**React란?**

React는 페이스북에서 개발한 사용자 UI를 개발하기 위한 자바스크립트 라이브러리입니다.

컴포넌트 기반 아키텍처를 제공하여 UI 개발을 단순화

**React의 특징**

**컴포넌트 기반:** UI를 작은 독립적인 부분으로 나누어 관리

**단방향 데이터 흐름:** 데이터의 단방향 흐름을 통해 예측 가능하고 유지보수가 용이한 코드를 작성

**재사용 가능 컴포넌트**: 컴포넌트를 재사용 가능하다

**가상 DOM**: 실제 DOM 과 가상 DOM과 비교가 가능

**가상 DOM**:

React 상태가 업데이트 될 때 마다 가상 DOM과 실제 DOM 과 비교

**장점**

**성능:** 효율적인 업데이트를 가능하게 하여 빠른 렌더링과 최적화된 성능을 제공

**주요 React. 라이브러리 및 도구 소개**

* ﻿﻿**Create React App (CRA)**
* ﻿﻿React 애플리케이션을 빠르게 시작할 수 있는 공식 CLI 도구입니다.
* ﻿﻿번들링, 빌드 설정 등의 초기 설정을 자동으로 수행해 줍니다.
* ﻿﻿**React Router**
* ﻿﻿React 애플리케이션에서 라우팅을 관리하는 라이브러리입니다.
* ﻿﻿SPA(Single Page Application) 에서 페이지 간 이동을 쉽게 구현할 수 있습니다.
* ﻿﻿**Redux**
* ﻿﻿애플리케이션의 상태 관리를 위한 라이브러리입니다.
* ﻿﻿복잡한 상태 관리 로직을 간결하게 유지할 수 있습니다.
* ﻿﻿**React Query**
* ﻿﻿서버 상태를 관리하고 데이터 페칭을 간소화하는 라이브러리입니다.

﻿﻿캐싱, 동기화, 서버 상태 관리 등을 효율적으로 처리하도록 해줍니다.

* ﻿﻿**Styled-components**
* ﻿﻿컴포넌트 기반 스타일링 도구입니다.
* ﻿﻿자바스크립트 코드 내에서 CSS를 작성하여 스타일을 컴포넌트와 함께 관리할 수 있습니다.
* ﻿﻿**Tailwind CSS:**
* ﻿﻿미리 정의된 CSS 클래스 사용하여 개별 스타일링을 할 수 있는 도구입니다.
* ﻿커스터마이징과 재사용성에 강점이 있습니다.
* ﻿﻿**NextUI**
* ﻿﻿최신 UI 컴포넌트 라이브러리로서 아름답고 사용하기 쉬운 컴포넌트 라이브러리입니다.

﻿﻿다크 모드, 테마 커스터마이징 등의 기능을 포함하고 있습니다.

React 컴포넌트

UI를 구성하는 독립적인 단위

컴포넌트는 웹 애플리케이션의 UI를 구성하는 독립적인 블록

자신만의 상태와 UI를 가지고 있으며, 다른 컴포넌트와 독립적으로 작동

재사용 가능한 코드 블록

컴포넌트 재사용 가능한 코드 블록으로, 한 번 작성한 컴포넌트를 여러곳에서 재사용할 수 있다.

컴포넌트의 종류

함수형 컴포넌트 자바스크립트 함수로 작성

상태 관리와 생명주기 메소드를 사용하기 위해 React Hooks 사용할 수 있음

상태와 생명주기 메소드를 Hook으로 대체 useEffect를 사요하여 컴포넌트가 마운트되거나 업데이트될 때 특정 작업을 수행할 수 있습니다.

**클래스형 컴포넌트**

* ﻿﻿클래스형 컴포넌트는 ES6 클래스를 사용하여 작성됩니다.
* ﻿﻿React.Component를 상속받아 생성되며, 상태(state)와 생명주기 메소드를 포함합니다.

﻿﻿기존의 React 버전에서 널리 사용되었지만, React Hooks의 도입으로 함수형 컴포넌트로 대체되는 추세입니다.

**JSX란 무엇인가?**

**JavaScript XML (JSX)**

* ﻿﻿JSX는 자바스크립트와 XML을 결합한 확장 문법으로, React에서 UI를 표현하기 위해 사용됩니다.
* ﻿﻿자바스크립트 코드 내에 HTML과 비슷한 문법을 사용하여 React 요소를 생성할 수 있게 해줍니다.

**React 요소를 만들기 위한 문법**

* ﻿﻿JSX를 사용하여 React 요소(React elements)를 생성하고 이들을 결합하여 UI를 작성할 수 있습니다.

﻿﻿JSX는 React.createElement 함수 호출로 변환되어 React와 브라우저가 이해할 수 있는 자바스크립트 객체로 변환됩니다.

JSX 장점

읽기 쉬운 코드

익숙한 HTML 문법 사용

컴포넌트 구조 명확화 구조가 명확하게 드러나며, 각 요소와 이벤트 핸들러가 어떻게 결합되어 있는지 쉽게 파악할 수 있다.

JSX 특징 중괄호를 사용한 자바스크립트 코드 삽입

Key

React에서 리스트를 렌더링 할 때 각 항목에 고유한 식별자를 부여

리스트 항목 간의 식별을 통해 React가 어떤 항목이 변경 추가 삭제 정확히 알 수 있다

이를 통해 재렌더링 성능을 최적화

**React Hooks**

* ﻿﻿리액트 훅(React Hooks)은 리액트 16.8 버전에서 새롭게 추가된 기능입니다.
* ﻿﻿이전에는 클래스 컴포넌트에서만 상태(state)를 관리하거나 라이프사이클 메서드를 사용할 수 있었지만, 훅을 이용하면 함수형 컴포넌트에서도 상태 관리와 다양한 리액트 기능을 사용할 수 있게 되었습니다.
* ﻿﻿일반적으로 usestate, useEffect 등과 같이, 함수 이름 앞에는 보통 use 가 사용됩니다.

**Hook의 장점**

* ﻿﻿가독성: 클래스 컴포넌트보다 함수형 컴포넌트가 보다 간결하고 가독성이 좋습니다.
* ﻿﻿재사용성: 상태 관리나 부수 효과 로직을 별도의 혹으로 분리하여 재사용하기 용이합니다.

﻿﻿테스트 용이성: 훅을 사용하면 테스트가 쉬워지고 테스트 코드가 간단해집니다.

상태와 속성

상태(state)

정의와 개념

상태는 컴포넌트 내에서 관리되는 데이터. 각각의 컴포넌트 자체적인 상태를 가질 수 있음 컴포넌트의 생명주기 동안 변결될 수 있음

상태 관리의 필요성

Ui 동적인 변화를 표현하고, 사용자 상호작용에 따라 데이터를 저장하고 업데이트

상태 업데이트 방법 useState setState 메소드를 활용하여 업데이트 한다

속성(Props)

정의와 개념 부모 컴포넌트에서 자식 컴포넌트로 데이터 전달할 때 사용

속성은 내부에서 읽기전용이 사용되며, 변경 할 수 없다

속성 사용의 필요성

데이터 전달 재사용성을 높이고, 구조를 모듈화하여 관리

동적으로 데이터를 전달하므로 다양한 상황에 유연하게 대응

Children 속성

컴포넌트의 태그 사이에 위치한 요소들이 children props로 전달

이를 활용하여 부모 컴포넌트에서 자식 요소를 포함하고 조작할 수 있음

**React에서 이벤트 처리하기**

**이벤트 처리 기본 개념**

* ﻿﻿React에서는 DOM 요소에 이벤트를 처리하는 방식이 일반적인 HTML과 다릅니다.
* ﻿﻿이벤트 처리는 자바스크립트의 이벤트 핸들러와 유사하지만 몇 가지 차이가 있습니다.

﻿﻿React 이벤트는 소문자 대신 camelCase를 사용합니다. 예를 들어, HTML의 onclick 은 React에서 onClick 으로 표기됩니다.

**이벤트 처리 방법**

* ﻿﻿이벤트 핸들러 등록
* ﻿﻿JSX 문법을 통해 이벤트 핸들러를 DOM 요소에 직접 등록할 수 있습니다.
* ﻿﻿예를 들어, oncick, onchange, onSubmit 등의 속성을 사용하여 이벤트 핸들러를 등록할 수 있습니다.
* ﻿﻿이벤트 핸들러 구현
* ﻿﻿함수 형태의 이벤트 핸들러를 정의하고, 이를 이벤트 속성에 할당합니다.
* ﻿﻿이벤트 객체
* ﻿﻿React 이벤트 핸들러는 항상 첫 번째 인수로 이벤트 객체를 받습니다.
* ﻿﻿이벤트 객체는 DOM 이벤트와 같은 방식으로 동작하지만, React에서는 크로스 브라우징을 처리하기 위해 일관된 API를 제공합니다.

﻿﻿이벤트 객체를 사용하여 이벤트에 대한 추가 정보를 얻거나, 이벤트의 기본 동작을 방지할 수 있습니다.

**주요 이벤트 종류**

* ﻿﻿onClick: 버튼이나 다른 요소를 클릭할 때 발생하는 이벤트입니다.
* ﻿﻿onSubmit: 폼을 제출할 때 발생하는 이벤트입니다.
* ﻿﻿onChange: 입력 필드나 선택 필드의 값이 변경될 때 발생하는 이벤트입니다.
* ﻿﻿onMouseOver, onMouseOut: 마우스가 요소 위로 올라가거나 빠져나갈 때 발생하는 이벤트입니다.

﻿﻿onKeyDown, onkeyUp: 키보드의 키가 눌렸을 때와 떼어졌을 때 발생하는 이벤트입니다.

React 생명주기

주요 생명 주기

마운팅

컴포넌트가 생성될 때 초기 상태를 설정

컴포넌트가 화면에 나타날 때 초기 데이터 로드나 외부 API 호출을 수행

업데이트

컴포넌트의 상태가 변경될 때마다 UI를 업데이트 할 수 있습니다.

성능 최적화를 위해 업데이트를 처리할지 여부를 결정

언마운팅

컴포넌트가 제거되기 전에 정리 작업을 수행

useState와 useReducer를 이용한 상태 관리 훅

useState

함수형 컴포넌트에서 상태를 관리하기 위한 훅

상태 변수와 해당 상태를 갱신할 수 있는 함수를 제공

장점: 간단하고 직관적으로 컴포넌트의 상태를 관리, 여러 개의 상태를 독립적으로 관리

**useReducer**

함수형 컴포넌트에서 복잡한 상태 관리 로직을 처리하기 위한 훅으로, 상태를 업데이트하는 로직을 더 세밀하게 제어할 수 있습니다.

• 사용 방법: 현재 상태와 상태를 업데이트하는 액션을 처리하는 리듀서

Actions

함수를 받아 사용합니다.

• 장점: 복잡한 상태 관리 로직을 구현할 때 유용하며, 다양한 액션에 따라 상태를 변경할 수 있습니다.

useReduce의 Flux 패턴

* ﻿﻿액션(Action): 상태의 변화를 설명하는 객체입니다. 액션은 상태 변경의 이벤트 를 나타냅니다.
* ﻿﻿디스패처(Dispatcher): 액션을 리듀서로 전달하는 역할을 합니다.

﻿﻿리듀서(Reducer): 현재 상태와 액션을 받아 새로운 상태를 반환합니다.

useState와 useReducer를 이용한 상태 관리

**useReducer을 사용하는 이유**

* ﻿﻿복잡한 상태 관리
* ﻿﻿useState는 단순한 상태 값만 관리할 수 있지만, useReducer는 복잡한 상태 로직을 다루기에 적합합니다.
* ﻿﻿상태와 그 상태를 업데이트하는 로직을 하나의 리듀서 함수로 관리할 수 있습니다.
* ﻿﻿이는 애플리케이션 규모가 크거나 상태 관리가 복잡한 경우 유용합니다.
* ﻿﻿컴포넌트 로직 분리
* ﻿﻿useReducer는 상태와 업데이트 로직을 하나의 단일 객체로 관리하기 때문에, 컴포넌트의 UI 로직과 분리할 수 있습니다.

﻿﻿이로 인해 컴포넌트가 더 간결해지고 재사용성이 증가합니다.

Context API를 이용한 상태 관리

**Context ApI**는 React에서 전역적으로 상태를 관리하기 위한 메커니즘입니다. 이는, 모든 컴포넌트 혹은 개발자가 설정한 범위 내의 컴포넌트에서 상태를 공유할 수 있다는 것을 의미합니다.

**전역 상태 관리가 필요한 이유**

1. 복잡한 컴포넌트 계층 구조에서의 상태 전파

* ﻿﻿React에서는 props를 통해 상위에서 하위로 데이터를 전달하는 방식을 사용하는데, 컴포넌트 계층 구조가 깊어질수록 데이터 전달이 번거로워질 수 있습니다. (prop driling 문제)

﻿﻿전역 상태 관리는 중첩된 컴포넌트 구조에서 상태를 일관되게 유지하고, 필요한 곳에서 쉽게 접근할 수 있도록 도와줍니다.

**전역 상태 관리가 필요한 이유**

**2. 상태의 공유와 재사용성 증대**

* ﻿﻿사용자 인증 상태, 테마 등 여러 컴포넌트가 동일한 상태를 공유해야 하는 경우가 있습니다.
* ﻿﻿전역 상태를 사용하면 이러한 데이터를 각 컴포넌트마다 props로 전달하지 않고, 필요한 곳에서 직접 접근할 수 있어 코드의 중복을 줄이고 재사용성을 증대시킬 수 있습니다.

**3. 상태 관리의 중앙 집중화**

• 전역 상태 관리는 애플리케이션 내에서 상태를 중앙 집중화하여 관리할 수 있습니다.

* ﻿﻿이는 코드의 유지보수성을 높이고, 버그를 줄이는 데 도움을 줍니다.

﻿﻿상태 관리 로직이 하나의 곳에서 관리되기 때문에 상태 업데이트 로직을 추적하고 이해하기 쉬워집니다.

**Context. API 사용 시 고려 사항**

* ﻿﻿**복잡성 증가**: Context를 사용하면 애플리케이션의 전역 상태가 많은 컴포넌트에 걸쳐 퍼질 수 있습니다. 이로 인해 상태의 흐름을 이해하기 어려워질 수 있고, 코드의 복잡성이 증가할 수 있습니다.

﻿﻿**성능 문제:** Context를 사용할 때 모든 Consumer는 provider가 업데이트될 때마다 다시 렌더링됩니다. 따라서 Provider가 자주 업데이트되는 경우 성능 문제가 발생할 수 있습니다.