

# 10강

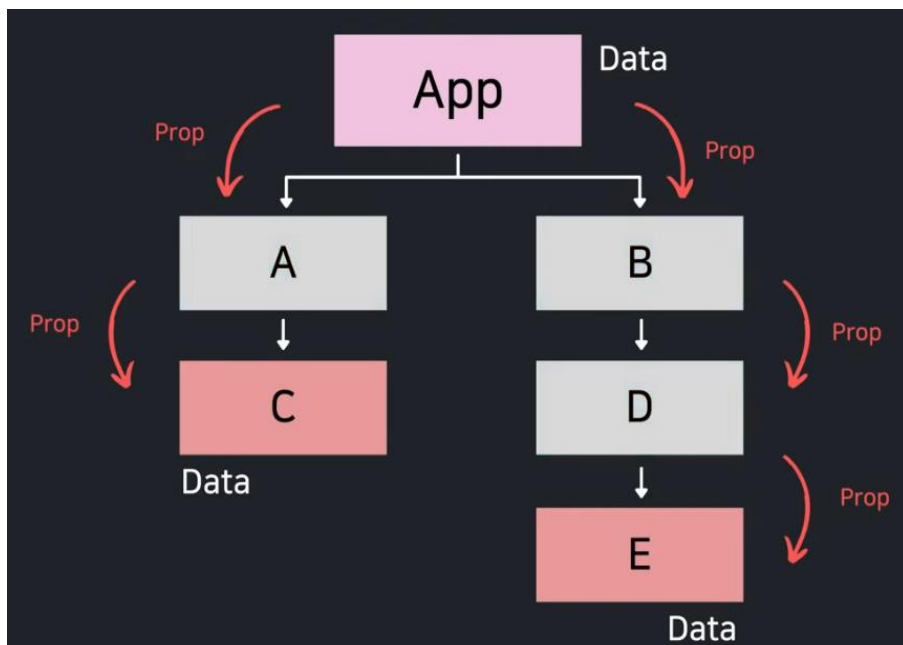
## Context API





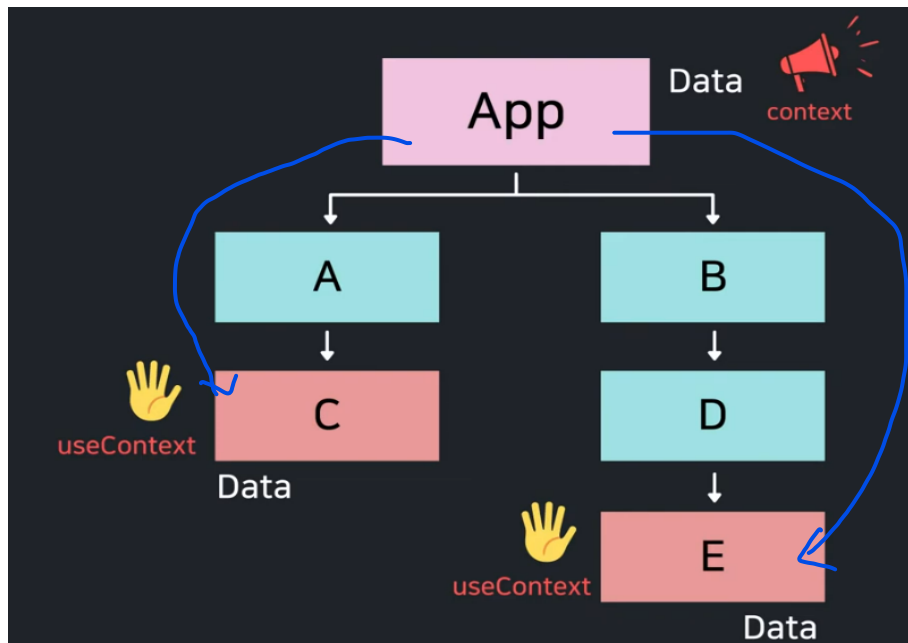
## ➤ Context API 란

- Context API는 컴포넌트 간 상태(state)나 데이터를 전역적으로 관리하고 전달하기 위한 내장 기능이다.  
쉽게 말해, props 없이도 데이터를 여러 컴포넌트가 공유할 수 있게 해주는 통로이다.
- Context API를 쓰면 중간 단계를 건너뛰고 바로 전달할 수 있다.
- 단, Context 를 사용하면 컴포넌트를 재사용하기 어려워 질 수 있다.



### Context API가 필요한 이유

- React의 기본 구조는 부모 → 자식으로 props를 전달하는 단방향 데이터 흐름이다.  
하지만 컴포넌트 구조가 깊어질수록 이런 문제가 생긴다.
- Props를 통해 중간 컴포넌트를 거쳐 전달해야 한다.
- 이런 과정을 Props Drilling(뚫고 지나간다)이라 한다.
- 문제점 C, E 컴포넌트만 자료가 필요함에도 불구하고 모든 컴포넌트에 다 Data를 전달해야 한다.



- Props 대신 context를 사용하면 데이터 필요한 사람에게 바로 데이터를 전달해 줄 수 있다.
- Props 대신 context를 사용해 전달 받은 데이터는 useContext 혹을 사용해서 데이터를 받아 오기만 하면 된다.
- useContext는 context로 공유한 데이터를 쉽게 받아오는 훅이다.

- Context는 부모 컴포넌트로부터 자식 컴포넌트로 전달되는 데이터의 흐름과는 상관없이 전역적인 데이터를 다룰 때 사용한다. 전역 데이터를 Context에 저장한 후, 데이터가 필요한 컴포넌트에서 해당 데이터를 불러와 사용할 수 있다.
- React에서 Context를 사용하기 위해서는 Context API를 사용해야 하며, Context의 Provider와 Consumer를 사용해야 한다.
- Context에 저장된 데이터를 사용하기 위해서는 공통 부모 컴포넌트에 Context의 Provider를 사용하여 데이터를 제공해야 하며, 데이터를 사용하려는 컴포넌트에서 Context의 Consumer를 사용하여 실제로 데이터를 사용한다.



### ① createContext() – “공유할 공간 만들기”

- createContext()는 데이터의 “공유 공간(저장소)”을 만드는 역할을 한다.

```
import { createContext } from 'react';
```

```
// Context 생성 (이름은 자유)
```

```
export const UserContext = createContext();
```



## ② Provider – “공유할 값 제공하기”

- **Provider**는 “이 값을 하위 컴포넌트 모두가 사용할 수 있게 하겠다”는 의미이다.
- **Value**에 전달한 값은 트리 아래의 모든 컴포넌트에서 접근할 수 있다.

```
import { useState } from 'react';
import { UserContext } from './UserContext';
export default function App() {
  const [user, setUser] = useState("홍길동");

  return (
    <UserContext.Provider value={{ user, setUser }}>
      <Layout />
    </UserContext.Provider>
  );
}
```



### ③ useContext() – “값 꺼내쓰기”

- `useContext()`를 사용하면 Provider에서 공유한 값을 바로 꺼내 쓸 수 있다.
- props 전달 없이도 접근 가능하다!

```
import { useContext } from 'react';  
import { useContext } from './UserContext';  
  
export default function UserInfo() {  
  const { user, setUser } = useContext(UserContext);  
  return (  
    <>  
      <p>현재 사용자: {user}</p>  
      <button onClick={() => setUser("김철수")}>변경</button>  
    </>  
  );  
}
```



### ➤ Context API 3단계 구조 요약

// 1. context 생성

```
export const MyContext = createContext();
```

// 2. 상위 컴포넌트에서 제공

```
<MyContext.Provider value={공유값}>
```

```
  <자식컴포넌트 />
```

```
</MyContext.Provider>
```

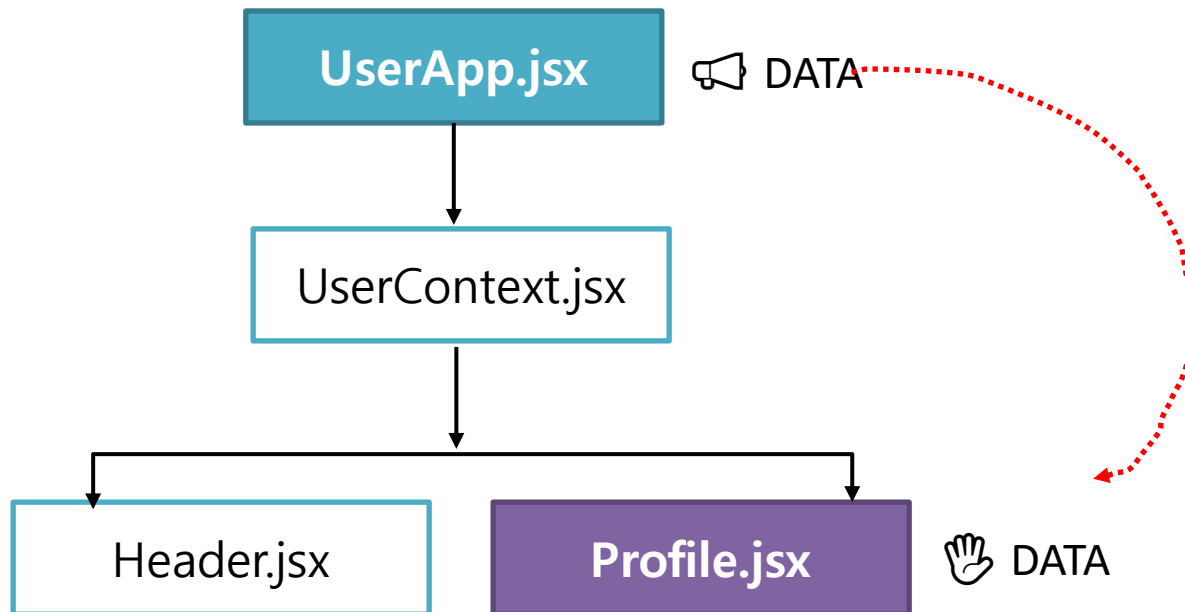
// 3. 하위 컴포넌트에서 사용

```
const data = useContext(MyContext);
```



## ➤ 여기서 잠깐

- ① 전역 사용자 이름 공유하기
- ② 부모 컴포넌트(UserApp.jsx)에서 만든 username을 손자 컴포넌트(Profile.jsx) 까지 props없이 전달 하기
- ③ src -> UserApp.jsx 파일생성 후, src -> Context 폴더 생성 -> UserContext.jsx 파일을 생성한다.
- ④ Src -> Context -> component 폴더 생상 -> Header.jsx , Profile.jsx 파일을 생성한다.
- ⑤ UserContext.jsx 파일에 createContext() 이용하여 데이터의 공유 저장소를 만든다.
- ⑥ UserApp.jsx 파일에 <UserContext.Provider value={{username, setUsername}}> /> 이용해 하위 컴포넌트가 모두 사용할 수 있게 공유할 값을 만든다.
- ⑦ Profile.jsx 파일에서 useContext()를 사용해 Provider에서 공유한 값을 꺼내서 사용한다.





## ➤ UserApp.jsx

```
import { useState } from "react";
import { UserContext } from "../Context/userContext";
import Header from "../Context/component/Header";

export default function UserApp(){
  const[username, setUsername] = useState("홍길동")
  // provider로 감싸서 전역 값 전달
  return(
    <UserContext.Provider value={{username, setUsername}}>
      <div>
        <h2>Context API 기본예제 </h2>
        <Header />
      </div>
    </UserContext.Provider>
  )
}
```



## ➤ useContext.jsx

```
import { createContext } from "react";

// Context 생성한다.
// null값은 초기값이다.
export const UserContext = createContext(null);
```

## ➤ Header.jsx

```
import Profile from "../Profile";

export default function Header(){
  return(
    <header>
      <h2>헤더 영역</h2>
      <Profile />
    </header>
  )
}
```



## ➤ Profile.jsx

```
import { useContext } from "../userContext";
import { useContext } from "react";

export default function Profile(){
  // useContext로 값 사용
  const {username, setUsername} = useContext(UserContext)
  return(
    <div>
      <p>현재 사용자 :{username}</p>
      <button type="button" onClick={()=>setUsername('이순신')}>이름 바꾸기</button>
    </div>
  )
}
```



## ➤ 출력 결과

### Context API 기본예제

#### 헤더 영역

현재 사용자:홍길동

이름 바꾸기



### Context API 기본예제

#### 헤더 영역

현재 사용자:이순신

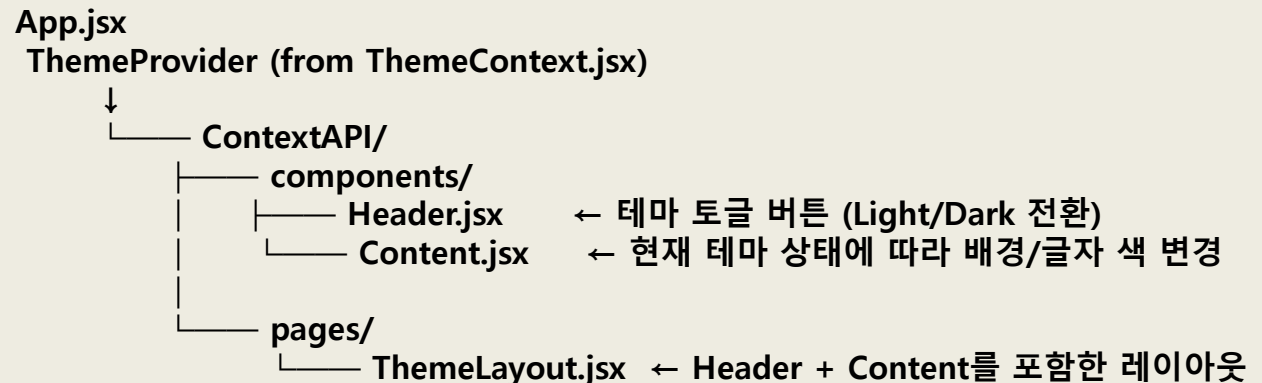
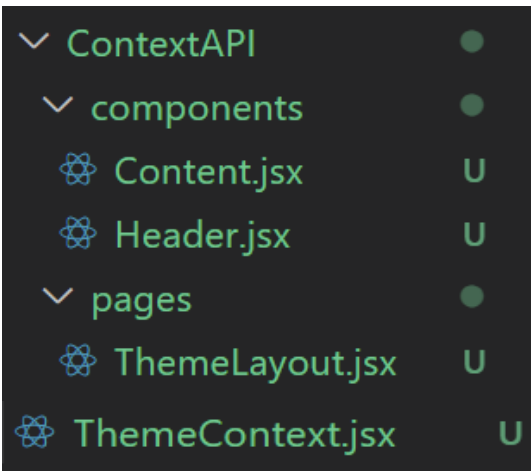
이름 바꾸기



## ➤ 여기서 잠깐

- ① Context API + useContext 혹은 사용하여 전역 상태(테마 코드)를 관리하도록 작성 하기
- ② 여러 컴포넌트에 같은 테마 적용하기
- ③ CSS를 통한 다크/라이트 모드 스타일링 버튼으로 토글기능
- ④ src -> ThemeContext.jsx 파일생성 한다.
- ⑤ src -> ContextAPI 폴더 생성한다. , ContextAPI 폴더 안에 -> components폴더, pages폴더 생성한다.
- ⑥ src -> ContextAPI -> components -> Header.jsx , Content.jsx 파일을 생성한다.
- ⑦ src -> ContextAPI -> pages -> ThemeLayout.jsx 파일을 생성한다.

## 폴더 구조





## ➤ ThemeContext.jsx

```
import { createContext, useState } from 'react';

1) Context 생성
export const ThemeContext = createContext();

2) Provider 컴포넌트 정의
export default function ThemeProvider({ children }) {

  // 불리언 값으로 테마 관리 (false = light, true = dark)
  const [theme, setTheme] = useState(false);

  /*
  토글 함수 (함수형 업데이트)
  - prev는 React가 자동으로 이전 상태를 전달해줍니다.
  - !prev : 이전 상태의 반대값으로 바꿔주는 연산자 (true ↔ false 전환)
  */
  const toggleTheme = () => {
    setTheme((prev) => !prev);
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}
```



## ➤ Header.jsx

```
import { useContext } from 'react';
import { ThemeContext } from '../ThemeContext';

export default function Header() {
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <header className={`header ${theme ? 'dark' : 'light'}}>
      <h1>Theme Switcher</h1>
      <button onClick={toggleTheme}>
        {theme ? 'Light Mode' : 'Dark Mode'}로 전환
      </button>
    </header>
  );
}
```



## ➤ Content.jsx

```
import { useContext } from 'react';
import { ThemeContext } from '../ThemeContext';
export default function Content() {
  const { theme } = useContext(ThemeContext);
  return (
    <main className={`content ${theme ? 'dark' : 'light'}}>
      <p>
        현재 테마는 <strong>{theme ? 'DARK' : 'LIGHT'}</strong> 모드입니다.
      </p>
      <p>Context API와 useContext 훅으로 전역 상태를 관리하고 있어요!</p>
    </main>
  );
}
```



## ➤ ThemeLayout.jsx

```
import Header from '../components/Header';
import Content from '../components/Content';

export default function ThemeLayout() {
  return (
    <div className="app">
      <Header />
      <Content />
    </div>
  );
}
```



## ➤ App.jsx

```
import ThemeProvider from './ThemeContext';  
import ThemeLayout from './ContextAPI/pages/ThemeLayout';
```

```
function App() {  
  <ThemeProvider>  
    <ThemeLayout />  
  </ThemeProvider>  
};  
}
```



## ➤ index.css

```
/* === 리셋 === */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Pretendard', system-ui, sans-serif;
  background-color: #f4f4f4;
  color: #222;
  transition: background 0.3s ease, color 0.3s ease;
  min-height: 100vh;
}

/* === 공통 레이아웃 === */
.app {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}

header,
main {
  padding: 2rem;
  text-align: center;
}
```

```
/* === 테마 스타일 === */
.light {
  background-color: #f4f4f4;
  color: #222;
}

.dark {
  background-color: #222;
  color: #f4f4f4;
}

/* === 헤더 배경 강조 (선택사항) === */
.header.light {
  background-color: #9ed4ff;
}

.header.dark {
  background-color: #333;
}

/* === 버튼 기본 스타일 === */
button {
  border: 2px solid #fff;
  border-radius: 8px;
  padding: 0.6rem 1.2rem;
  font-size: 1rem;
  cursor: pointer;
  transition: all 0.3s ease;
}
```



## ➤ index.css

```
/* === 라이트 모드용 버튼 === */
.light button {
  background-color: #0077ff;
  color: #fff;
  border-color: #0077ff;
}

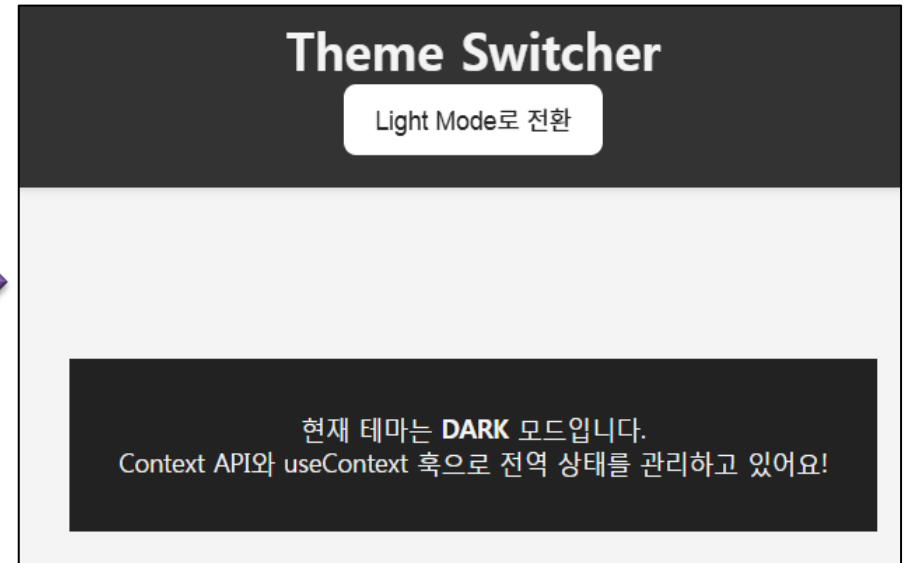
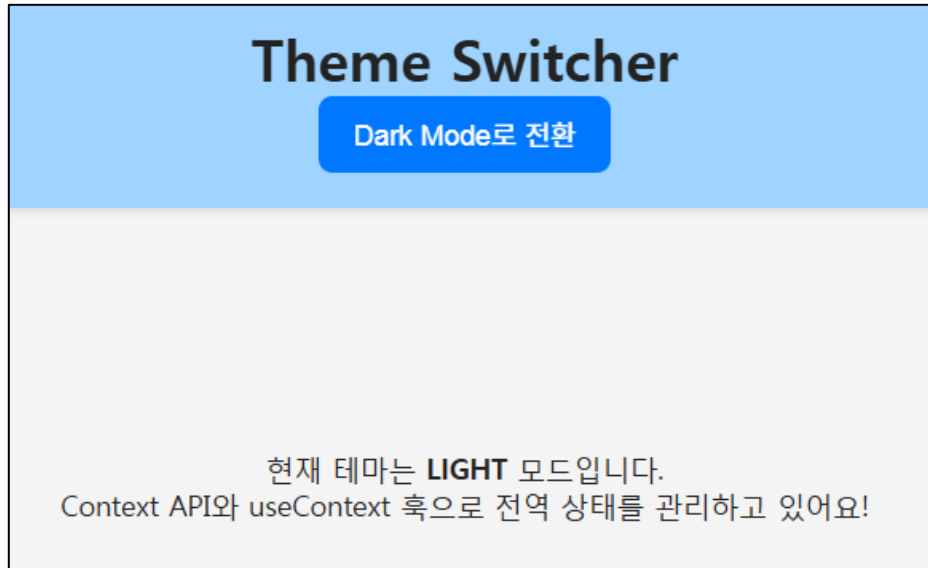
.light button:hover {
  background-color: #005ad5;
}

/* === 다크 모드용 버튼 === */
.dark button {
  background-color: #fff;
  color: #222;
  border-color: #fff;
}

.dark button:hover {
  background-color: #e0e0e0;
  color: #000;
}
```



## ➤ 출력 결과





## ➤ Context API 언제 사용할까?

- 여러 컴포넌트가 "**같은 데이터**"를 **공유**해야 할 때
- 부모 → 자식 → 손자 → 증손자로 props를 계속 전달해야 하는 상황(prop drilling)이라면  
→ Context API로 한 번에 공유하는 게 효율적 이다.

상황	설명
테마 관리 (Light / Dark)	모든 페이지나 컴포넌트에서 공통된 테마를 적용해야 할 때
로그인 정보 (사용자 상태)	로그인 상태(isLoggedIn), 사용자 이름(username), 권한(role) 등이 여러 컴포넌트에서 필요할 때
장바구니 상태 (쇼핑몰)	장바구니에 담긴 상품 목록을 상단 Navbar, 결제 페이지, 상품 상세 페이지에서 동시에 접근해야 할 때
언어 설정 / 알림 설정	사이트 전체의 언어(locale)나 알림 상태를 한 곳에서 관리할 때
게임 상태 / 전역 점수판	여러 화면 간에 점수, 남은 목숨, 진행 단계 등을 공유할 때

# 연습 문제





## 문제] 다음의 조건에 만족하도록 React를 작성 하시오.

### 조건

- ① Context API로 로그인 상태 관리하기
- ② Context API를 사용하여 로그인 상태를 전역으로 관리하도록 작성한다.
- ③ 페이지 이동 시에도 로그인 상태가 유지되도록 작성한다.
- ④ Context 폴더 생성 후 -> AuthContext.jsx 파일을 생성한다.
- ⑤ createContext()와 useState()를 이용해 user 상태를 관리한다.
- ⑥ 기본값은 null (로그인 안 된 상태)
- ⑦ login() 함수와 logout() 함수를 정의하여 상태를 변경할 수 있도록 한다.
- ⑧ AuthProvider로 App.jsx 전체를 감싼다.



## 문제] Header 컴포넌트

### 조건

- ① 상단 네비게이션 역할을 한다.
- ② 로그인 상태(user)에 따라 다른 UI를 보여준다.
- ③ 로그인 X → "홈", "로그인" 메뉴만 표시
- ④ 로그인 O → "홈", "프로필", "로그아웃" 버튼 표시
- ⑤ 로그아웃 버튼을 누르면 Context의 logout() 함수가 호출되어야 한다.



## 문제] LoginForm 컴포넌트

### 조건

- ① 사용자 이름을 입력하고 "로그인" 버튼을 누르면 로그인 처리된다.
- ② 입력한 이름으로 login(username) 함수를 호출한다.
- ③ 로그인 완료 후 /profile 페이지로 이동한다.
- ④ 이미 로그인되어 있을 경우 "이미 로그인 중입니다." 메시지를 보여준다.



## 문제] Profile 컴포넌트

### 조건

- ① 로그인된 사용자의 이름을 보여준다.
- ② 예: "안녕하세요, 홍길동 님!"
- ③ 로그인되지 않은 상태라면 다음 문구를 출력한다.  
예) "로그인이 필요합니다."
- ④ 로그인하지 않은 상태에서 접근 시, "로그인 페이지로 이동" 버튼을 제공한다.



<초기 화면 상태>

# Context 인증 실습

홈로그인

로그인이 필요합니다.

로그인 하러 가기

<로그인 하러 가기 상태>

# Context 인증 실습

홈로그인

로그인

홍길동

로그인

<로그인 후 상태>

# Context 인증 실습

홈프로필

로그아웃

프로필 페이지

안녕하세요, 홍길동 님!

오늘도 좋은 하루 되세요 🌻