05. 컴포넌트로 UI 구축하기

Chapter 5: 컴포넌트로 UI 구축하기

React의 핵심 개념

React 애플리케이션을 구축하기 위해 익숙해져야 할 세 가지 핵심 개념은 다음과 같습니다:

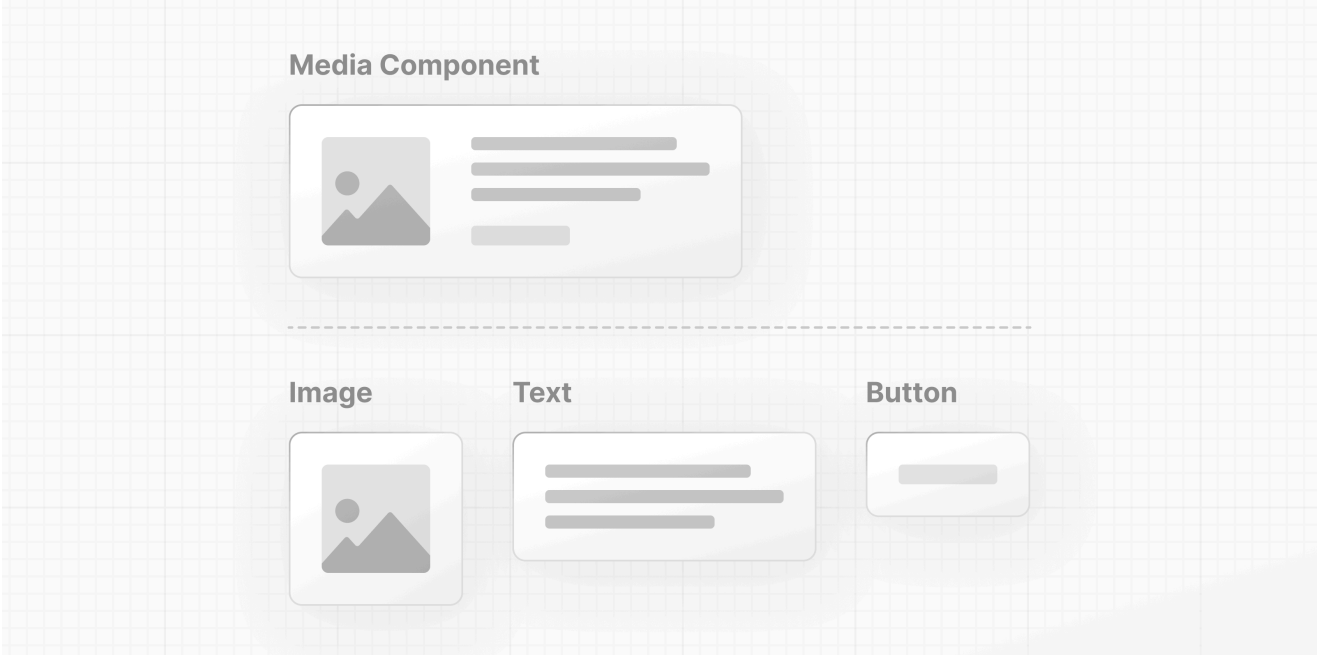
- **컴포넌트(Components)**
- **프롭스(Props)**
- **상태(State)**

다음 챕터들에서 이 개념들을 하나씩 살펴보고, 추가 학습을 위한 리소스를 제공할 예정입니다. 이 개념들에 익숙해진 후에는 Next.js를 설치하고, Server 및 Client 컴포넌트와 같은 최신 React 기능을 사용하는 방법을 소개할 것입니다.

컴포넌트(Components)

사용자 인터페이스는 **컴포넌트**라고 불리는 더 작은 구성 요소들로 나눌 수 있습니다.

컴포넌트를 사용하면 **자체 포함되고 재사용 가능한 코드 조각**을 만들 수 있습니다. 컴포넌트를 레고 블록에 비유하면, 이러한 개별 블록을 조합하여 더 큰 구조를 만들 수 있습니다. UI의 일부를 업데이트해야 할 경우, 특정 컴포넌트나 블록만 업데이트하면 됩니다.



이러한 모듈성은 애플리케이션이 커짐에 따라 코드를 더 유지 관리하기 쉽게 만듭니다. 애플리케이션의 나머지 부분을 건드리지 않고도 컴포넌트를 추가, 업데이트 또는 삭제할 수 있습니다.

React 컴포넌트의 좋은 점은 **단지 JavaScript 함수**라는 것입니다. JavaScript 관점에서 React 컴포넌트를 어떻게 작성할 수 있는지 살펴보겠습니다.

컴포넌트 생성하기

React에서 컴포넌트는 **함수**입니다. `script` 태그 안에 `header` 라는 새 함수를 만들어보세요:

```
<script type="text/jsx">
  const app = document.getElementById("app");

  function header() {
  }

  const root = ReactDOM.createRoot(app);
  root.render(<h1>Develop. Preview. Ship.</h1>);
</script>
```

컴포넌트는 UI 요소를 반환하는 함수입니다. 함수의 `return` 문 안에 JSX를 작성할 수 있습니다:

```
<script type="text/jsx">
  const app = document.getElementById("app");

  function header() {
    return (<h1>Develop. Preview. Ship.</h1>);
  }

  const root = ReactDOM.createRoot(app);
  root.render(<h1>Develop. Preview. Ship.</h1>);
</script>
```

이 컴포넌트를 DOM에 렌더링하려면, `root.render()` 메서드의 첫 번째 인자로 전달하면 됩니다:

```
<script type="text/jsx">
  const app = document.getElementById("app");

  function header() {
    return (<h1>Develop. Preview. Ship.</h1>);
  }

  const root = ReactDOM.createRoot(app);
  root.render(header);
</script>
```

하지만 위 코드를 브라우저에서 실행하면 에러가 발생할 수 있습니다. 이를 해결하려면 두 가지를 확인해야 합니다:

1. **컴포넌트 이름을 대문자로 시작**해야 합니다. 이는 React 컴포넌트를 일반 HTML 및 JavaScript와 구분하기 위함입니다:

```
function Header() {
  return <h1>Develop. Preview. Ship.</h1>;
}

const root = ReactDOM.createRoot(app);
// React 컴포넌트는 대문자로 시작해야 합니다
root.render(Header);
```

2. **React 컴포넌트를 사용할 때는 일반 HTML 태그처럼 꺾쇠 괄호 `<>` 를 사용**해야 합니다:

```
function Header() {
  return <h1>Develop. Preview. Ship.</h1>;
}

const root = ReactDOM.createRoot(app);
root.render(<Header />);
```

이제 브라우저에서 코드를 실행하면 변경 사항을 확인할 수 있습니다.

컴포넌트 중첩하기

애플리케이션은 보통 단일 컴포넌트보다 더 많은 콘텐츠를 포함합니다. React 컴포넌트는 일반 HTML 요소처럼 서로 중첩할 수 있습니다.

예를 들어, `HomePage` 라는 새 컴포넌트를 만들어보세요:

```
function Header() {
  return <h1>Develop. Preview. Ship.</h1>;
}

function HomePage() {
  return <div></div>;
}

const root = ReactDOM.createRoot(app);
root.render(<Header />);
```

그런 다음, `<Header>` 컴포넌트를 새 `<HomePage>` 컴포넌트 안에 중첩해보세요:

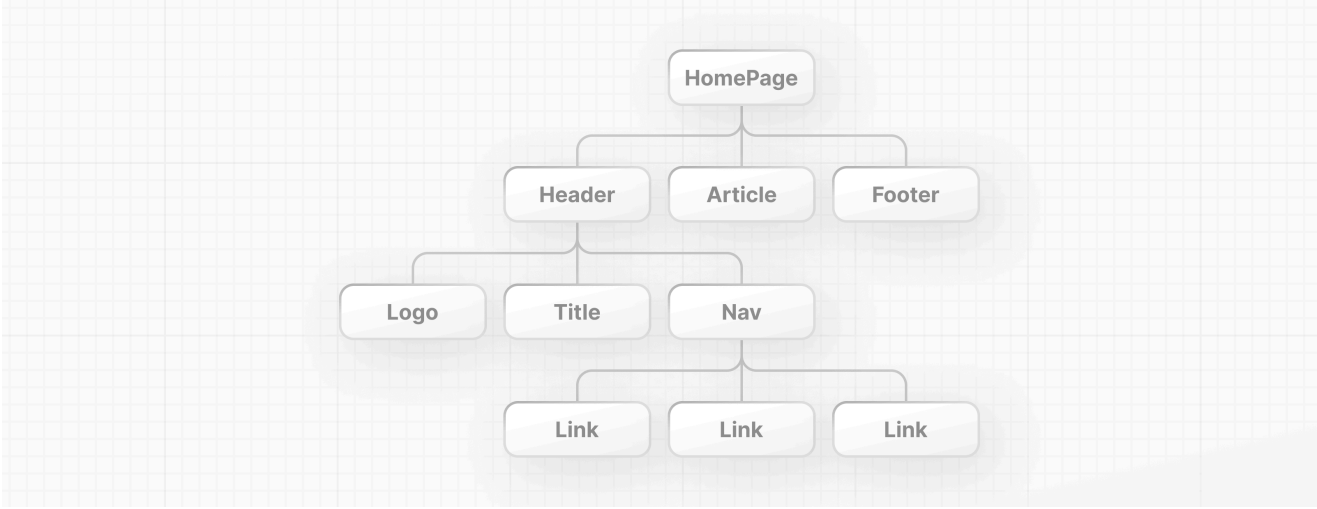
```
function Header() {
  return <h1>Develop. Preview. Ship.</h1>;
}

function HomePage() {
  return (
    <div>
      {/* Header 컴포넌트 중첩 */}
      <Header />
    </div>
  );
}

const root = ReactDOM.createRoot(app);
root.render(<Header />);
```

컴포넌트 트리

이러한 방식으로 React 컴포넌트를 계속 중첩하여 **컴포넌트 트리**를 형성할 수 있습니다.



예를 들어, 최상위 `HomePage` 컴포넌트는 `Header`, `Article`, `Footer` 컴포넌트를 포함할 수 있습니다. 그리고 각 컴포넌트는 자체 자식 컴포넌트를 가질 수 있습니다. 예를 들어, `Header` 컴포넌트는 `Logo`, `Title`, `Navigation` 컴포넌트를 포함할 수 있습니다.

이러한 모듈 형식은 애플리케이션 내의 다른 위치에서 컴포넌트를 재사용할 수 있게 해줍니다.

이제 프로젝트에서 `<HomePage>`가 가장 상위 레벨 컴포넌트이며, 이 컴포넌트를 `ReactDOM.render()` 메서드에 전달할 수 있습니다:

```
function Header() {
  return <h1>Develop. Preview. Ship.</h1>;
}
```

```
function HomePage() {
  return (
    <div>
      <Header />
    </div>
  );
}

const root = ReactDOM.createRoot(app);
root.render(<HomePage />);
```