Using a Component

After creating our first component, it's time to use it.

To make it happen, we will return again to JavaScript modularity. In order to use any external module in our local file, we should import it first.

Let's see an example to consolidate this.

MyFirstComponent.js

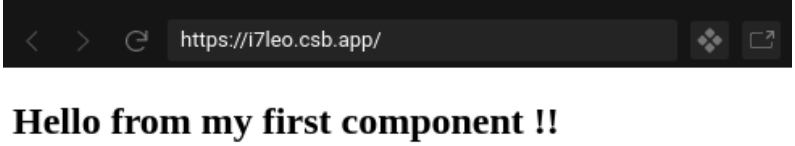
```
import React from "react";

const MyFirstComponent = () => {
 return <h1> hello from my first component </h1>;
};
export default MyFirstComponent;
```

App.js

```
import React from "react";
import MyFirstComponent from "./MyFirstComponent";
const App = () => (
 <>
   <MyFirstComponent />
 </>
);
export default App;
```

Output



Using a Component

Buckle up! It's time we started creating things.

As we can see, we find the component MyFirstComponent inside the App. It's only right to ask if the constant App is a component or not. The answer is: Yes, a component can be composed of many other components.

```
const App = () => {
 return (
   <div>
     <Header /> // Header Component
     <Footer />
     // Footer Component
   </div>
 );
};
```

When one component uses another like this, a parent-child relationship is formed. In above example :

- The App component is the parent to the Header and Footer components.
- The Header and Footer components are the child of the App component.

Using a component as a root

We talked earlier about the **ReactDom.render()** method in the JSX chapter. Basically, that method exists in the index.js file. This file is called the root component.

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
const rootElement = document.getElementById("root");//We call this a “root” DOM node
ReactDOM.render(<App />,rootElement);
```

You can notice that applications built with React have a **single root** DOM node.

‹ Previous next ›