getTime()) {
      accessToken = refreshToken();
    }
    const results = await searchTracks(accessToken, searchValue);
    if (results.tracks && results.tracks.items.length !== 0) {
      setTracks(results.tracks.items);
      setError(false);
      return;
    }
    setError(true);
    setTracks([])
  }

  return <form onSubmit={handleSubmit}>
    <input placeholder="Enter a song title" name="search-bar" className={styles.searchInput} value={searchValue} onChange={handleChange} />
    <button className={styles.searchButton} type="submit" formNoValidate>Search</button>
  </form>
}
```

SearchResults

The searchResults component receives error as its only property.
In its return statement it checks if error is true and renders a <p> element with modular style searchError and a helpful error message



```
import Tracklist from '../tracklist/Tracklist';
import styles from './SearchResults.module.css';
import addIcon from '../../assets/icons/addIcon.svg';


export default function SearchResults({ error }) {    'error' is missing in props validation
    return (<section className={styles.searchResults}>
        <h2>Results</h2>
        <Tracklist icon={addIcon} />
        {error && <p className={styles.searchError}>
        Oh-oh! Invalid Characters Detected. Please use only letters (lowercase or uppercase) in your search and avoid extremely long search terms. If the error persists, Spotify could also be down during this time.</
        p>}
    </section>);
```

Playlist

The Playlist component takes a different approach, relying more on CSS pseudo elements as the complexity increases meaningfully when more than just one error is possible, it also uses two references for DOM elements (nameInputRef and saveButtonRef) using React useRef() hook.

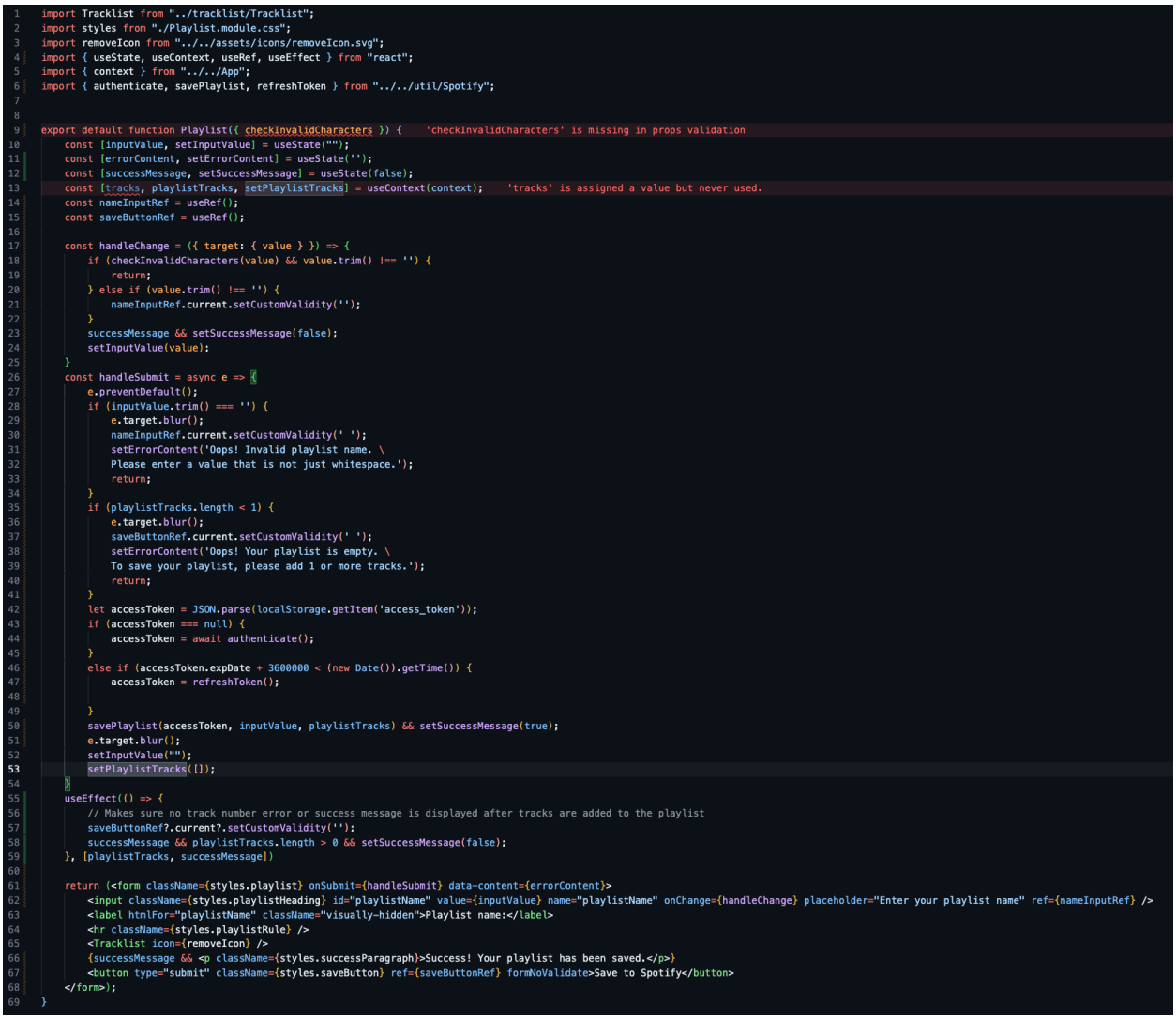
We also have an extra piece of state (errorContent and setError content) which controls the behaviour of our error message by directly setting the content property of our form:invalid:after, we did this by using the attr() function in CSS to access our custom data-content attribute, this is a very tricky method but allows us to use no extra markup.

In the handleChange method (called every time something is typed on the <input> with name="searchBar"), among other things it is checked that if we trim the value it isn't just whitespace and if it isn't we set the validity of saveButtonRef to true.

In the handleSubmit method we check for the condition that the inputValue is just whitespace and that the playlistTracks.length is less than 1 (that means the user didn't actually add any tracks to the playlist).

Using the useEffect hook (among other things) we make sure no track number error is displayed after the playlistTracks changes (usually implicating the addition of a new track).

In the component return statement we also give our <button type="submit"> a formNoValidate boolean attribute to avoid handleSubmit not being called after setting an element's validity to false.



```
import Tracklist from "../tracklist/Tracklist";
import styles from "./Playlist.module.css";
import removeIcon from "../../assets/icons/removeIcon.svg";
import { useState, useContext, useRef, useEffect } from "react";
import { context } from "../../App";
import { authenticate, savePlaylist, refreshToken } from "../../util/Spotify";


export default function Playlist({ checkInvalidCharacters }) {    'checkInvalidCharacters' is missing in props validation
    const [inputValue, setInputValue] = useState("");
    const [errorContent, setErrorContent] = useState('');
    const [successMessage, setSuccessMessage] = useState(false);
    const [tracks, playlistTracks, setPlaylistTracks] = useContext(context);    'tracks' is assigned a value but never used.
    const nameInputRef = useRef();
    const saveButtonRef = useRef();

    const handleChange = ({ target: { value } }) => {
        if (checkInvalidCharacters(value) && value.trim() !== '') {
            return;
        } else if (value.trim() !== '') {
            nameInputRef.current.setCustomValidity('');
        }
        successMessage && setSuccessMessage(false);
        setInputValue(value);
    }
    const handleSubmit = async e => {
        e.preventDefault();
        if (inputValue.trim() === '') {
            e.target.blur();
            nameInputRef.current.setCustomValidity(' ');
            setErrorContent('Oops! Invalid playlist name. \
            Please enter a value that is not just whitespace.');
            return;
        }
        if (playlistTracks.length < 1) {
            e.target.blur();
            saveButtonRef.current.setCustomValidity(' ');
            setErrorContent('Oops! Your playlist is empty. \
            To save your playlist, please add 1 or more tracks.');
            return;
        }
        let accessToken = JSON.parse(localStorage.getItem('access_token'));
        if (accessToken === null) {
            accessToken = await authenticate();
        }
        else if (accessToken.expDate + 3600000 < (new Date()).getTime()) {
            accessToken = refreshToken();

        }
        savePlaylist(accessToken, inputValue, playlistTracks) && setSuccessMessage(true);
        e.target.blur();
        setInputValue("");
        setPlaylistTracks([]);
    }
    useEffect(() => {
        // Makes sure no track number error or success message is displayed after tracks are added to the playlist
        saveButtonRef?.current?.setCustomValidity('');
        successMessage && playlistTracks.length > 0 && setSuccessMessage(false);
    }, [playlistTracks, successMessage])

    return (<form className={styles.playlist} onSubmit={handleSubmit} data-content={errorContent}>
        <input className={styles.playlistHeading} id="playlistName" value={inputValue} name="playlistName" onChange={handleChange} placeholder="Enter your playlist name" ref={nameInputRef} />
        <label htmlFor="playlistName" className="visually-hidden">Playlist name:</label>
        <hr className={styles.playlistRule} />
        <Tracklist icon={removeIcon} />
        {successMessage && <p className={styles.successParagraph}>Success! Your playlist has been saved.</p>}
        <button type="submit" className={styles.saveButton} ref={saveButtonRef} formNoValidate>Save to Spotify</button>
    </form>);
}
```

CAVEATS

In the SearchBar we use a very counter-intuitive and non-scalable way of accessing the children of our <form> element, however its performance should be better than by using references (Haven't benchmarked yet).
In the Playlist component we tried another paradigm, this time using multiple refs, which resulted in code a bit more readable and scalable, but in poorer performance (as stated before, this is only based in common sense, as no tests were performed yet).

The playlist component could be refactored to stop using CSS pseudo elements (which would probably end up with the content logic inside of the error state), this may or not be better for

performance and could make our code more readable. However it would impact negatively on our CSS codebase, making it harder to understand.