اجزا(Component) در ReactJS

در این بخش می خواهیم به چگونگی ترکیب اجزا بپردازیم، به گونه ای که نگهداری از برنامه آسان تر شود. با استفاده از این روش می توانیم اجزای خود را بدون تحت تأثیر قرار دادن بخش های باقی مانده ی صفحه به روز رسانی کرده و تغییر دهیم.

مثال بدون State

اولین جزء ما درمثال زیر App است. این جزء مالک Header و Content است. می خواهیم به صورت مجزا Header و Content را ایجاد کنیم و صرفا آن ها را داخل درخت JSX جزء App خود اضافه کنیم. تنها جزء App باید اکسپورت شود.

App.jsx

```
import React from 'react';


class App extends React.Component {
  render() {
    return (
      <div>
        <Header/>
        <Content/>
      </div>
    );
  }
}
class Header extends React.Component {
  render() {
```

```
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>Content</h2>
        <p>The content text!!!</p>
      </div>
    );
  }
}
export default App;
```

برای آن که بتوانیم نتیجه را در صفحه نمایش دهیم، باید این کد را داخل فایل main.js ایمپورت کنیم و reactDOM.render() را فراخوانی کنیم. قبلا این کار را در بخش برپا کردن محیط انجام داده ایم.

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
```

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

کد بالا نتیجه ی زیر را ایجاد می کند.

مثال با State

در این مثال می خواهیم state یا حالت جزء مالک (App) را تنظیم کنیم. با توجه به اینکه جزء Header هیچ حالتی ندارد، آن را مانند مثال قبل اضافه می کنیم. به جای برچسب content عناصر table و tbody را ایجاد می کنیم و از طریق آن ها به ازای هر شیء از آرایه ی data، TableRow را به صورت پویا درج می کنیم.

همان طور که مشخص است، ما از سینتکس EcmaScript 2015 arrow (=>) استفاده می کنیم. چرا که این سینتکس تمیزتر از سینتکس قدیمی جاوا اسکریپت به نظر می رسد. از این طریق می توانیم عناصر خود را با کدهای کمتری ایجاد کنیم. این سینتکس به ویژه در مواقعی کاربرد دارد که بخواهیم لیستی را با آیتم های بسیاری ایجاد کنیم.

App.jsx

```
import React from 'react';


class App extends React.Component {
  constructor() {
    super();
    this.state = {
      data:
```

```
    [
      {
        "id":1,
        "name":"Foo",
        "age":"20"
      },
      {
        "id":2,
        "name":"Bar",
        "age":"30"
      },
      {
        "id":3,
        "name":"Baz",
        "age":"40"
      }
    ]
  }
}
render() {
  return (
    <div>
      <Header/>
      <table>
        <tbody>
          {this.state.data.map((person, i) => <TableRow key = {i}
            data = {person} />)}
```

```
          </tbody>
        </table>
      </div>
    );
  }
}
class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
class TableRow extends React.Component {
  render() {
    return (
      <tr>
        <td>{this.props.data.id}</td>
        <td>{this.props.data.name}</td>
        <td>{this.props.data.age}</td>
      </tr>
    );
  }
}
export default App;
```

main.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';



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

نکته: توجه داشته باشید که ما از تابع map() inside key = {i} استفاده می کنیم. از این طریق ری اکت راحت تر می تواند صرفا عناصر ضروری را به روز رسانی کند. به جای آن که زمانی که چیزی تغییر می کند، کل لیست را مجددا رندر کند. در صورتی که با تعداد زیادی از عناصری سروکار داشته باشیم که به صورت پویا ایجاد شده اند، این کار باعث می شود عملکرد تا حد چشمگیری افزایش یابد.