Explicit Data Fetching with React

Learn to change implementation details from implicit to explicit data (re-)fetching.

We'll cover the following

- Exercises:

Re-fetching all data each time someone types in the input field isn't optimal. Since we're using a third-party API to fetch the data, its internals are out of our reach. Eventually, we will incur rate limiting, which returns an error instead of data.

To solve this problem, change the implementation details from implicit to explicit data (re-)fetching. In other words, the application will refetch data only if someone clicks a confirmation button. First, add a button element for the confirmation to the JSX:

```
const App = () => {
  ...

  return (
    <div>
      <h1>My Hacker Stories</h1>

      <InputWithLabel
        id="search"
        value={searchTerm}
        isFocused

        onInputChange={handleSearchInput}

      >
        <strong>Search:</strong>
      </InputWithLabel>


      <button
        type="button"
        disabled={!searchTerm}
        onClick={handleSearchSubmit}
      >
        Submit
      </button>

      ...
    </div>
  );
```

```
};
```

src/App.js

Second, the handler, input, and button handler receive implementation logic to update the component's state. The input field handler still updates the `searchTerm`; the button handler sets the `url` derived from the *current* `searchTerm` and the static API URL as a new state:

```
const App = () => {
  const [searchTerm, setSearchTerm] = useSemiPersistentState(
    'search',
    'React'
  );

  const [url, setUrl] = React.useState(
    `${API_ENDPOINT}${searchTerm}`
  );

  ...
  const handleSearchInput = event => {

    setSearchTerm(event.target.value);
  };

  const handleSearchSubmit = () => {
    setUrl(`${API_ENDPOINT}${searchTerm}`);
  };

  ...
};
```

src/App.js

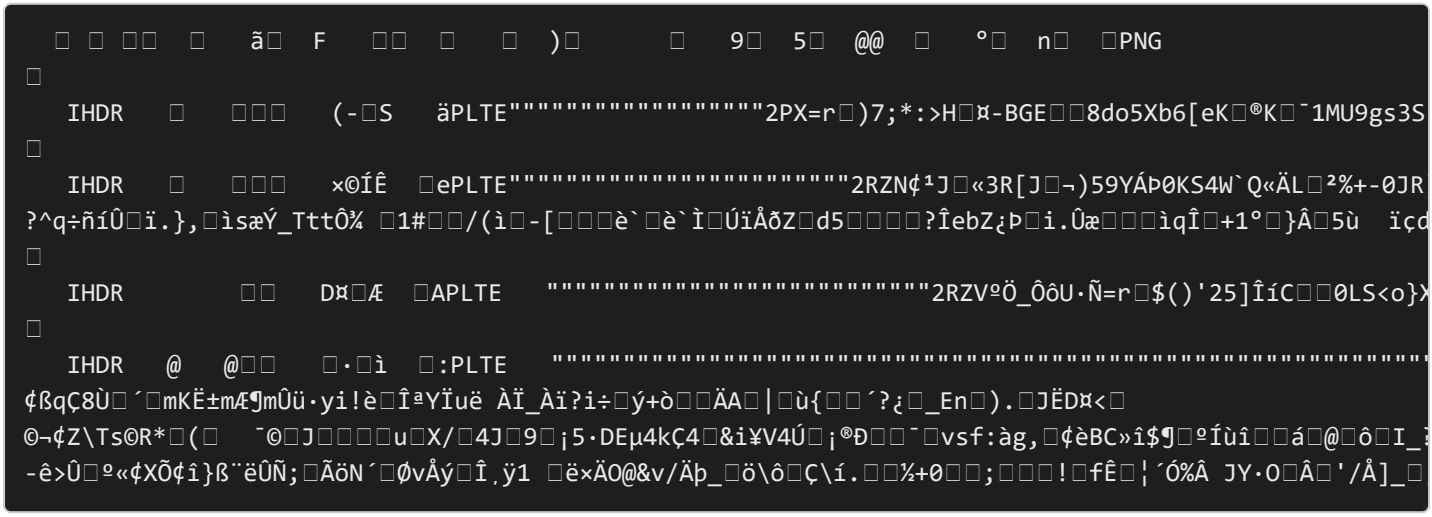
Third, instead of running the data fetching side-effect on every `searchTerm` change – which would happen each time the input field's value changes – the `url` is used. The `url` is set explicitly by the user when the search is confirmed via our new button:

```
const App = () => {
  ...

  const handleFetchStories = React.useCallback(() => {
    dispatchStories({ type: 'STORIES_FETCH_INIT' });

    fetch(url)
      .then(response => response.json())
      .then(result => {
        dispatchStories({
          type: 'STORIES_FETCH_SUCCESS',
          payload: result.hits,
        });
      })
```

```
      })
      .catch(() =>
        dispatchStories({ type: 'STORIES_FETCH_FAILURE' })
      );

  }, [url]);


  React.useEffect(() => {
    handleFetchStories();
  }, [handleFetchStories]);

  ...
};
```

src/App.js

Before the `searchTerm` was used for two cases: updating the input field's state and activating the side-effect for fetching data. Too many responsibilities one would have said. Now it's only used for the former. A second state called `url` got introduced for triggering the side-effect for fetching data which only happens when a user clicks the confirmation button.



Exercises:

- Confirm the changes from the last section.

Test your knowledge!

1 Why is `useState` instead of `useSemiPersistentState` used for the `url` state management?

2

Why is there no check for an empty `searchTerm` in the `handleFetchStories` function anymore?

Retake Quiz