CONTAINER COMPONENTS FROM PRESENTATIONAL COMPONENTS

Separate Container Components From Presentational Components

In this lesson, you will learn a programming pattern that will help organize your React code.

As you continue building your React application, you will soon realize that one component has too many responsibilities, but how do you know when you have reached that point?

Separating container components from presentational components helps to answer that question. It shows you when it might be a good time to divide a component into smaller components. It also shows you how to perform that division.

Instructions

1.

Click Run. You are looking at a rendered `<GuineaPigs />` component.

`<GuineaPigs />`'s job is to render a photo carousel of guinea pigs. It does this perfectly well! And yet, it has a problem: it does too much stuff. How might we divide this into a container component and a presentational component?

GuineaPig.js

```
import React from 'react';
import ReactDOM from 'react-dom';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-4.jpg'
];
```

```jsx
export class GuineaPigs extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    let src = GUINEAPATHS[this.state.currentGP];
    return (
      <div>
        <h1>Cute Guinea Pigs</h1>
        <img src={src} />
      </div>
    );
  }
}

ReactDOM.render(
  <GuineaPigs />,
  document.getElementById('app')
);
```

Create Container Component

Separating container components from presentational components is a popular React programming pattern. It is a special application of the concepts learned in the Stateless Components From Stateful Components module.

If a component has to have state, make calculations based on props, or manage any other complex logic, then that component shouldn’t also have to render HTML-like JSX.

The functional part of a component (state, calculations, etc.) can be separated into a *container component*.

Instructions

1.

GuineaPigs.js contains a lot of logic! It has to select the correct guinea pig to render, wait for the right amount of time before rendering, render an image, select the next correct guinea pig, and so on.

Let’s separate the logic from the `GuineaPigs` component into a container component.

Near the top left of the code editor, click on the folder icon. Create a new folder named **containers**. **containers/** should be next to **components/**.

Inside of **containers/**, create a new file named **GuineaPigsContainer.js**. Make sure that GuineaPigs is plural, but Container is singular!

Once you have made **containers/GuineaPigsContainer.js**, click Run.

Checkpoint 2 Passed

Hint

If you haven’t already, open the file browser by clicking the folder icon at the top-left of the code editor.

Create a new folder called **containers/** that is a sibling to **components/**.

Inside that folder, create a new file called **GuineaPigsContainer.js**. You don’t need to put anything in it yet.
2.

Good!

Open **components/GuineaPigs.js**.

You want to separate the logic of this component class into a container component. How do you do that?

To start, just make a copy. After that, you can delete the appropriate parts.

Highlight the entire contents of components/GuineaPigs.js, and copy it to the clipboard.

Now, open **containers/GuineaPigsContainer.js**.

Click inside the empty file, and paste. **containers/GuineaPigsContainer.js** and **components/GuineaPigs.js** should be identical.
Checkpoint 3 Passed

Hint

Your new file, **containers/GuineaPigsContainer.js**, should be identical to **components/GuineaPigs.js**.

The shortcuts for select-all and copy on Mac are:

- `cmd` - `A`
- `cmd` - `C`

The shortcuts for select-all and copy on Windows are:

- `ctrl` - `A`
- `ctrl` - `C`

GuineaPigs.js

```
import React from 'react';
import ReactDOM from 'react-dom';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-4.jpg'
];

class GuineaPigs extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    let src = GUINEAPATHS[this.state.currentGP];
    return (
      <div>
        <h1>Cute Guinea Pigs</h1>
        <img src={src} />
      </div>
    );
  }
}

ReactDOM.render(
  <GuineaPigs />,
  document.getElementById('app')
);
```

GuinePigsContainer.js

```
import React from 'react';
import ReactDOM from 'react-dom';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-4.jpg'
];

class GuineaPigs extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;
    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    let src = GUINEAPATHS[this.state.currentGP];
    return (
      <div>
        <h1>Cute Guinea Pigs</h1>
        <img src={src} />
      </div>
    );
  }
}

ReactDOM.render(
  <GuineaPigs />,
  document.getElementById('app')
);
```

Separate Presentational Component
Now that we've created a container component for the logic, we can dedicate the original component, `GuineaPigs`, to be a presentational component.

The presentational component's only job is to contain HTML-like JSX. It should be an exported component and will not render itself because a presentational component will always get rendered by a container component.

As a separate example, say we have `Presentational` and `Container` components. **Presentational.js** must export the component class (or function, when applicable):

```
export class Presentational extends Component {
```

Container.js must import that component:

```
import { Presentational } from 'Presentational.js';
```

Instructions

1.
Select **components/GuineaPigs.js**.

Look at the `GuineaPigs` component class, starting on line 11. This is going to be your presentational component class. That means that its only job will be to contain JSX.

On line 2, delete `import ReactDOM from 'react-dom'`.

At the bottom of the file, delete the `ReactDOM.render()` call.

Export GuineaPigs by adding the keyword `export` to the beginning of `class GuineaPigs`.

Checkpoint 2 Passed

Hint
At a high level, we're going to make sure that **components/GuineaPigs.js** is an exported component and doesn't render itself. That means we'll do two things:

1. Delete all references to `ReactDOM` in this file.
2. Export the component with `export class GuineaPigs`.

Refer to the instructions for more details about how to do this.

2.
Good! But why did you just do that?

GuineaPigs will get rendered by GuineaPigsContainer. Any component that gets rendered by a different component should use export.

Select **containers/GuineaPigsContainer.js**.

Make a new line after line 2. On your new line, import GuineaPigs.

This will be slightly different from what you've done before! As you saw when you opened the file navigator, **GuineaPigs.js** and **GuineaPigsContainer.js** are not neighbors. The file path that you pass to import will have to navigate up one level, and then down into the components folder.

Checkpoint 3 Passed

Hint

Import GuineaPigs using:

```
import { GuineaPigs } from '../components/GuineaPigs';
```

GuineaPig.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { GuineaPigs } from '../components/GuineaPigs';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-4.jpg'
];
```

```
class GuineaPigs extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    let src = GUINEAPATHS[this.state.currentGP];
    return (
      <div>
        <h1>Cute Guinea Pigs</h1>
        <img src={src} />
      </div>
    );
  }
}

ReactDOM.render(
  <GuineaPigs />,
  document.getElementById('app')
);
```

GuinePigsContainer.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { GuineaPigs } from '../components/GuineaPigs';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-4.jpg'
];

class GuineaPigs extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    let src = GUINEAPATHS[this.state.currentGP];
    return (
      <div>
        <h1>Cute Guinea Pigs</h1>
        <img src={src} />
      </div>
    );
  }
}

ReactDOM.render(
  <GuineaPigs />,
  document.getElementById('app')
);
```

Render Presentational Component in Container Component

We now have a container component (**containers/GuineaPigsContainer.js**) for logic and a presentational component (**components/GuineaPigs.js**) for rendering JSX!

The container component should now render the presentational component instead of rendering JSX.

Instructions

1.

Select **containers/GuineaPigsContainer.js**.

On line 12, change the component class's name from `GuineaPigs` to `GuineaPigsContainer`. In the `ReactDOM.render()` call near the bottom of the file, change `<GuineaPigs />` to `<GuineaPigsContainer />`.

Checkpoint 2 Passed

Hint

In **containers/GuineaPigsContainer.js**, we'll be renaming the component to `GuineaPigsContainer`. This means we'll need to update the definition (on the line that starts with class).

We'll also need to update it wherever it's used to reflect the new name; in this case, that means updating the argument to `ReactDOM.render()`.

2.

`GuineaPigsContainer` contains a lot of logic. It shouldn't also have to render JSX.

Delete any JSX from `GuineaPigsContainer`'s return statement in the render function. Instead, return an instance of `GuineaPigs`.

The new render function should look like this:

```
render() {
  const src = GUINEAPATHS[this.state.currentGP];
  return <GuineaPigs />;
}
```

Checkpoint 3 Passed

Hint

Open **containers/GuineaPigsContainer.js**. If you haven't already, delete its `render()` method and replace it with the code from the instructions.

3.

Once your container component has chosen a guinea pig, it must pass that guinea pig to the presentational component.

In `GuineaPigsContainer`'s render function, pass the chosen guinea pig by giving `<GuineaPigs />` a prop of `src = {src}`.

Checkpoint 4 Passed

Hint

Before you started this step, `GuineaPigsContainer` was rendering `GuineaPigs` with no attributes, like this:

```
<GuineaPigs />
```

You'll want to add the `src={src}` attribute to that component.

GuineaPigsContainer.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { GuineaPigs } from '../components/GuineaPigs';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-
guineapig-4.jpg'
];

class GuineaPigsContainer extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    const src = GUINEAPATHS[this.state.currentGP];
    return <GuineaPigs src={src}/>;
  }
}

ReactDOM.render(
  <GuineaPigsContainer />,
  document.getElementById('app')
);
```

Remove Logic from Presentational Component

Our container component now renders the `GuineaPigs` presentational component instead of JSX statements!

The last step to separating the container component from the presentational component is to remove redundant logic in the presentational component. The presentational component should be left with the render function that contains JSX statements.

Instructions

1.

Select **components/GuineaPigs.js**.

This component's only job is to render HTML-like JSX. Delete everything inside of the `GuineaPigs` component class, except for the render function. When you're done, the class should only have one method: `render()`!

Inside of the `render()` function, delete this line of logic:

```
let src = GUINEAPATHS[this.state.currentGP];
```

…and replace it with this:

```
let src = this.props.src;
```
Lastly, delete the `GUINEAPATHS` array.

Checkpoint 2 Passed

Hint

Open **components/GuineaPigs.js** and find the `GuineaPigs` component. Delete everything inside of the class except the `render()` function. That means deleting the constructor, calls to `componentDidMount()` and `componentWillUnmount()`, and everything that isn't `render()`. Once you've done that, replace this line in `render()`…

```
let src = GUINEAPATHS[this.state.currentGP];
```
…with this:

```
let src = this.props.src;
```
Lastly, delete the `GUINEAPATHS` array from this file, because we no longer need it.

GuineaPigs.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { GuineaPigs } from '../components/GuineaPigs';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-4.jpg'
];

class GuineaPigsContainer extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };
    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    const src = GUINEAPATHS[this.state.currentGP];
    return <GuineaPigs src={src} />;
  }
}

ReactDOM.render(
  <GuineaPigsContainer />,
  document.getElementById('app')
);
```

GuineaPigsContainer.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { GuineaPigs } from '../components/GuineaPigs';

const GUINEAPATHS = [
  'https://content.codecademy.com/courses/React/react_photo-guineapig-1.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-2.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-3.jpg',
  'https://content.codecademy.com/courses/React/react_photo-guineapig-4.jpg'
];

class GuineaPigsContainer extends React.Component {
  constructor(props) {
    super(props);

    this.state = { currentGP: 0 };

    this.interval = null;

    this.nextGP = this.nextGP.bind(this);
  }

  nextGP() {
    let current = this.state.currentGP;
    let next = ++current % GUINEAPATHS.length;
    this.setState({ currentGP: next });
  }

  componentDidMount() {
    this.interval = setInterval(this.nextGP, 5000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    const src = GUINEAPATHS[this.state.currentGP];
    return <GuineaPigs src={src} />;
  }
}
```

```
ReactDOM.render(
  <GuineaPigsContainer />,
  document.getElementById('app')
);
```

Review

Congrats! You divided the `GuineaPigs` component into a container component and a presentational component: **containers/GuineaPigsContainer.js** and **components/GuineaPigs.js**.

Here are the steps we took:

1. Identified that the original component needed to be refactored: it was handling both calculations/logic and presentation/rendering
2. Copied the original component to a new **containers/** folder and renamed it `GuineaPigsContainer`
3. Removed all of the presentation/rendering code from the container component
4. Removed all of the calculation/logic code from the presentational component
5. Accessed the presentational component from within the container using `import` and `export`
6. Edited the container's `render()` method to render the presentational component with the proper props

In this programming pattern, the container component does the work of figuring out what to display. The presentational component does the work of actually displaying it. If a component does a significant amount of work in both areas, then that's a sign that you should use this pattern!

If you’d like to learn more about this pattern, here are some articles to start with:

- Container Components
- Presentational and Container Components