React Conditional Rendering

Learn about conditional states for asynchronous data.

We'll cover the following

- Exercises:

Handling asynchronous data in React leaves us with conditional states: with data and without data. This case is already covered, though, because our initial state is an empty list rather than `null`. If it was null, we'd have to handle this issue in our JSX. However, since it's `[]`, we `filter()` over an empty array for the search feature, which leaves us with an empty array. This leads to rendering nothing in the List component's `map()` function.

In a real-world application, there are more than two conditional states for asynchronous data, though. Consider showing users a loading indicator when data loading is delayed:

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

  const [stories, setStories] = React.useState([]);

  const [isLoading, setIsLoading] = React.useState(false);


  React.useEffect(() => {

    setIsLoading(true);

    getAsyncStories().then(result => {
      setStories(result.data.stories);

      setIsLoading(false);

    });
  }, []);

  ...
};
```

src/App.js

With JavaScript's ternary operator, we can inline this conditional state as a **conditional rendering** in JSX:

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

  return (
    <div>
      ...
      <hr />

      {isLoading ? (
        <p>Loading ...</p>
      ) : (

        <List
          list={searchedStories}
          onRemoveItem={handleRemoveStory}
        />

      )}
    </div>
  );
};
```

src/App.js

Asynchronous data comes with error handling, too. It doesn't happen in our simulated environment, but there could be errors if we start fetching data from another third-party API. Introduce another state for error handling and solve it in the promise's `catch()` block when resolving the promise:

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

  const [stories, setStories] = React.useState([]);
  const [isLoading, setIsLoading] = React.useState(false);
  const [isError, setIsError] = React.useState(false);

  React.useEffect(() => {
    setIsLoading(true);

    getAsyncStories()
      .then(result => {
        setStories(result.data.stories);
        setIsLoading(false);
      })

      .catch(() => setIsError(true));

  }, []);

  ...
};
```

src/App.js

Next, give the user feedback in case something went wrong with another conditional rendering. This time, it's either rendering something or nothing. So instead of having a ternary operator where one side returns `null`, use the logical `&&` operator as shorthand:

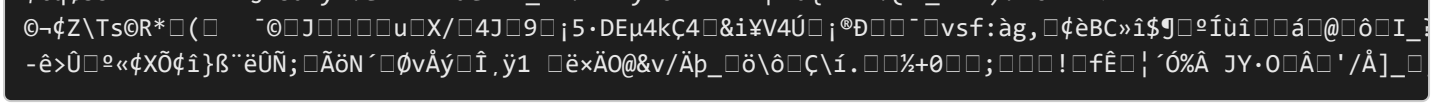
```
const App = () => {
  ...

  return (
    <div>
      ...

      <hr />

      {isError && <p>Something went wrong ...</p>}
      {isLoading ? (
        <p>Loading ...</p>
      ) : (
        ...
      )}
    </div>
  );
};
```

src/App.js

In JavaScript, a `true && 'Hello World'` always evaluates to 'Hello World'. A `false && 'Hello World'` always evaluates to false. In React, we can use this behaviour to our advantage. If the condition is true, the expression after the logical `&&` operator will be the output. If the condition is false, React ignores it and skips the expression.

Conditional rendering is not just for asynchronous data though. The simplest example of conditional rendering is a boolean flag state that's toggled with a button. If the boolean flag is true, render something, if it is false, don't render anything.


```
  □ □ □□  □   ã□  F   □□  □   □  )□     □   9□  5□  @@  □   °□  n□  □PNG
□
   IHDR   □   □□□   (-□S   äPLTE"""""""""""""""""""2PX=r□)7;*:>H□¤-BGE□□8do5Xb6[eK□®K□¯1MU9gs3S
□
   IHDR   □   □□□   ×©ÍÊ  □ePLTE""""""""""""""""""""""""2RZN¢¹J□«3R[J□¬)59YÁÞ0KS4W`Q«ÄL□²%+-0JR
?^q÷ñíÛ□ï.},□ìsæÝ_TttÔ¾ □1#□□/(ì□-[□□□è`□è`Ì□ÚïÅðZ□d5□□□□?ÎebZ¿Þ□i.Ûæ□□□ìqÎ□+1°□}Â□5ù  ïçd
□
   IHDR       □□   D¤□Æ  □APLTE   """""""""""""""""""""""""""2RZVºÖ_ÔôU·Ñ=r□$()'25]ÎíC□□0LS<o}X
□
   IHDR   @   @□□   □·□ì  □:PLTE   """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
¢ßqC8Ù□´□mKË÷mÆ¶mÛü·vi¦è□Îª¥Ïuë ÀÏ Àï?i÷□ý+ò□□ÄA□|□ù{□□´?¿□ En□).□JËD¤<□
```



This feature can be quite powerful, because it gives you the ability to conditionally render JSX. It's yet another tool in React to make your UI more dynamic. And as we've discovered, it's often necessary for more complex control flows like asynchronous data.

Exercises:

- Confirm the changes from the last section.
- Read more about conditional rendering in React.