React Asynchronous Data

Learn to simulate asynchronous data in the application.

We'll cover the following

- Exercises:

We have two interactions in our application: searching the list, and removing items from the list. The first interaction is a fluctuant interference through a third-party state (`searchTerm`) applied on the list; the second interaction is a non-reversible deletion of an item from the list.

Sometimes we must render a component before we can fetch data from a third-party API and display it. In the following, we will simulate this kind of asynchronous data in the application. In a live application, this asynchronous data gets fetched from a real remote API. We start off with a function that returns a promise – data, once it resolves – in its shorthand version. The resolved object holds the previous list of stories:

```
const initialStories = [ ... ];

const getAsyncStories = () =>
  Promise.resolve({ data: { stories: initialStories } });
```

src/App.js

In the App component, instead of using the `initialStories`, use an empty array for the initial state. We want to start off with an empty list of stories, and simulate fetching these stories asynchronously. In a new `useEffect` hook, call the function and resolve the returned promise. Due to the empty dependency array, the side-effect only runs once the component renders for the first time:

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

  const [stories, setStories] = React.useState([]);
  React.useEffect(() => {
    getAsyncStories().then(result => {
      setStories(result.data.stories);
```

```
    });
  }, []);

  ...
};
```

src/App.js

Even though the data should arrive asynchronously when we start the application, it appears to arrive synchronously, because it's rendered immediately. Let's change this by giving it a bit of a realistic delay because every network request to an API would come with a delay. First, remove the shorthand version for the promise:

```
const getAsyncStories = () =>
  new Promise(resolve =>
    resolve({ data: { stories: initialStories } })
  );
```

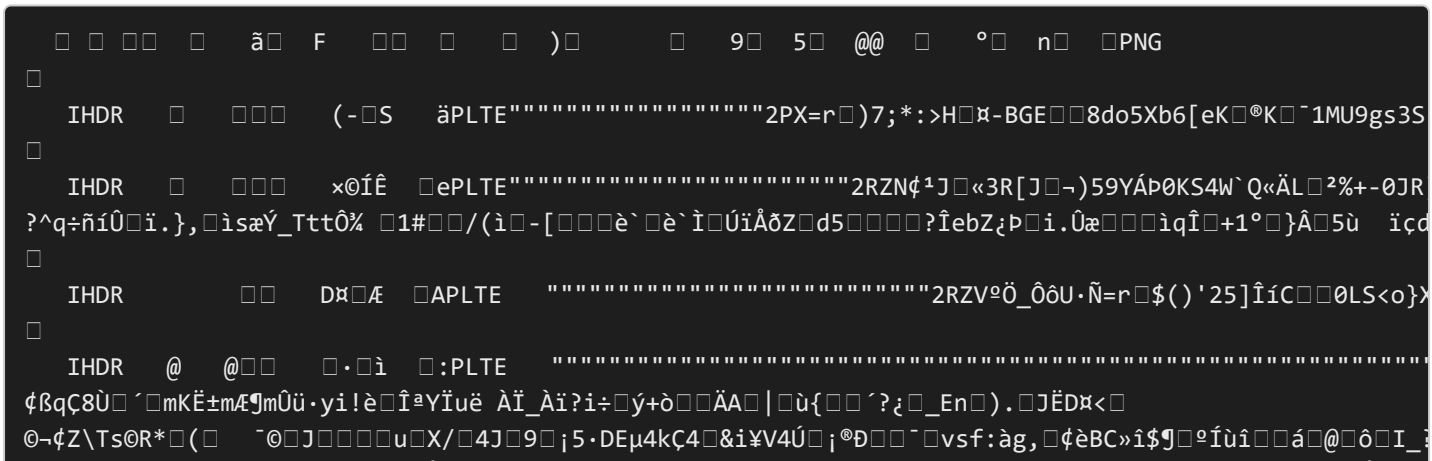
src/App.js

And second, when resolving the promise, delay it for a few seconds:

```
const getAsyncStories = () =>
  new Promise(resolve =>
    setTimeout(
      () => resolve({ data: { stories: initialStories } }),
      2000
    )
  );
```

src/App.js

Once you start the application again, you should see a delayed rendering of the list. The initial state for the stories is an empty array. After the App component rendered, the side-effect hook runs once to fetch the asynchronous data. After resolving the promise and setting the data in the component's state, the component renders again and displays the list of asynchronously loaded stories.



-ê>Û□º«¢XÕ¢î}ß¨ëÛÑ;□ÃöN´□ØvÅý□Î¸ÿ1 □ë×ÄO@&v/Äþ_□ö\ô□Ç\í.□□½+0□□;□□□!□fÊ□¦´Ó%Â JY·O□Â□'/Å]_□

Exercises:

- Confirm the changes from the last section.
- Read more about JavaScript Promises.