React Design Patterns

What are Design Patterns ?

Design Patterns are effective solutions to common application development challenges.

Common Challenges

- Creating reusable layouts
- Reusing complex logic between multiple components
- Working with forms
- Incorporating functional concepts into our code

Layout Components

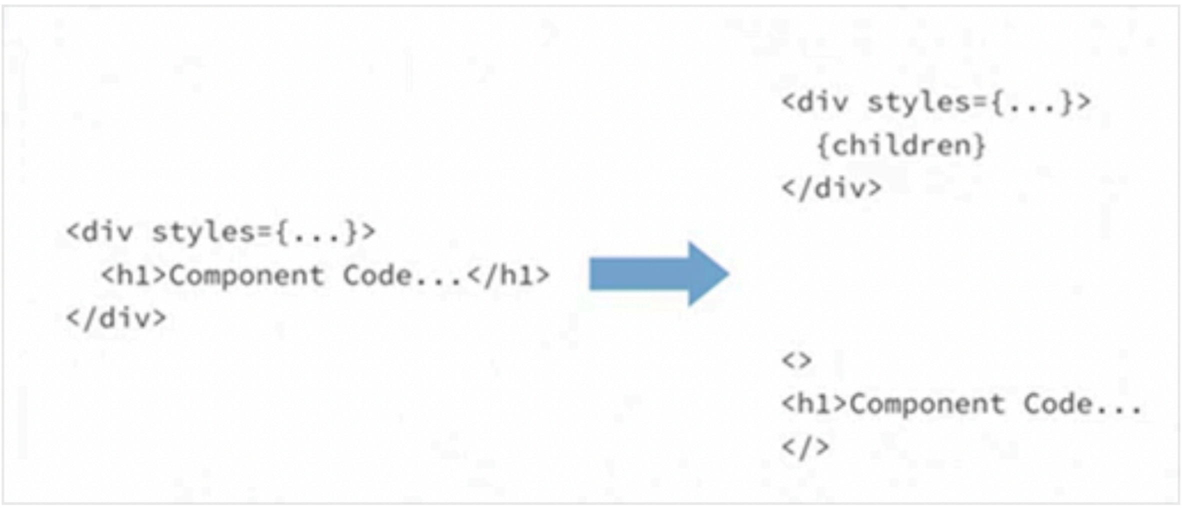
React components that deal primarily with arranging other components on the page.

Example:

- Split Screen
- List and Items
- Modals

Idea of Layout Components

Our Components shouldn't know where they are being displayed.



```
<div styles={...}>
  <h1>Component Code...</h1>
</div>
```

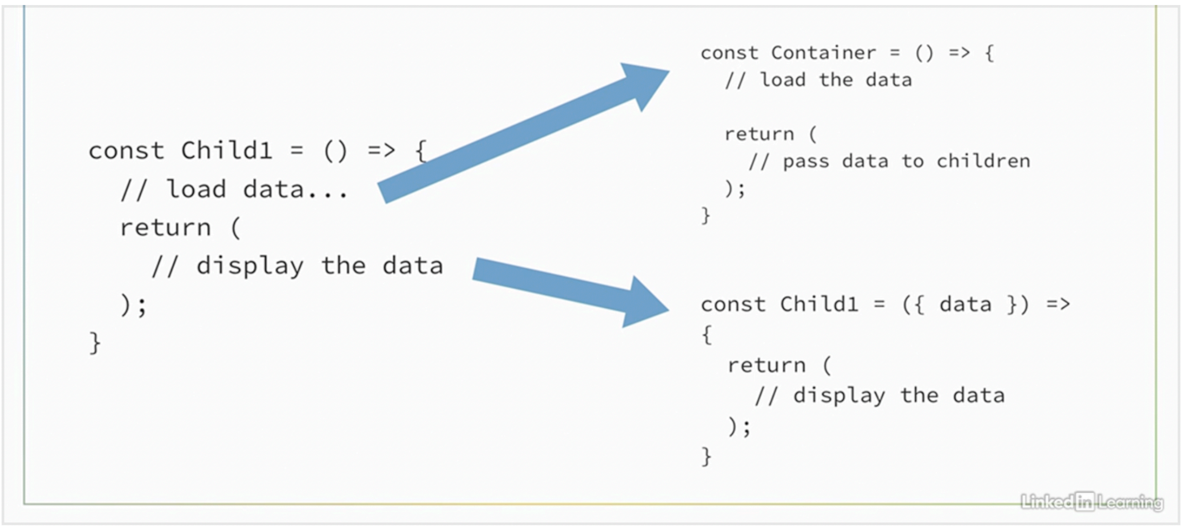
```
<div styles={...}>
  {children}
</div>
```

```
<>
<h1>Component Code...
</>
```

Container Components

Components that take care of loading and managing data for their child components

```
<ContainerComponent>
    <Child1 />
    <Child2 />
    <Child3 />
</ContainerComponent>
```



```
const Child1 = () => {
  // load data...
  return (
    // display the data
  );
}
```

```
const Container = () => {
  // load the data

  return (
    // pass data to children
  );
}
```

```
const Child1 = ({ data }) =>
{
  return (
    // display the data
  );
}
```

Idea of the container components is our components shouldn’t know where their data is coming from.

Controlled and Uncontrolled components

Uncontrolled Components are components that keep track of their own states and release data only when some event occurs (like submit event for HTML forms)

Ex:

```
const MyComponent = ({ onSubmit }) => {
    const [someState, setState] = useState(…);

    return …;
}

<MyComponent onSubmit={data => …} />
```

Controlled components don't keep track of their own state - all state is passed in as props (i.e. when we use useState hook with text input).

Ex:

```
 const MyComponet = ({ data, changeData, onSubmit }) =>
{
   return …;
}

<MyComponent
   data={…}
   changeData={() => …}
   onSubmit={() => …}
/>
```

Higher-Order Components(HOCs)

A component that returns another component instead of JSX.

```
MyComponent ——> <h1>I'm JSX!</h1>

HOC ———> SomeComponent ——> <h1>I'm JSX!</h1>
```

HOCs are just functions

HOCs are used for

- sharing complex behaviour between multiple components (much like with container components)
- Add extra functionality to existing components

Custom Hooks

Sharing complex behaviour between multiple components (much like with HOCs and container components)

Functional Programming

A method of organising code in a way that

- Minimizes mutation and state change
- Keeps functions independent of external data
- Treat functions as first class citizens

Applications of FP in React

- Controlled Components
- Functional Components
- HOCs
- **Recursive Components**
- **Partially Applied Components**
- **Component Composition**