



21. Shop Front

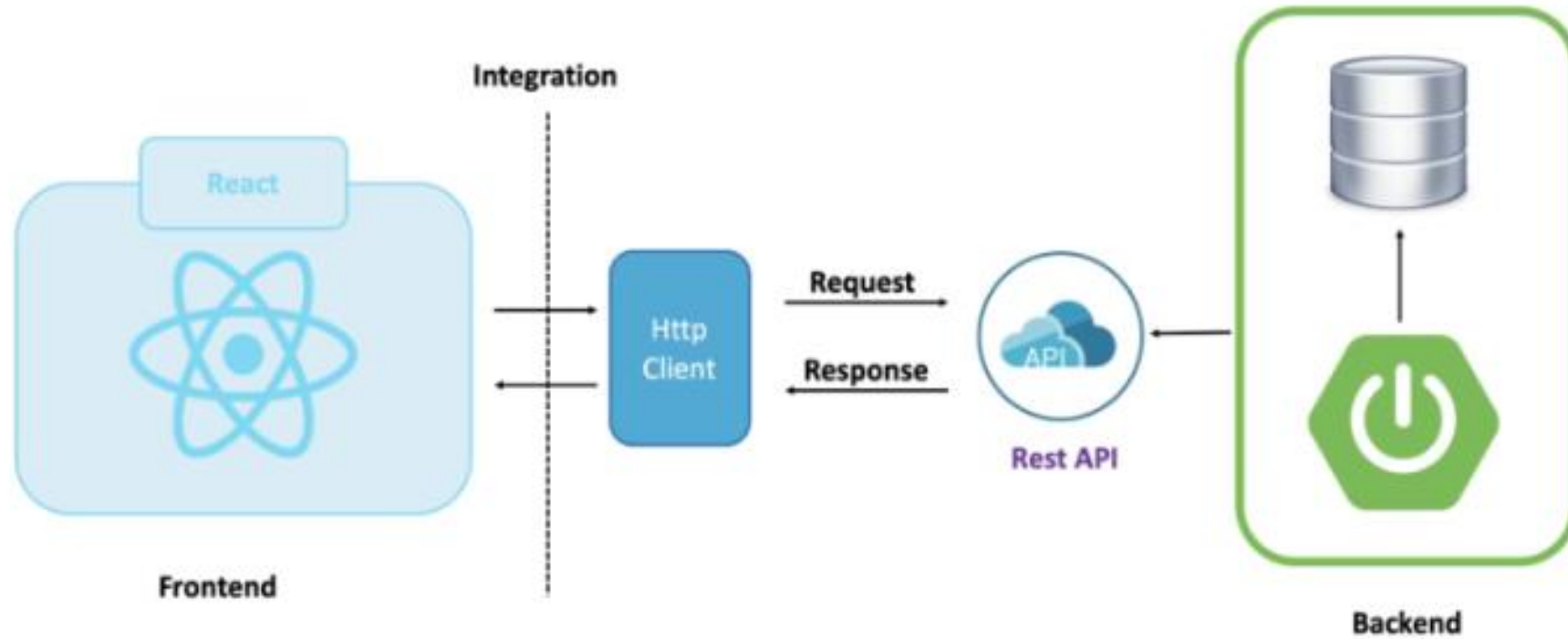


contents

- ▶ 프로젝트 생성 및 설정
- ▶ 공통 컴퍼넌트
- ▶ 모델
- ▶ Axios와 auth service
- ▶ 리덕스
- ▶ Register 처리
- ▶ 로그인 처리
- ▶ Admin Page: 상품 입력, 수정, 삭제
- ▶ 구매 처리
- ▶ User 서비스



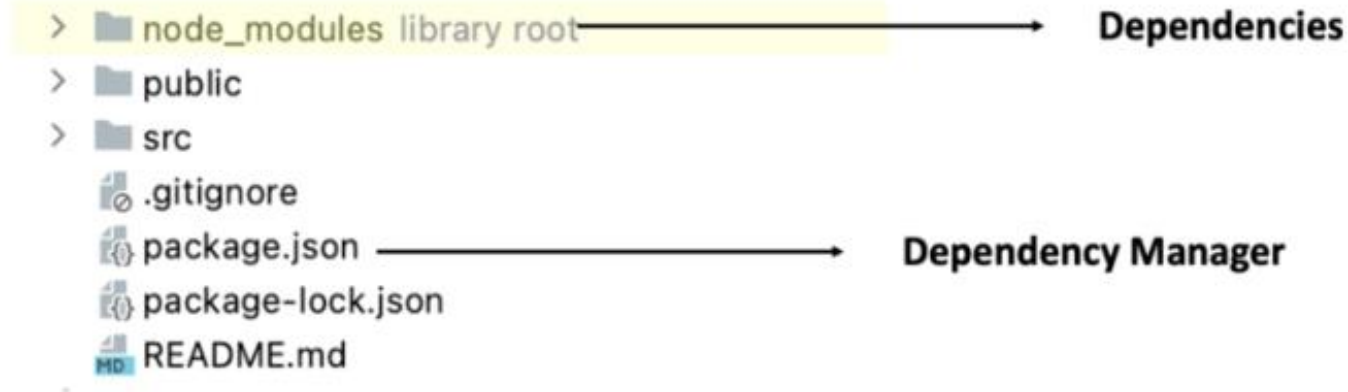
프론트 백엔드 구조



1. 프로젝트 생성 및 설정

▶ 새 프로젝트 생성 및 구조

`npx create-react-app react-shop`

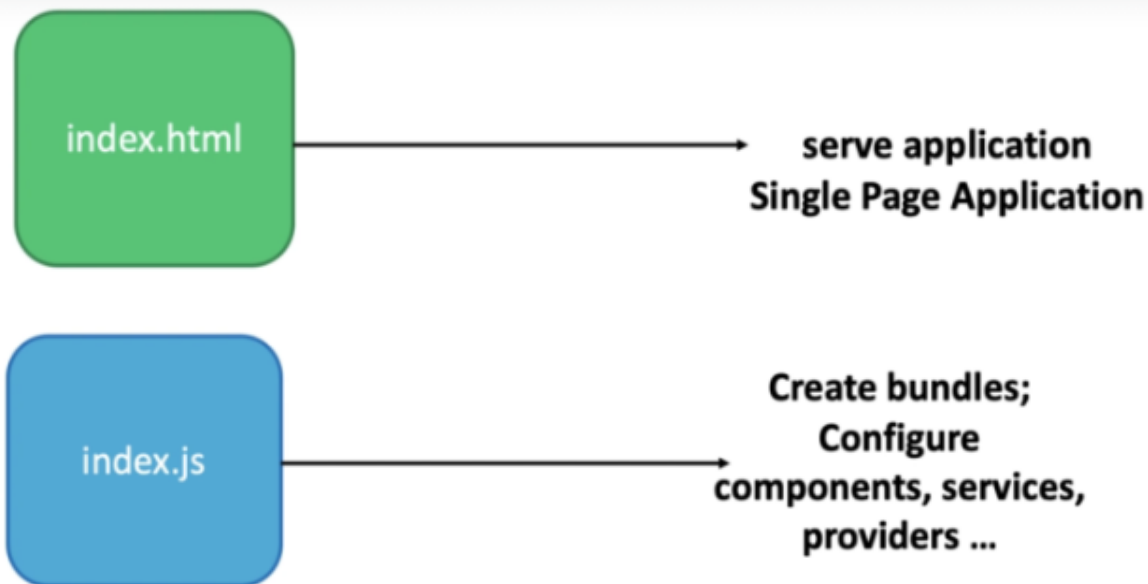


- **node_modules** : npm install을 하여 설치된 모듈들이 위치하는 폴더
- **public** : static 자원 폴더, 정적 파일들을 위한 폴더
- **src** : 리액트를 작업할 폴더



1. 프로젝트 생성 및 설정

▶ 새 프로젝트 생성 및 구조



public 폴더

favicon.icon

파비콘을 만들어주는 사이트.

<https://www.degraeve.com/favicon/>

<http://tools.dynamicdrive.com/favicon/>

index.html

가상 DOM을 위한 html 파일. (싱글 페이지)

메인 프로그램인 index.js에 대응되는 것으로, HTML 템플릿 파일.

index.js에 의해 동적으로 렌더링된 결과 표시.

=> 실제로 runtime으로 화면을 렌더링 하므로 실제 html 코드는 검사를 해도 기본 index.html 내용만 나옴.

1. 프로젝트 생성 및 설정

▶ index.js

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

root.render(컴포넌트, 위치)

첫 번째 인자에는 일반적인 HTML의 태그처럼 생긴 Component 들을 'root'라는 id를 가진 element에 띄워 달라는 의미

React.StrictMode

StrictMode는 애플리케이션 내의 잠재적인 문제를 알아내기 위한 도구.

Fragment와 같이 UI를 렌더링 하지 않으며, 자손들에 대한 부가적인 검사와 경고를 활성화함.
Strict 모드는 개발 모드에서만 활성화되기 때문에, 프로덕션 빌드에는 영향을 끼치지 않는다.

App

App.js에서 만든 Component를 띄움.



1. 프로젝트 생성 및 설정

▶ 라이브러리 설치

▶ bootstrap

```
npm install react-bootstrap bootstrap
```

▶ 폰트오썸 아이콘 디펜던시 추가

```
npm i --save @fortawesome/fontawesome-svg-core  
npm i --save @fortawesome/free-solid-svg-icons  
npm i --save @fortawesome/react-fontawesome
```

▶ axios

```
npm i axios
```

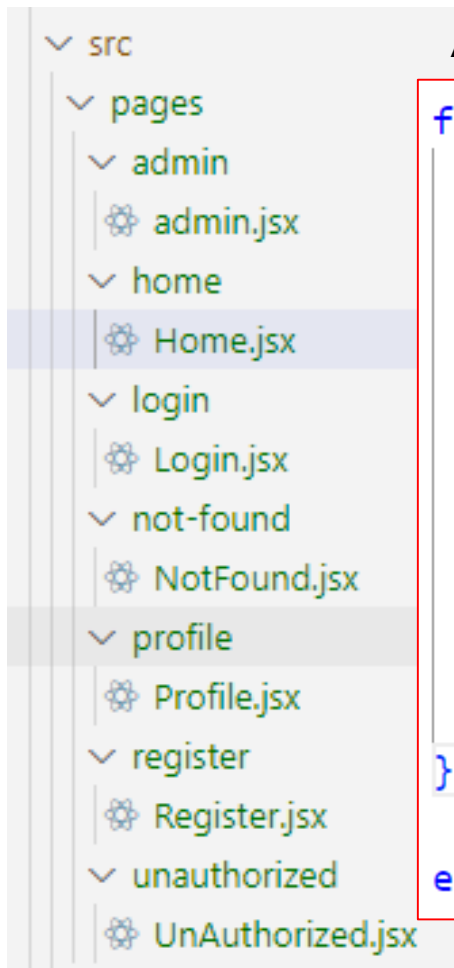
▶ React Router

```
npm i react-router-dom
```



1. 프로젝트 생성 및 설정

▶ 소스 페이지 분리하기



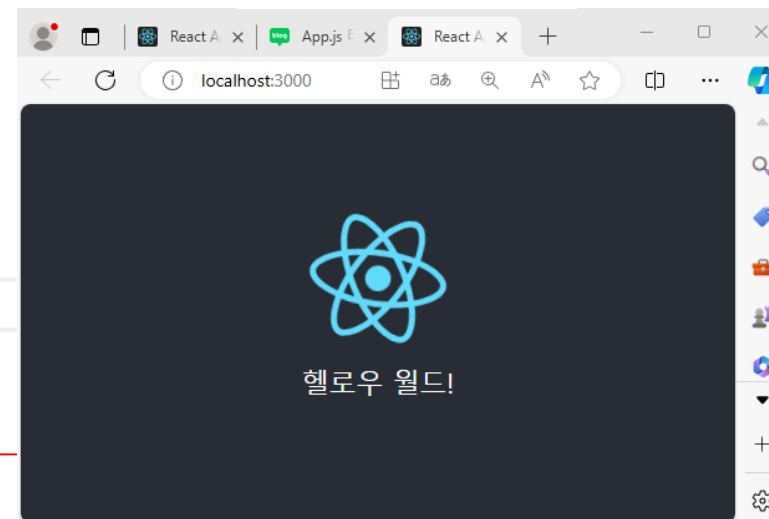
App.js

```
function App() {  
  return (  
    <div className="App">  
      <header className="App-header">  
        <img src={logo} className="App-logo" alt="logo" />  
        <p>  
          헬로우 월드!  
        </p>  
      </header>  
    </div>  
  );  
}  
  
export default App;
```

Admin.jsx

```
import React from "react";  
  
const Admin = () =>{  
  return <div>관리자 페이지</div>  
}  
  
export default Admin;
```

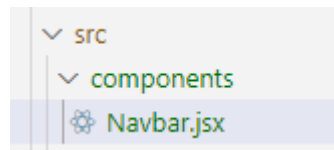
npm start 결과



2. 공통 컴포넌트 작성

▶ Navbar

- ▶ 페이지마다 사용할 공통 컴포넌트



```
import logo from '../logo.svg';
const Navbar = () => {
  return (
    <nav className='navbar navbar-expand navbar-dark bg-dark'>
      <a href='https://react.dev' className='navbar-brand ms-1'>
        <img src={logo} className='App-logo' alt='logo' />
        React
      </a>
    </nav>
  );
};
export default Navbar;
```

2. 공통 컴포넌트 작성

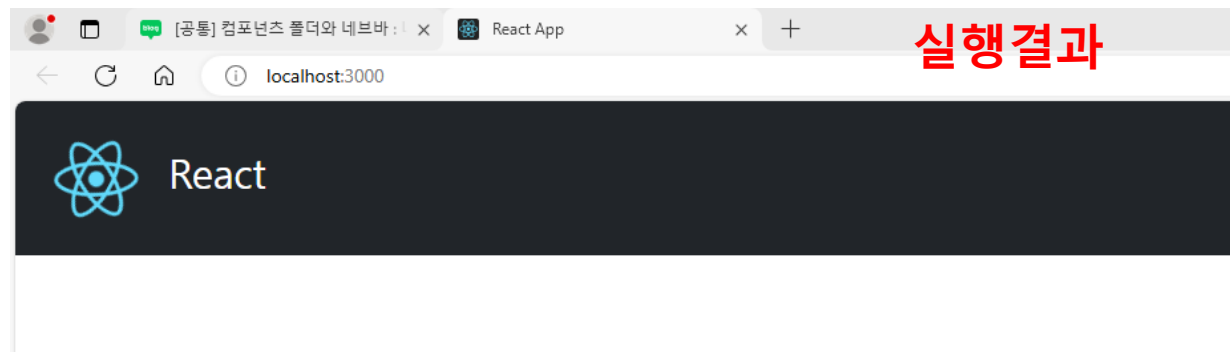
▶ Navbar

App.js

```
function App() {  
  return (  
    <div className="App">  
      <Navbar />  
    </div>  
  );  
}  
  
export default App;
```

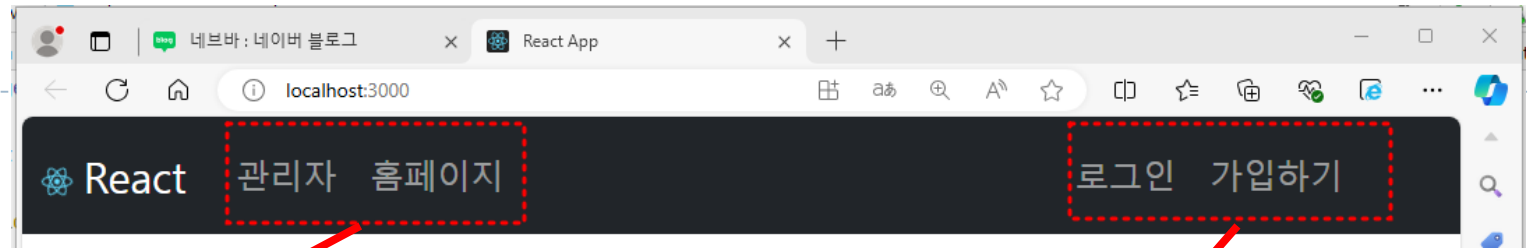
App.css

```
.App-logo {  
  height: 8vmin;  
  pointer-events: none;  
}
```



2. 공통 컴포넌트

▶ Navbar



```
<div className='navbar-nav me-auto'>
  <li className='nav-item'>
    <a href='##' className='nav-link'>
      관리자
    </a>
  </li>
</div>
```

```
<div className='navbar-nav ms-auto me-5'>
  <li className='nav-item'>
    <a href='##' className='nav-link'>
      로그인
    </a>
  </li>
</div>
```

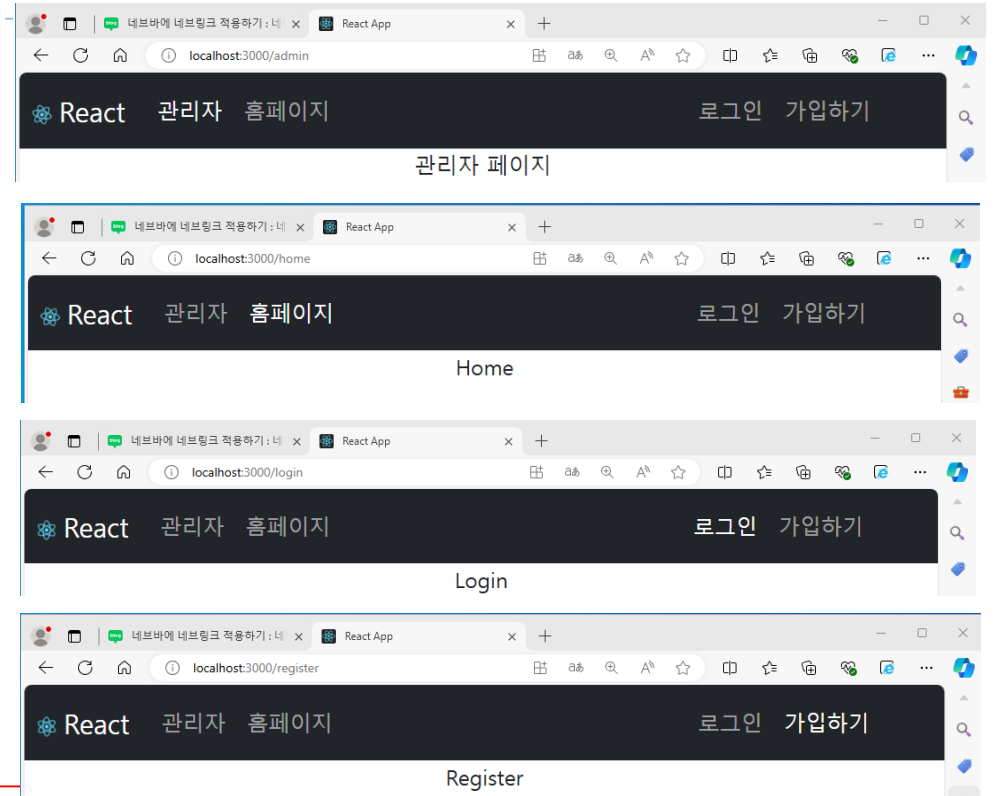
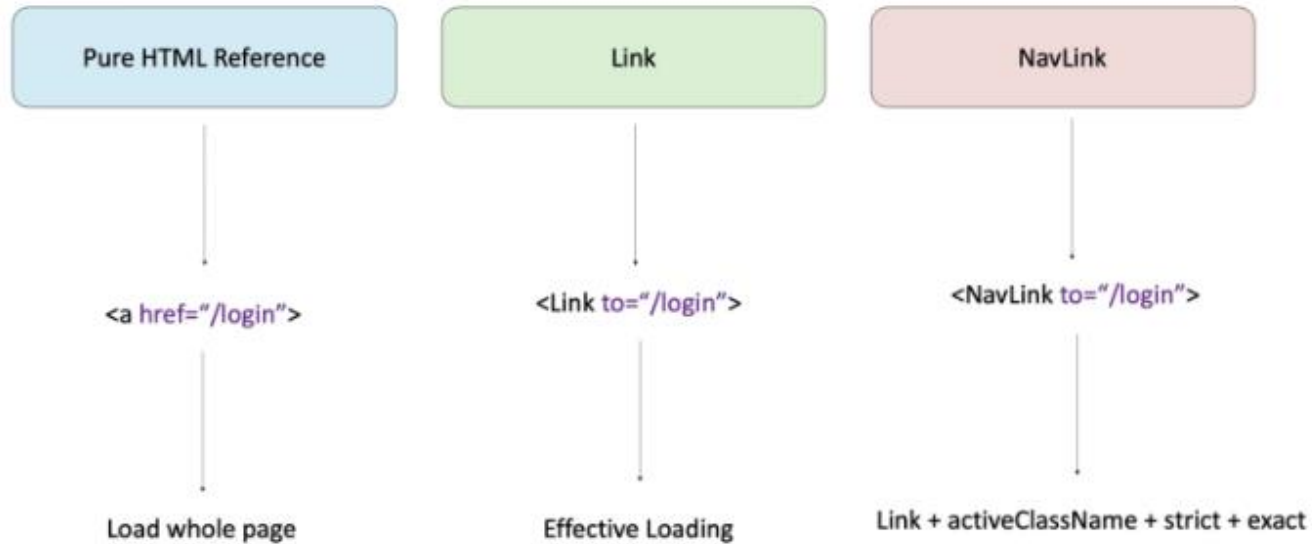
2. 공통 컴포넌트

▶ 라우터 적용

```
function App() {  
  return (  
    <div className="App">  
      <BrowserRouter>  
        <Navbar />  
        <div className="container">  
          <Routes>  
            <Route path="/" element={<Home />}></Route>  
            <Route path="/home" element={<Home />}></Route>  
            <Route path="/login" element={<Login />}></Route>  
            <Route path="/register" element={<Register />}></Route>  
            <Route path="/admin" element={<Admin />}></Route>  
            <Route path="/404" element={<NotFound />}></Route>  
            <Route path="/401" element={<Unauthorized />}></Route>  
            <Route path="*" element={<NotFound />}></Route>  
          </Routes>  
        </div>  
      </BrowserRouter>  
    </div>  
  );  
}
```

2. 공통 컴포넌트

▶ Navbar 링크 적용



▶ a태그를 NavLink로 변경

```
<li className='nav-item'>  
  <NavLink to="/admin" href='##' className='nav-link'>  
    관리자  
  </NavLink>  
</li>
```

2. 공통 컴포넌트

▶ 링크, 네브링크 공통점과 차이점

- ▶ 공통점 : Link와 NavLink 두 컴포넌트 모두 화면에 태그로 렌더링 되고, to 속성에 정의해 둔 url로 이동한다.
- ▶ NavLink 공식 문서에 다음과 같이 정의 됨
 - ▶ NavLink 는 Link의 특별한 버전입니다.
 - ▶ 현재 URL과 클릭시 이동할 URL이 같을 경우, "style 속성" 을 추가해주는 기능을 가지고 있습니다.
 - ▶ 스타일 외에도 추가 기능으로 선택된 NavLink의 경우 aria-current="page" 로 지정되어, 접근성에 도움이 됩니다.



2. 공통 컴포넌트

▶ NotFound 페이지

```
const NotFound=() =>{
  return (
    <div className='container'>
      <div className='row'>
        <div className='col-md-12 text-center'>
          <span className='display-1'>404</span>
          <div className='mb-4 lead'>주소가 변경되었거나 페이지를 찾지 못했습니다!</div>
          <Link to='/home' className='btn btn-link'>
            Back to Home
          </Link>
        </div>
      </div>
    </div>
  )
}
export default NotFound;
```

▶ Unauthorized 페이지


```
const Unauthorized=() =>{  
  return (  
    <div className='container'>  
      <div className='row'>  
        <div className='col-md-12 text-center'>  
          <span className='display-1'>401</span>  
          <div className='mb-4 lead'>권한없음! 이 주소로 접근이 거부되었습니다.</div>  
  
          <Link to='/home' className='btn btn-link'>  
            Back to Home  
          </Link>  
        </div>  
      </div>  
    </div>  
  )  
}  
export default Unauthorized;
```


3. Model

▶ Model User와 Role

```
public class User
{
    private Long id;
    private String username;
    private String password;
    private String name;
    private LocalDateTime createTime;
    private Role role;
    private String token;
}

public enum Role
{
    USER,
    ADMIN,
}
```



A diagram showing a reference from the 'role' field in the User class to the Role enum. A line starts from the 'role' field in the User class and ends with an arrow pointing to the Role enum.



3.Model

▶ Model User 와 Role

▼ model

JS role.js

JS User.js

```
export default class User{  
  constructor(username, password, name, role, token, id){  
    this.username=username;  
    this.password=password;  
    this.name=name;  
    this.role=role;  
    this.token=token;  
    this.id=id;  
  }  
}
```

```
export const Role={  
  USER:'USER',  
  ADMIN:'ADMIN'  
}
```



3. Model

▶ 모델 Product 와 Purchase

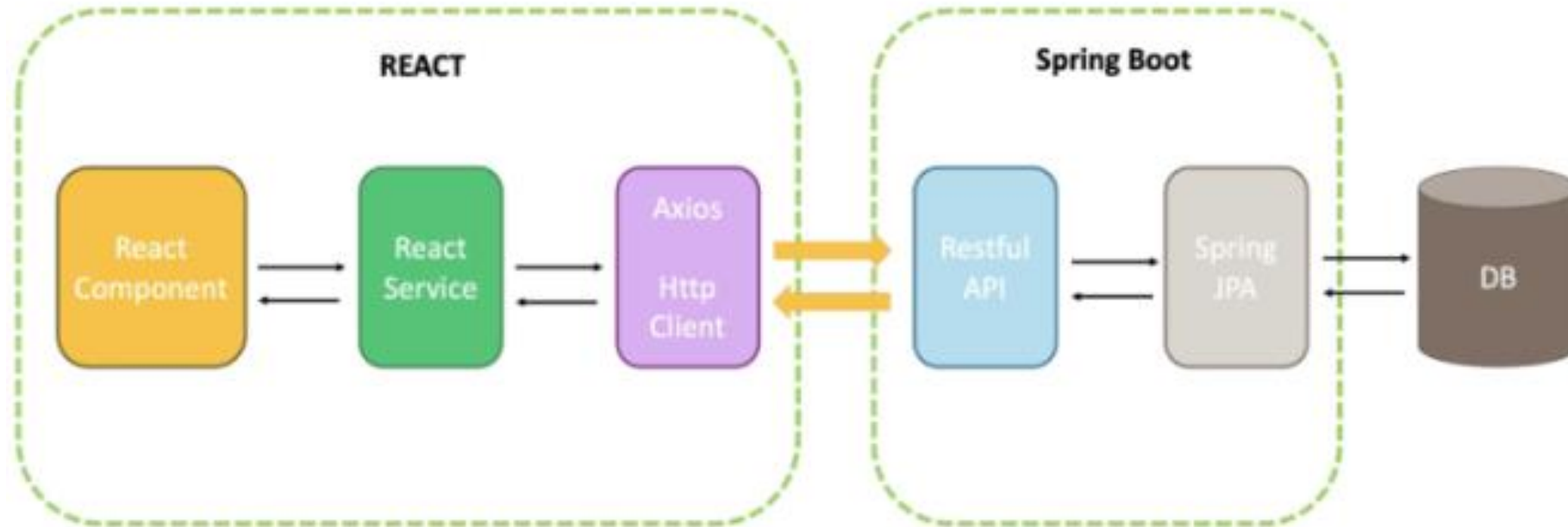
```
export default class Product {  
  constructor(name, description, price, createTime, id) {  
    this.name = name;  
    this.description = description;  
    this.price = price;  
    this.createTime = createTime;  
    this.id = id;  
  }  
}
```

```
export default class Purchase {  
  constructor(userId, productId, quantity, purchaseTime, id) {  
    this.userId = userId;  
    this.productId = productId;  
    this.quantity = quantity;  
    this.purchaseTime = purchaseTime;  
    this.id = id;  
  }  
}
```



4. Axios와 auth service

- ▶ Axios 와 auth.service.js



4. Axios와 auth service

▶ auth.service.js

Services

JS auth.service.js

```
import axios from "axios";
import { BASE_API_URL } from "../common/constants";

const BASE_URL = BASE_API_URL + '/api/authentication';

const loginService = (user) => {
  return axios.post(BASE_URL + '/sign-in', user);
}

const registerService = (user) => {
  return axios.post(BASE_URL + '/sign-up', user);
}

export {loginService, registerService}
```

common

JS constants.js

```
export const BASE_API_URL = 'http://localhost:8090';
```



5. 리덕스

▶ Redux 란?

- ▶ 컴포넌트의 개수가 많은 대규모 프로젝트에서 상태 값을 효율적으로 관리할 수 있는 라이브러리

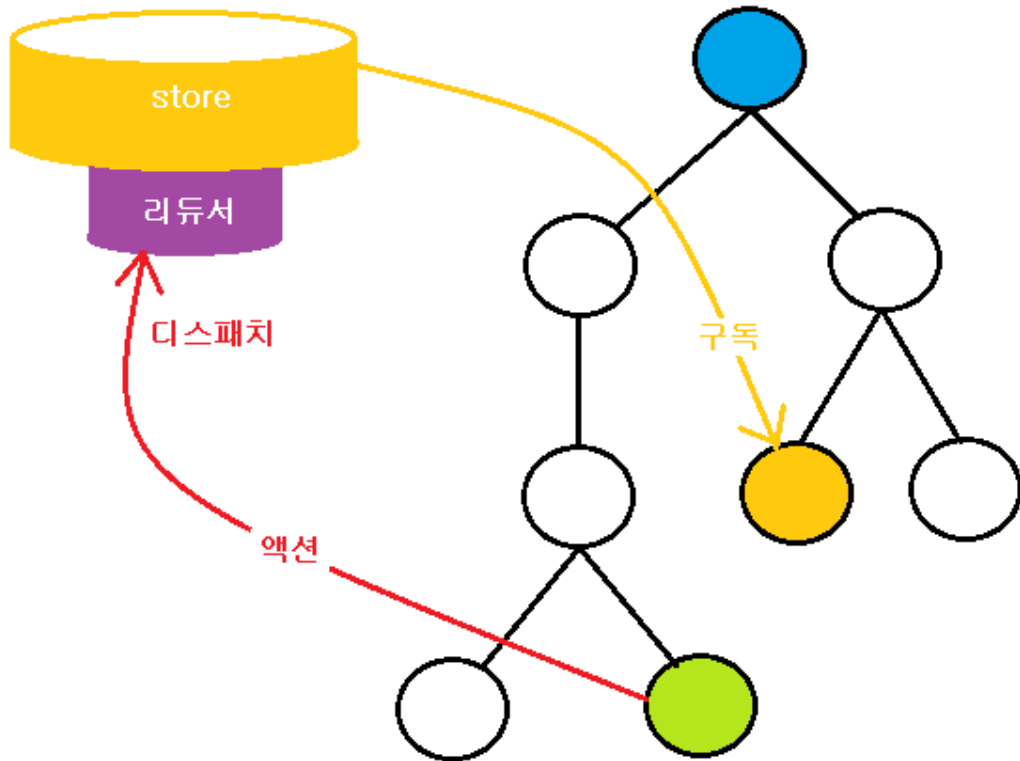
▶ Redux 개념

- ▶ 상태 관리 로직을 컴포넌트 밖에서 처리
- ▶ 리덕스를 사용하면 스토어(store)라는 객체 내부에 상태 값을 담게 된다.
- ▶ 이를 사용하면 컴포넌트가 많은 복잡한 형태의 프로젝트에서 상태 값에 연관된 모든 컴포넌트들을 업데이트 하고 리렌더링 할 필요 없이 바로 전달할 수 있게 된다는 것이다.



5. 리덕스

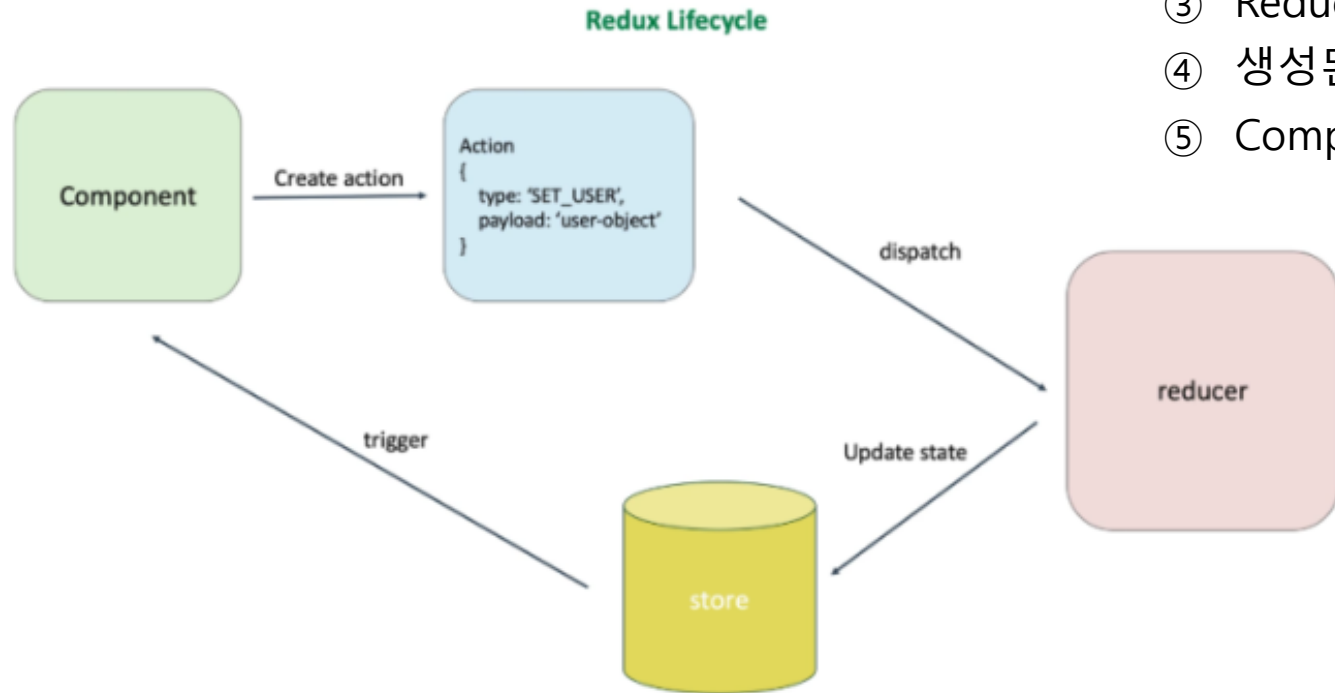
리덕스를 적용한 구조



- a. 스토어: 애플리케이션의 상태 값들을 내장하고 있다. 스토어는 하나만 존재한다.
- b. 액션: 상태 변화를 일으킬 때 참조하는 객체.
- c. 디스패치: 액션을 스토어에 전달하는 것을 의미한다.
- d. 리듀서: 상태를 변화시키는 로직이 있는 함수이다.
- e. 구독: 스토어 값이 필요한 컴포넌트는 스토어를 구독한다

5. 리덕스

▶ Redux Lifecycle



리덕스 상태관리 흐름

- ① 리덕스 Store을 Component에 연결
- ② Component에서 상태 변화가 필요할 때 Action을 부름
- ③ Reducer을 통해 새로운 상태 값 생성
- ④ 생성된 상태값을 Store에 저장
- ⑤ Component가 Store에서 새로운 상태 값을 받아옴

Store : 리덕스를 적용하기 위한 공간
Reducer : 저장된 데이터를 변경하는 함수

State
Action
ActionCreator
Reducer
Store
dispatch

5. 리덕스

▶ 관련 용어-I

- ▶ Store : 리덕스를 적용하기 위한 공간
- ▶ Reducer : 저장된 데이터를 변경하는 함수

▶ **State :**

- ▶ 리덕스에서 저장하고 있는 상태 값(데이터)
- ▶ 사전 형태({[key]:value})로 보관함.

▶ **Action**

- ▶ 상태 변화가 필요할 때(가지고 있는 데이터를 변경할 때) 발생

```
{type: 'change_state', data:{...}}
```

- ▶ 액션은 객체, type는 이름 같은 것이고, 임의의 문자열을 정해서 data에 넣는다.



5. 리덕스

▶ 관련 용어-I

▶ Reducer

- ▶ 리덕스에 저장된 상태(데이터)를 변경하는 함수입니다.
- ▶ 액션 생성 함수를 부르고 → **액션을 만들고** → 리듀서가 **현재 상태와 액션 객체를 받아서** → **새로운 데이터를 만들고** → 돌려줍니다.

▶ Store

- ▶ 프로젝트에 리덕스를 적용하기 위한 것
- ▶ 현재 상태와 리듀서랑 상호작용하는 내장 함수가 포함되어 있음

▶ dispatch

- ▶ Store의 내장 함수 중 하나.
- ▶ **Action을 발생** 시키는 역할 수행.

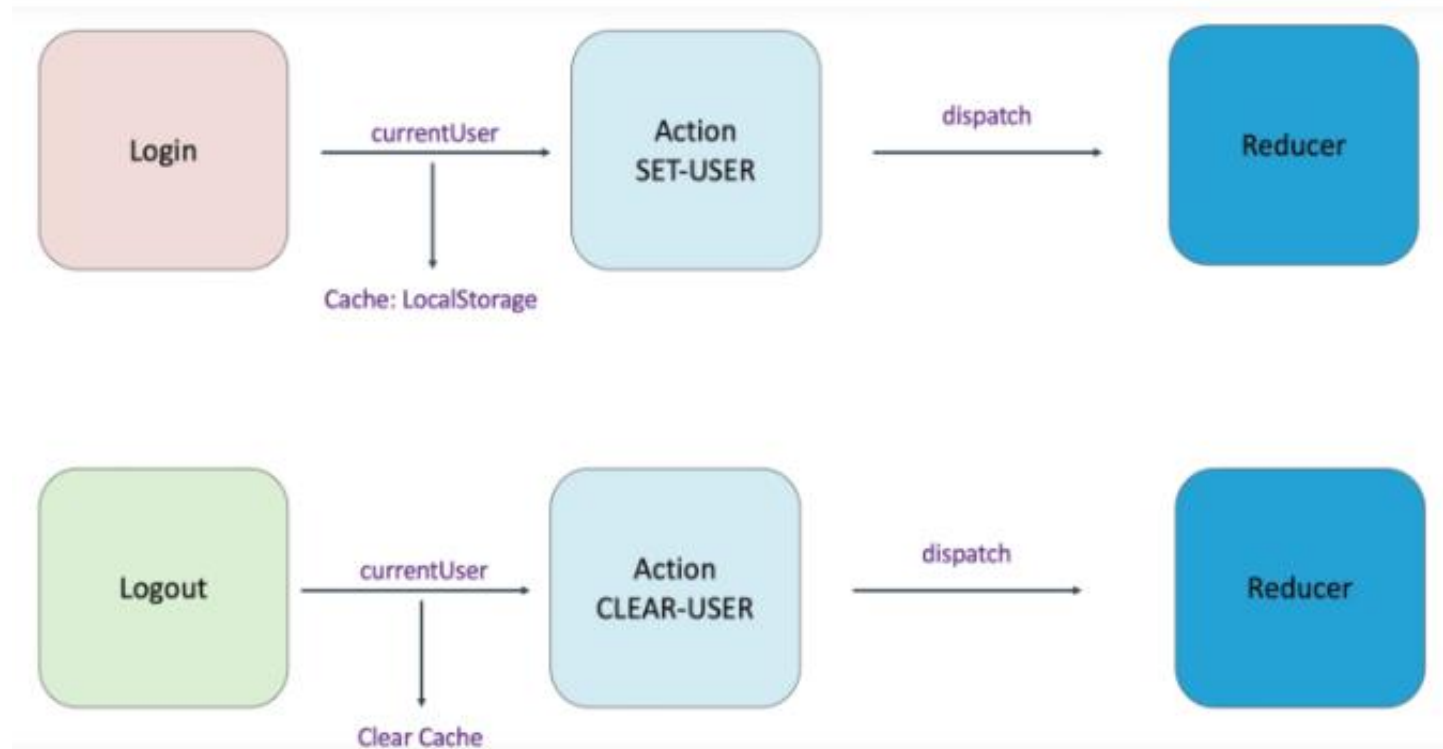
reducer 예

```
const initialState = {
  name: 'mar'
}

function reducer(state = initialState, action) {
  switch(action.type){
    // action의 타입마다 케이스문을 걸어주면,
    // 액션에 따라서 새로운 값을 돌려줍니다!
    case CHANGE_STATE:
      return {name: 'mar1'};
    default:
      return false;
  }
}
```

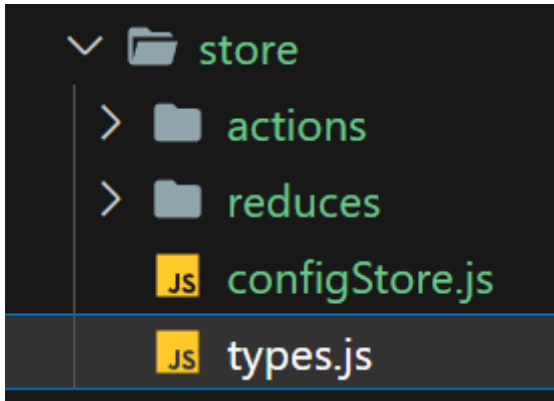
5. 리덕스

- ▶ 리덕스 설치 : `npm i redux react-redux`
- ▶ 로그인과 로그아웃 시 사용



5. 리덕스

- ▶ 리덕스 파일들 구조 (액션파일들과 리듀서파일들을 분리)



```
export const SET_CURRENT_USER = 'SET_CURRENT_USER';  
export const CLEAR_CURRENT_USER = 'CLEAR_CURRENT_USER';
```



5. 리덕스

▶ 액션과 리듀서 작성

▶ actions > user.js

```
import { CLEAR_CURRENT_USER, SET_CURRENT_USER } from "../types";

export const setCurrentUser=(user) => {
  return {
    type: SET_CURRENT_USER,
    payload:user // 수정
  }
}

export const clearCurrentUser =() =>{
  return {
    type: CLEAR_CURRENT_USER,
  }
}
```



5. 리덕스

▶ reducers > user.js

```
import { CLEAR_CURRENT_USER, SET_CURRENT_USER } from "../types";


const userReducer=(state={}, action) => {
  switch (action?.type){
    case SET_CURRENT_USER: SET_CURRENT_USER => 로컬저장소에 유저정보(payload)를 저장하고 유저정보를 업데이트
      localStorage.setItem('currentUser', JSON.stringify(action?.payload));
      return action?.payload;
    case CLEAR_CURRENT_USER: CLEAR_CURRENT_USER => 유저정보를 로컬저장소에서 제거하고 null 값으로 업데이트
      localStorage.removeItem("currentUser");
      return null;
    default: 액션타입의 값이 없을 경우 디폴트 => 로컬저장소에 저장된 유저정보를 가져옴(캐싱된 데이터)
      return JSON.parse(localStorage.getItem('currentUser'))
  }
};

export default userReducer;
```

옵셔널 체이닝 '?.': 옵셔널 체이닝(optional chaining) ?.을 사용하면 프로퍼티가 없는 중첩 객체를 에러 없이 안전하게 접근할 수 있음

5. 리덕스

- ▶ 리듀서들로 스토어 만들고 index.js에 적용하기



```
import {combineReducers, legacy_createStore as createStore} from 'redux'
import userReducer from './reduces/user'

const allReducers=combineReducers({
  user:userReducer
})

const store=createStore(allReducers);

export default store;
```

여러 리덕스를 결합



5. 리덕스

- ▶ 리듀서들로 스토어 만들고 index.js에 적용하기

```
6 import store from './store/configStore';
7 import { Provider } from 'react-redux';
8
9 const root = ReactDOM.createRoot(document.getElementById('root'));
10 root.render(
11   <React.StrictMode>
12     <Provider store={store} >
13       <App />
14     </Provider>
15   </React.StrictMode>
16 );
```


6. Register & Login

▶ Register-I

```
import { useEffect, useState } from 'react';
import User from '../../models/User';
import { useNavigate } from 'react-router-dom';
import { useSelector } from 'react-redux';
import { registerService } from '../../service/auth.service';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faUserCircle } from '@fortawesome/free-solid-svg-icons';
import './Register.css'

const Register=()=>{
  const [user, setUser] = useState(new User('', '', ''));
  const [loading, setLoading] = useState(false);
  const [submitted, setSubmitted] = useState('');
  const [errorMessage, setErrorMessage] = useState('');
}
```

- useState로 사용할 Hook들을 정리.
- useSelector는 리액트의 리덕스 스토어 관련 Hook중 하나로 이 Hook은 스토어의 상태 값을 반환해주는 역할 수행



6. Register & Login

▶ Register-2

```
const currentUser = useSelector((state) => state.user);
const navigate = useNavigate();
```

```
useEffect(() => {
  if (currentUser?.id) {
    navigate('/profile');
  }
}, []);

const handleChange = (e) => {
  const { name, value } = e.target;
  console.log(name, value);
  setUser((prevState) => {
    return {
      ...prevState,
      [name]: value,
    };
  });
};
```

- useEffect 는 [] 시작시 위의 useSelector로 받아온 유저정보가 있으면 profile 페이지로 이동
- 입력창에 내용을 새로 적거나 수정시 이벤트
- 입력창에 이름과 value 값으로 user 객체의 값을 업데이트
- 만약 입력창이 name 이면 user.name 의 값이 업데이트



6. Register & Login

▶ Register-3

```
import { registerService } from '../Services/auth.service';
```

```
const handleRegister=(e) =>{
  e.preventDefault();
  setSubmitted(true);

  if (!user.username || !user.password || !user.name){
    return;
  }

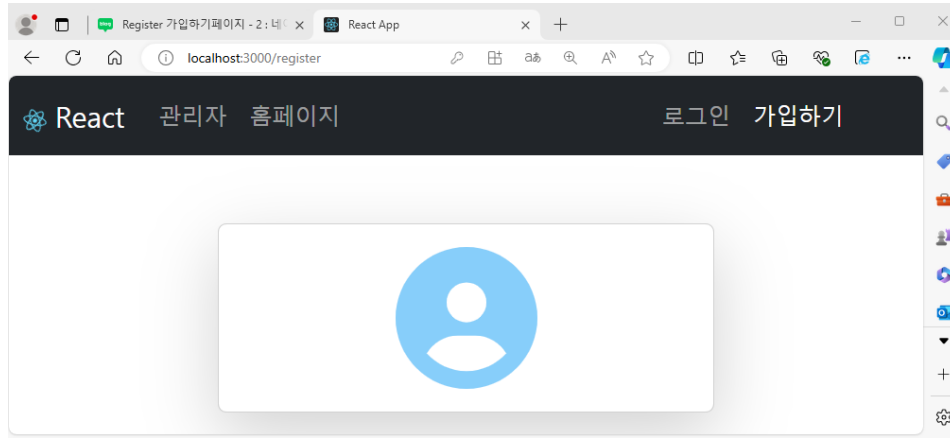
  setLoading(true);

  registerService(user)
    .then((ok) => {
      navigate("/login")
    })
    .catch((error)=>{
      console.log(error);
      if(error?.response?.status ===409){
        setErrorMessage("username 또는 password가 틀립니다.");
      }else{
        setErrorMessage("예상하지 못한 에러가 발생했습니다.")
      }
      setLoading(false);
    });
}
```

- 가입하기 버튼 클릭시 이벤트
- 버튼 클릭시 submitted의 값을 참으로 업데이트
- 유저네임, 패스워드,이름 등이 입력되지 않았을 경우 바로 리턴
- loading 값을 참으로 업데이트
- 미리 작성한 registerService에 user 입력 성공시 /login 페이지로
- 실패시 에러상태에 따라 에러 메세지 출력

6. Register & Login

▶ Register-4



Register.css

```
.custom-card {  
  width: 350px;  
  max-width: 100%;  
}  
  
.user-icon {  
  font-size: 100px;  
  color: lightskyblue;  
}  
  
form label {  
  text-indent: 10px;  
}
```

```
return (  
  <div className='container mt-5'>  
    <div className='card ms-auto me-auto p-3 shadow-lg custom-card'>  
      <FontAwesomeIcon icon={faUserCircle} className='ms-auto me-auto user-icon' />  
      {errorMessage && <div className='alert alert-danger'>{errorMessage}</div>}  
  
    </div>  
  </div>  
)
```

6. Register & Login

▶ Register-5

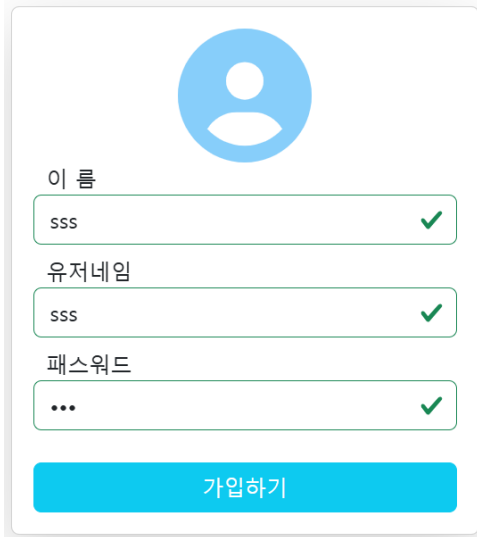
```
return(  
  <div className='container mt-5'>  
    <div className='card ms-auto me-auto p-3 shadow-lg custom-card'>  
      <FontAwesomeIcon icon={faUserCircle} className='ms-auto me-auto user-icon' />  
  
      {errorMessage && <div className='alert alert-danger'>{errorMessage}</div>}  
  
      <form onSubmit={handleRegister} noValidate className={submitted ? 'was-validated':''}>  
  
        <div className='form-group mb-2'>  
          <label htmlFor='name'>이름</label>  
          <input type='text' name='name' className='form-control'  
            placeholder='name' value={user.name} onChange={handleChange} required />  
          <div className='invalid-feedback'>이름을 입력해주세요</div>  
        </div>  
        ...  
        <button className='btn btn-info text-white w-100 mt-3' disabled={loading}>  
          가입하기  
        </button>  
      </form>  
    </div>  
  </div>  
)
```

- 폼 위에 에러 메시지 표시 (에러가 있을 시)
- 부트스트랩 자체 유효성검사를 하기 위해 form 에 속성 noValidate (자체 검사로 html 검사 제외)
- class에 was-validate가 있으면 검사하여 입력창 하단의 'invalid-feedback'에 글자가 표시됨
- className={submitted ? => 가입하기 버튼을 눌렀을 경우에 검사함 (CSS만으로 검사)

6. Register & login

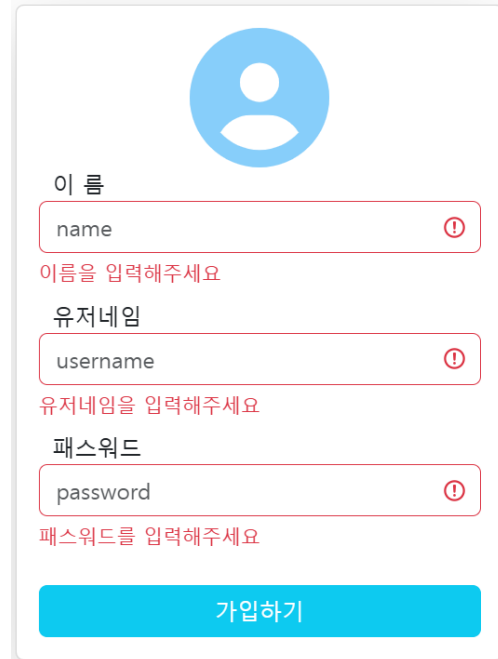
▶ Register-6

초기화면



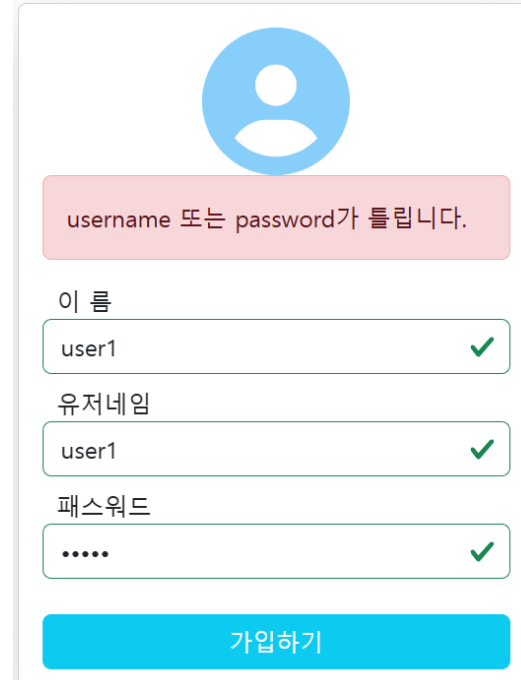
Initial registration form with a blue user icon. Fields: 이름 (ssss), 유저네임 (ssss), 비밀번호 (.....). All fields have green checkmarks. A blue '가입하기' button is at the bottom.

name, username, password 미입력 한 상태에서 가입하기 버튼 클릭 했을 시



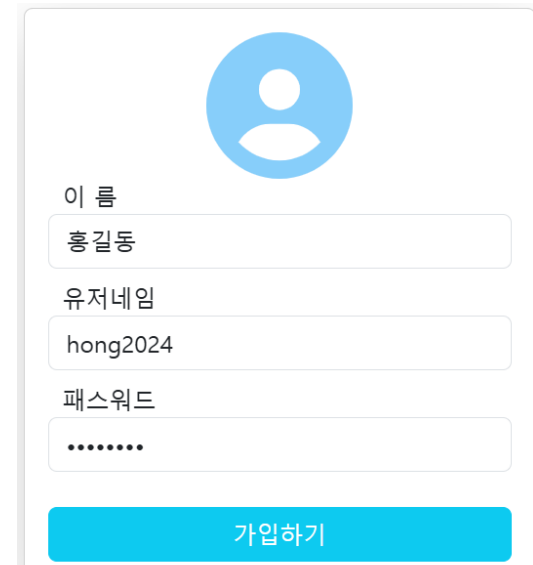
Registration form with validation errors. Fields: 이름 (name), 유저네임 (username), 비밀번호 (password). Each field has a red border and a red exclamation mark icon. Red text messages are shown below each field: '이름을 입력해주세요', '유저네임을 입력해주세요', '패스워드를 입력해주세요'. A blue '가입하기' button is at the bottom.

username 중복된 경우

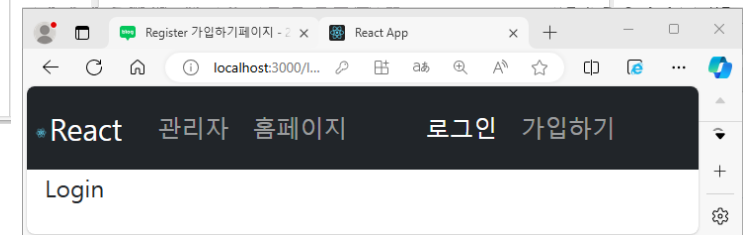


Registration form with a duplicate username error. A red message box at the top says 'username 또는 password가 틀립니다.'. Fields: 이름 (user1), 유저네임 (user1), 비밀번호 (.....). All fields have green checkmarks. A blue '가입하기' button is at the bottom.

가입이 정상적으로 처리된 경우



Registration form with successful registration. Fields: 이름 (홍길동), 유저네임 (hong2024), 비밀번호 (.....). All fields are filled. A blue '가입하기' button is at the bottom.



6. Register & Login

► Login-I

```
import { useState } from "react";
import { useSelector } from "react-redux";
import { useNavigate } from "react-router-dom";

const Login=() =>{
  const [user, setUser] = useState(new User('', '', ''));
  const [loading, setLoading] = useState(false);
  const [submitted, setSubmitted] = useState(false);
  const [errorMessage, setErrorMessage] = useState('');

  const currentUser = useSelector((state) => state.user);
  const navigate = useNavigate();

  return (
    <div>
      Login
    </div>
  )
}
export default Login;
```



6. Register & Login

▶ Login-2

- 디스패치는 리덕스 스토어의 내장함수 중 하나로 액션을 발생 시킴
- dispatch 라는 함수에는 액션을 파라미터로 전달: dispatch(action)
- dispatch 호출을 하면, 스토어는 리듀서 함수를 실행시켜서 해당 액션을 처리하는 로직이 있다면 액션을 참고하여 새로운 상태를 만들어 줌.

```
const dispatch = useDispatch();
```

```
useEffect(() => {  
  if (currentUser?.id) navigate('/profile');  
  // 이미 유저정보가 입력된(로그인된) 때에는 /profile 페이지로 이동 (페이지 처음 1회 실행)  
}, []);
```



6. Register & Login

▶ Login-3

```
const handleChange = (e) => {  
  const { name, value } = e.target;  
  console.log(name, value);  
  setUser((prevState) => {  
    return {  
      ...prevState,  
      [name]: value,  
    };  
  });  
};
```

로그인 입력창에 입력시 이벤트
입력내용들을 바로 업데이트함



6. Register & Login

▶ Login-4

```
const handleLogin = (e) => {  
  e.preventDefault();  
  setSubmitted(true);  
  
  if (!user.username || !user.password) {  
    return;  
  }  
  
  setLoading(true);  
  
  loginService(user)  
    .then((response) => {  
      //setCurrentUser로 만든 액션을 유저 리듀서에 전달  
      dispatch(setCurrentUser(response.data));  
      navigate('/profile');  
    })  
    .catch((error) => {  
      console.log(error);  
      setErrorMessage('유저네임 또는 비밀번호가 틀립니다.');
```

- submit 이벤트 발생시
- 기본동작 중지 e.preventDefault();
- 유저이름 또는 패스워드 입력하지 않았을 경우에 바로 리턴하여 종료
- 로그인서비스로 백엔드에 user 객체와 함께 요청하여

```
    setLoading(false);  
  };  
};
```

6. Register & Login

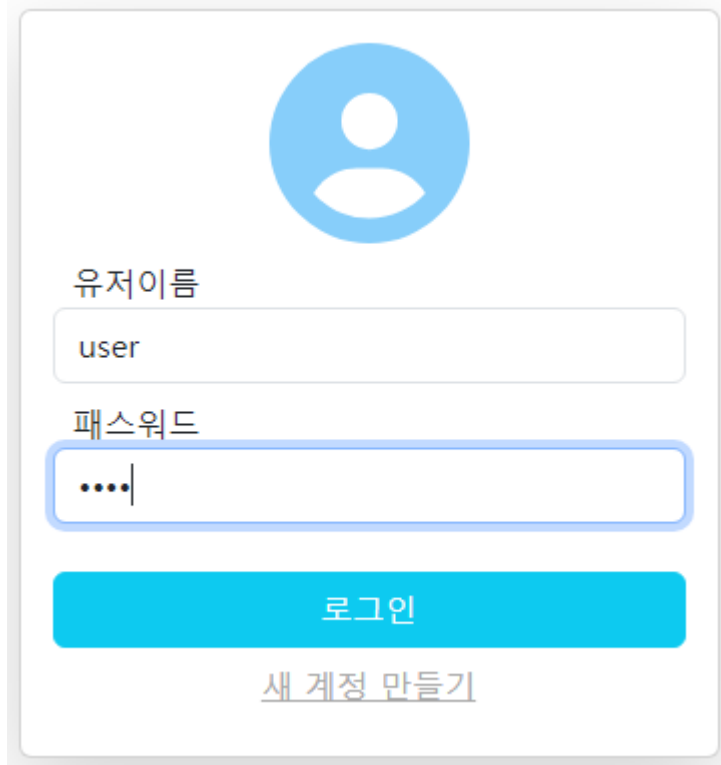
▶ Login-5

```
return (  
  <div className='container mt-5'>  
    <div className='card ms-auto me-auto p-3 shadow-lg custom-card'>  
      <FontAwesomeIcon icon={faUserCircle} className='ms-auto me-auto user-icon' />  
  
      {errorMessage && <div className='alert alert-danger'>{errorMessage}</div>}  
  
      <form onSubmit={handleLogin} noValidate className={submitted ? 'was-validated' : ''}>  
        <div className='form-group my-2'>  
          <label htmlFor='username'>유저이름</label>  
          <input type='text' name='username' className='form-control'  
            placeholder='username' value={user.username} onChange={handleChange} required />  
          <div className='invalid-feedback'>유저네임을 입력해주세요</div>  
        </div>  
        ...  
        <button className='btn btn-info text-white w-100 mt-3' disabled={loading}>  
          로그인  
        </button>  
      </form>  
  
      <Link to='/register' className='btn btn-link' style={{ color: 'darkgray' }}>  
        새 계정 만들기  
      </Link>  
    </div>  
  </div>  
)
```

6. Register & Login

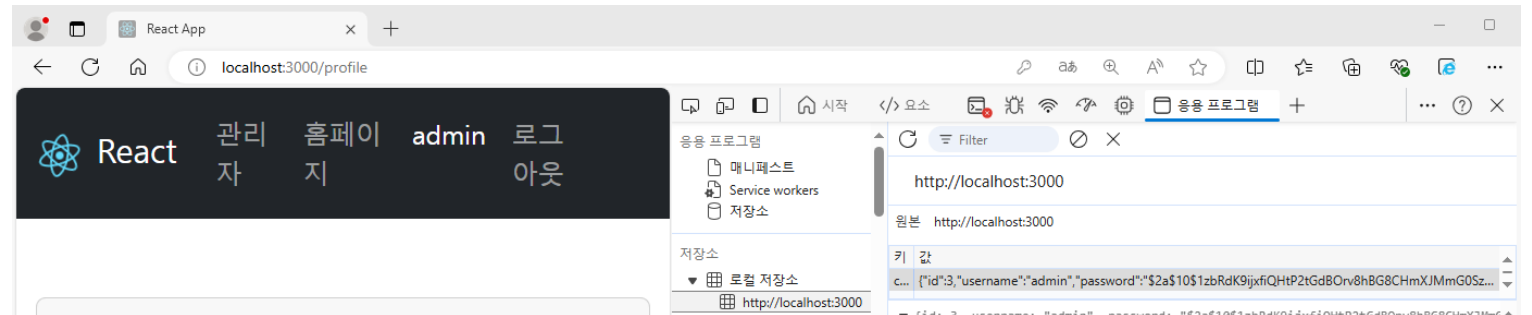
▶ Login-6

▶ 백엔드 서버를 켜고 실제 로그인 시도



A login form with a blue circular user icon at the top. Below it are two input fields: '유저이름' (Username) containing 'user' and '패스워드' (Password) with masked characters. A blue '로그인' (Login) button is below the password field. At the bottom is a link '새 계정 만들기' (Create new account).

- 실제 로그인이 되면 프로파일 페이지로 이동
- Application Local Storage에 서버로부터 받아온 유저의 정보와 토큰이 저장
- 이제 로그인이나 회원가입을 할 수 없다.
- 이미 로그인 정보가 저장되어 스토어에서 먼저 유저 정보를 가져가서 정보가 있으면 프로파일 페이지로 이동됨



6. Register & Login

► Login-7

```
import { useDispatch, useSelector } from 'react-redux';
import logo from '../logo.svg';
import { NavLink, useNavigate } from 'react-router-dom'
import { Role } from '../models/Role';
import { clearCurrentUser } from '../store/actions/user';
const Navbar = () => {
  const currentUser = useSelector((state) => state.user);

  const dispatch = useDispatch();
  const navigate = useNavigate();

  const logout = () => {
    dispatch( clearCurrentUser());
    navigate('/login');
  };
}
```



6. Register & Login

▶ Login-8

▶ Navbar.jsx

- ▶ 관리자 권한이 있는 유저만 관리자 링크 보여주기

```
{currentUser?.role === Role.ADMIN &&  
  <li className='nav-item'>  
    <NavLink to="/admin" href='##' className='nav-link'>  
      관리자  
    </NavLink>  
  </li>  
}
```



6. Register & Login

▶ Login-9

- ▶ 로그인 안된 유저에게만 로그인 가입 링크 보여주기

```
{!currentUser && (  
  <div className='navbar-nav ms-auto me-5'>  
    <li className='nav-item'>  
      <NavLink to="/login" href='##' className='nav-link'>  
        로그인  
      </NavLink>  
    </li>  
  
    <li className='nav-item'>  
      <NavLink to="/register" href='##' className='nav-link'>  
        가입하기  
      </NavLink>  
    </li>  
  </div>  
)}
```



React

홈페이지

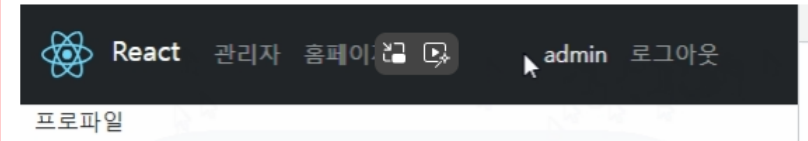
로그인 가입하기

6. Register & Login

▶ Lpgin-11

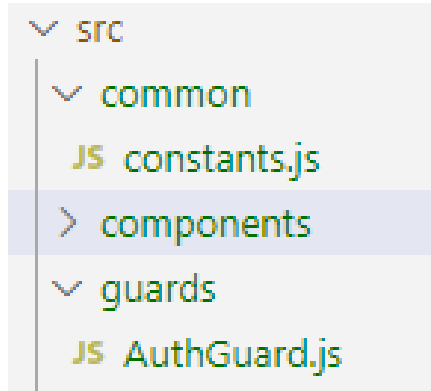
- ▶ 로그인 된 유저에게는 유저 프로파일과 로그아웃 링크 보여주기
- ▶ 로그아웃은 함수이므로 클릭 이벤트에 바인딩

```
{currentUser && (  
  <div className="navbar-nav ms-auto me-5">  
    <li className="nav-item">  
      <NavLink to="/profile" className="nav-link">  
        {currentUser.name}  
      </NavLink>  
    </li>  
    <li className="nav-item">  
      <a href="#" className="nav-link" onClick={logout}>  
        로그아웃  
      </a>  
    </li>  
  </div>  
)}
```



6. Register & Login

▶ AuthGard 인증 및 권한 확인



<Navigate> 요소는 렌더링될 때 현재 위치를 변경 useNavigate를 둘러싼 component wrapper 이며 props와 동일한 모든 인수를 허용. to, replace, state 를 모두 사용

```
import { useSelector } from "react-redux";
import { Navigate } from "react-router-dom";

const AuthGuard = ({children, roles}) => {
  const currentUser = useSelector(state => state.user);

  const authorize=() =>{
    //현재 인증된 유저가 아니면 401
    if(!currentUser){
      return <Navigate to="/401"/>
    }
    //유저의 권한이 props권한과 같은게 없을 경우 401
    if(roles?.indexOf(currentUser.role)===-1){
      return <Navigate to="/401" />
    }
    return children;
  }
  return authorize();
}
export default AuthGuard;
```

6. Register & Login

▶ AuthGard 인증 및 권한 확인

▶ App.js의 내용 수정

const {role, children} =props;

```
<Route path='/profile' element={
  <AuthGuard roles={[Role.ADMIN, Role.USER]}>
    <Profile />
  </AuthGuard>
} />
<Route path='/admin' element={
  <AuthGuard roles={[Role.ADMIN]}>
    <Admin />
  </AuthGuard>
} />
```

Children

Component

- children은 컴포넌트가 감싸고 있는 하위 컴포넌트
- roles 프로프스는 아래 페이지에 접근할 수 있는 권한들을 배열로 전달

백엔드를 켜고 user 로 인증 한 후 <http://localhost:3000/admin> 으로 접근
USER 이기 때문에 ADMIN 권한이 필요한 /admin 페이지는 접근할 수 없음.

6. Register & Login

▶ 테스트

- ▶ 백엔드를 켜고 테스트 user 로 인증 시도



401

권한없음! 이 주소로 접근이 거부되었습니다.

[Back to Home](#)

- ▶ 되었을때 권한은 USER 이기 때문에 ADMIN 권한이
- ▶ 필요한 /admin 페이지는 접근할 수 없다.



6. Register & Login

▶ base.service.js와 constants.js

```
import axios from "axios";
import { clearCurrentUser } from "../store/actions/user";
import store from "../store/configStore"

const authHeader=() => {
  const currentUser=store.getState().user;
  return {
    'Content-Type':'application/json',
    'authorization':'Bearer '+currentUser?.token,
  }
}
```

```
> public
  > src
    > common
      JS constants.js
```

```
1 export const BASE_API_URL='http://localhost:8090';
```

service

JS auth.service.js

JS base.service.js

- 리덕스 스토어에는 로그인시 백엔드에서 인증 후 다시 유저객체를 전달
- 이때 백엔드에서 발급한 JWT 토큰도 user 객체에 포함되어 있음.
- 리덕스 스토어에서 현재 저장된 유저정보를 가져와서 토큰을 헤더에 포함해서 요청하기 위한 authHeader 만들기

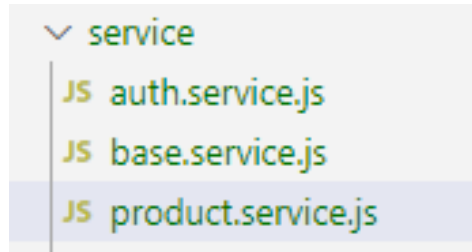
7. Admin Page: 상품 입력, 수정, 삭제

▶ product.service.js



7. Admin Page: 상품 입력, 수정, 삭제

▶ product.service.js



- 제품서비스의 API주소 설정
- 제품의 저장 및 삭제에는 axios 요청에 인증토큰을 포함한 headers를 추가하여 요청
- 제품리스트를 가져올때는 public 요청으로 headers 필요 없음

```
import axios from "axios";
import { BASE_API_URL } from "../common/constants";
import { authHeader } from "../base.service";

const API_URL=BASE_API_URL+"/api/product";

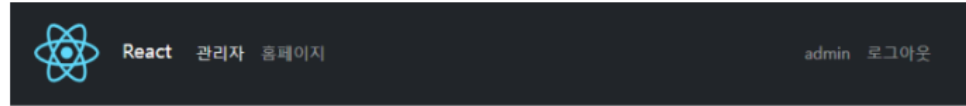
class ProductService {
  saveProduct(product) {
    return axios.post(API_URL, product, { headers: authHeader() });
  }
  deleteProduct(product) {
    return axios.delete(API_URL + '/' + product.id, { headers: authHeader() });
  }
  getAllProducts() {
    return axios.get(API_URL);
  }
}

//객체로 만들어서 사용(export)
const productService = new ProductService();
export default productService;
```

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

```
const Admin={()=>{  
  return (  
    <div className='container'>  
      <div className='card mt-5'>  
        ① header 영역  
        ② body 영역  
      </div>  
    </div>  
  );  
}  
export default Admin;
```



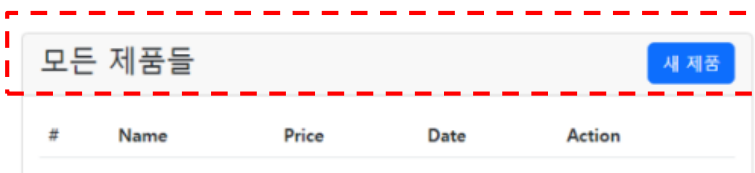
모든 제품들					새 제품
#	Name	Price	Date	Action	



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ header 영역



```
{errorMessage && <div className='alert alert-danger'>{errorMessage}</div>}
<div className='card-header'>
  <div className='row'>
    <div className='col-6'><h3>모든 제품들</h3></div>
    <div className='col-6 text-end'>
      <button className='btn btn-primary' onClick={createProductRequest}>새 제품</button>
    </div>
  </div>
</div>
```



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ body 영역

```
<div className="card-body">
  <table className='table table-striped'>
    <thead>
      <tr>
        <th scope='col'>#</th>
        <th scope='col'>Name</th>
        <th scope='col'>Price</th>
        <th scope='col'>Date</th>
        <th scope='col'>Action</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>Apple</td>
        <td>100000</td>
        <td>2020-10-10</td>
        <td><button>수정</button> <button>삭제</button></td>
      </tr>
    </tbody>
  </table>
</div>
```



모든 제품들				
새 제품				
#	Name	Price	Date	Action

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

- ▶ 제품 등록된 제품 리스트 불러오기

모든 제품들					새 제품
#	Name	Price	Date	Action	
1	짜장면	7000 원	2023. 7. 2. 오전 10:09:15	수정	삭제
2	짬뽕	8000 원	2023. 7. 2. 오전 10:10:10	수정	삭제

```
const Admin=()=>{
  const [productList, setProductList] = useState([]);
  const [selectedProduct, setSelectedProduct] = useState(new Product('', '', 0));
  const saveComponent = useRef();
  const [errorMessage, setErrorMessage] = useState('');
  const deleteComponent = useRef();

  useEffect(() =>{
    productService.getAllProducts().then((response) => {
      setProductList(response.data);
    });
  }, []);
```

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ 제품 리스트 화면 출력

```
<tbody>
{productList.map((item, ind) => (
  <tr key={item.id}>
    <th scope='row'>{ind + 1}</th>
    <td>{item.name}</td>
    <td>`${item.price} 원`</td>
    <td>{new Date(item.createTime).toLocaleString()}</td>
    <td>
      <button className='btn btn-primary me-1'>수 정</button>
      <button className='btn btn-danger'>삭 제</button>
    </td>
  </tr>
)}
</tbody>
```

모든 제품들					새 제품
#	Name	Price	Date	Action	
1	짜장면	7000 원	2023. 7. 2. 오전 10:09:15	수 정	삭 제
2	짬뽕	8000 원	2023. 7. 2. 오전 10:10:10	수 정	삭 제

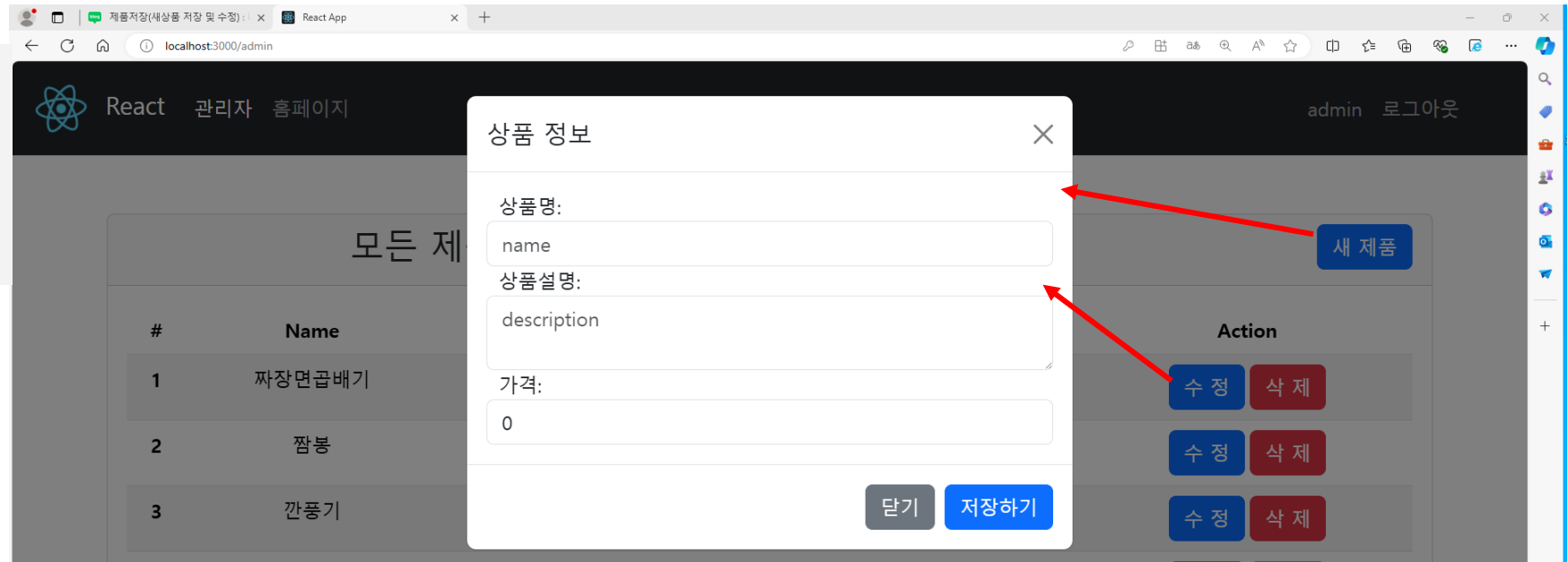
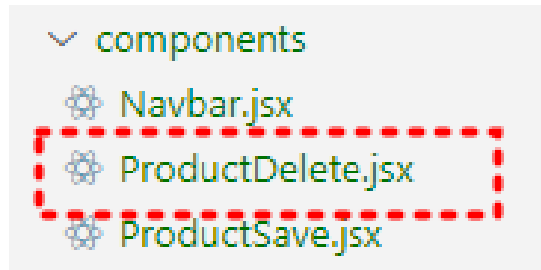
- toLocaleString()
- 날짜를 한국 표준으로 설정

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ 제품 저장 및 수정

- ▶ 부트스트랩으로 모달창(팝업창)을 만들어 버튼 클릭 시 제품의 정보를 수정 및 새로 작성하여 저장하는 컴포넌트 작성.



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ 제품 리스트 화면 출력

```
<tbody>
{productList.map((item, ind) => (
  <tr key={item.id}>
    <th scope='row'>{ind + 1}</th>
    <td>{item.name}</td>
    <td>`${item.price} 원`</td>
    <td>{new Date(item.createTime).toLocaleString()}</td>
    <td>
      <button className='btn btn-primary me-1'>수 정</button>
      <button className='btn btn-danger'>삭 제</button>
    </td>
  </tr>
)}
</tbody>
```

모든 제품들					새 제품
#	Name	Price	Date	Action	
1	짜장면	7000 원	2023. 7. 2. 오전 10:09:15	수 정	삭 제
2	짬뽕	8000 원	2023. 7. 2. 오전 10:10:10	수 정	삭 제

- toLocaleString()
- 날짜를 한국 표준으로 설정

7. Admin Page: 상품 입력, 수정, 삭제

- ▶ Admin 페이지
 - ▶ ProductSave.jsx

```
import { forwardRef, useEffect, useImperativeHandle, useState } from "react";
import Product from "../models/Product";
import { Modal } from 'react-bootstrap';
import productService from "../service/product.service";

const ProductSave=forwardRef((props, ref) =>{
  const [product, setProduct] = useState(new Product('', '', 0));
  const [errorMessage, setErrorMessage] = useState('');
  const [show, setShow] = useState(false);
  const [submitted, setSubmitted] = useState(false);

  useImperativeHandle(ref, () => ({
    //상위컴포넌트에서 변경
    showProductModal() {
      setTimeout(() => setShow(true), 0);
    }
  }));
});
```



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ ProductSave.jsx

▶ 상품 저장 및 수정 모달

```
import { forwardRef, useEffect, useImperativeHandle, useState } from "react";
import Product from "../models/Product";
import { Modal } from 'react-bootstrap';
import productService from "../service/product.service";

const ProductSave=forwardRef((props, ref) =>{
  const [product, setProduct] = useState(new Product('', '', 0));
  const [errorMessage, setErrorMessage] = useState('');
  const [show, setShow] = useState(false);
  const [submitted, setSubmitted] = useState(false);

  useImperativeHandle(ref, () => ({
    //상위컴포넌트에서 변경
    showProductModal() {
      setTimeout(() => setShow(true), 0);
    }
  }));
});
```

7. Admin Page: 상품 입력, 수정, 삭제

- ▶ Admin 페이지
 - ▶ ProductSave.jsx

```
import { forwardRef, useEffect, useImperativeHandle, useState } from "react";
import Product from "../models/Product";
import {Modal} from 'react-bootstrap';
import productService from "../service/product.service";
```

```
const ProductSave=forwardRef((props, ref) =>{
  const [product, setProduct] = useState(new Product('', '', 0));
  const [errorMessage, setErrorMessage] = useState('');
  const [show, setShow] = useState(false);
  const [submitted, setSubmitted] = useState(false);

  useImperativeHandle(ref, () => ({
    //상위컴포넌트에서 변경
    showProductModal() {
      setTimeout(() => setShow(true), 0);
    }
  }));
});
```

모달 show 의 기본값이 false 로 보이지 않기 때문에 보이게 하려면 setShow를 true로 설정.

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ ProductSave.jsx

▶ 모달 UI

```
56     return(  
57       <Modal show={show}>  
58         <form noValidate onSubmit={saveProduct} className={submitted ? 'was-validated' : ''}>  
59           <div className='modal-header'>  
60             <h5 className='modal-title'>상품 정보</h5>  
61             <button type='button' className='btn-close' onClick={() => setShow(false)}></button>  
62           </div>  
63  
64           <div className='modal-body'>  
65             {errorMessage && <div className='alert alert-danger'>{errorMessage}</div>}  
66  
67             <div className='form-group'>  
68               <label htmlFor='name'>상품명: </label>  
69               <input  
70                 type='text'  
71                 name='name'  
72                 placeholder='name'  
73                 className='form-control'  
74                 value={product.name}  
75                 onChange={handleChange}  
76                 required  
77               />  
78               <div className='invalid-feedback'>Name is required.</div>  
79             </div>  
    )
```

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ ProductSave.jsx

▶ 모달 UI

```
<div className='form-group'>
  <label htmlFor='description'>상품설명: </label>
  <textarea
    name='description'
    placeholder='description'
    className='form-control'
    value={product.description}
    onChange={handleChange}
    required
  />
  <div className='invalid-feedback'>Description is required.</div>
</div>
<div className='form-group'>
  <label htmlFor='price'>가격: </label>
  <input
    type='number'
    min='1'
    step='any'
    name='price'
    placeholder='price'
    className='form-control'
    value={product.price}
    onChange={handleChange}
    required
  />
  <div className='invalid-feedback'>Price is required and should be greater than 0.</div>
</div>
```

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ ProductSave.jsx

```
107     </div>
108     <div className='modal-footer'>
109         <button type='button' className='btn btn-secondary' onClick={() => setShow(false)}>
110             닫기
111         </button>
112         <button type='submit' className='btn btn-primary'>
113             저장하기
114         </button>
115     </div>
116 </form>
117 </Modal>
118 )
119 });
120 export default ProductSave;
```



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ Admin.jsx

- ▶ 아래쪽에 모달컴포넌트 ProductSave

```
    </div>
    <ProductSave />
  </div>
);
}
export default Admin;
```

useRef 로 직접 접근하기

```
const Admin=()=>{
  const [productList, setProductList] = useState([]);
  const [selectedProduct, setSelectedProduct] = useState(new Product('', '', 0));
  const saveComponent = useRef();
```

변경

```
<ProductSave ref={saveComponent}/>
```

```
const createProductRequest = () => {
  saveComponent.current?.showProductModal();
};
```

saveComponent.current로 ProductSave 컴포넌트의
useImperativeHandle 안에 있는 showProductModal() 를 직접 호출함

```
<button className='btn btn-primary' onClick={createProductRequest}> 새 제품 </button>
```

7. Admin Page: 상품 입력, 수정, 삭제

- ▶ Admin 페이지
 - ▶ 새 제품 버튼 클릭 시



The screenshot shows a web application interface for an admin page. At the top, there is a dark header with a React logo, the text 'React 관리자', and a user profile 'admin' with a '로그아웃' (Logout) link. The main content area is dimmed, showing a table with columns '#', 'Name', and a list of items: '1 파장', '2 잠봉'. A modal window titled '상품 정보' (Product Information) is open in the center. It contains three input fields: '상품명:' (Product Name) with 'name' as a placeholder, '상품설명:' (Product Description) with 'description' as a placeholder, and '가격:' (Price) with '0' as a placeholder. At the bottom of the modal, there are two buttons: a grey '닫기' (Close) button and a blue '저장하기' (Save) button. A mouse cursor is hovering over the '저장하기' button.

7. Admin Page: 상품 입력, 수정, 삭제

- ▶ Admin 페이지
 - ▶ 새 제품 입력하여 저장 처리
 - ▶ ProductSave.jsx

```
const saveProduct = (e) => {
  e.preventDefault();
  setSubmitted(true);

  if (!product.name || !product.description || !product.price) {
    return;
  }

  productService.saveProduct(product)
    .then((response) => {
      props.onSaved(response.data); //상위컴포넌트에 저장데이터 전달
      setShow(false);
      setSubmitted(false);
    })
    .catch((err) => {
      setErrorMessage('제품 저장시 에러발생!');
      console.log(err);
    });
  setProduct(new Product('', '', 0)); //입력창 초기화
};
```

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

- ▶ 새 제품 입력하여 저장 처리
- ▶ Admin.jsx

```
const saveProductWatcher = (product) => {  
  const newList = productList.concat(product);  
  setProductList(newList);  
};
```

```
<ProductSave ref={saveComponent} onSave={ (p) => saveProductWatcher(p)} />
```

상품 정보

상품명:

우동

상품설명:

해물

가격:

8000

닫기

저장하기

새 제품을 작성하여 저장하면 바로 Admin 페이지에 표시

모든 제품들					새 제품
#	Name	Price	Date	Action	
1	짜장면	7000 원	2023. 7. 2. 오전 10:09:15	수정	삭제
2	짬뽕	8000 원	2023. 7. 2. 오전 10:10:10	수정	삭제
3	우동	8000 원	2023. 7. 29. 오전 4:10:34	수정	삭제

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

- ▶ 수정 버튼 Edit => 수정할 제품 창에 보이기
- ▶ Admin.jsx

선택한 제품 스테이트 혹은 추가

```
const [selectedProduct, setSelectedProduct] = useState(new Product('', '', 0));
```

버튼 클릭시 실행할 함수

```
const editProductRequest = (item) => {  
    console.log(item); setSelectedProduct(item);  
    saveComponent.current?.showProductModal();  
};
```

수정버튼

```
<button className='btn btn-primary me-1' onClick={() => editProductRequest(item)} > 수정 </button>
```



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

- ▶ 수정 버튼 Edit => 수정할 제품 창에 보이기
- ▶ Admin.jsx

ProductSave 모달창으로 선택한제품 스테이트를 전달(Props)

```
<ProductSave  
  ref={saveComponent}  
  product={selectedProduct}  
  onSave={(p) => saveProductWatcher(p)} />
```



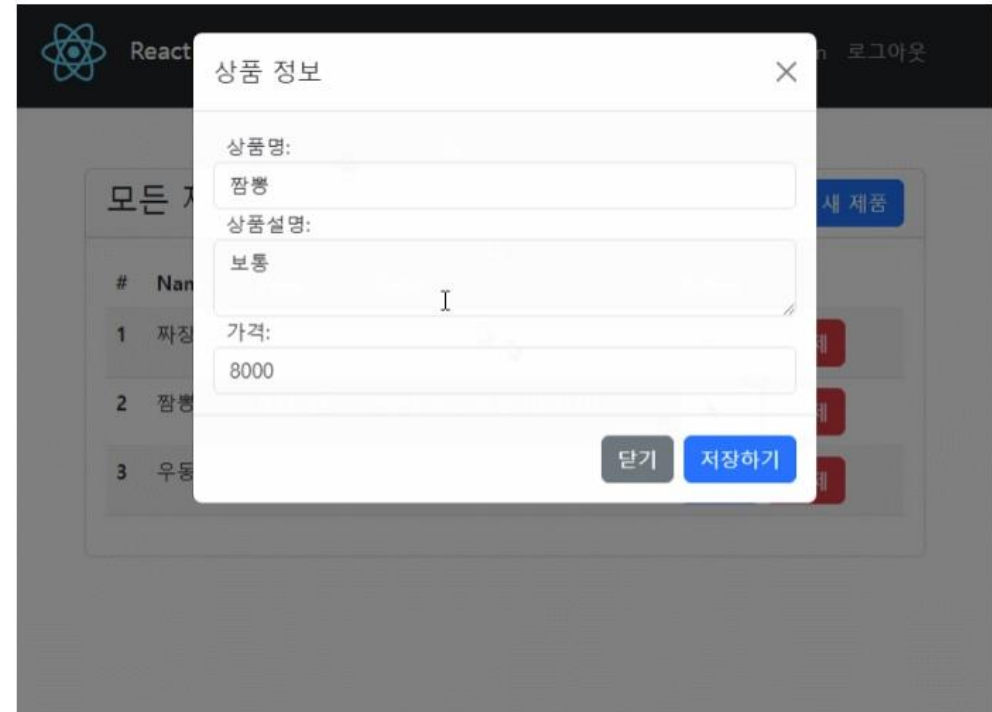
7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

- ▶ 수정 버튼 Edit => 수정할 제품 창에 보이기
- ▶ ProductSave.jsx

```
const ProductSave = forwardRef((props, ref) => {  
  useImperativeHandle(ref, () => ({  
    //상위컴포넌트에서 사용  
    showProductModal() {  
      setTimeout(() => setShow(true), 0);  
    }, }));  
  
  useEffect(() => {  
    setProduct(props.product);  
  }, [props.product]);  
});
```

admin의 selectedProduct 값이 바뀌면 바뀐값으로 product 저장,
edit 버튼 클릭시 그 아이템의 값들이 모달창에 보이게 됨!



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ 수정 처리(Admin.jsx)

```
const saveProductWatcher = (product) => {  
  let itemIndex = productList.findIndex((item) => item.id === product.id);  
  if (itemIndex !== -1) {  
    const newList = productList.map((item) => {  
      if (item.id === product.id) {  
        return product;  
      }  
      return item;  
    });  
    setProductList(?);  
  } else {  
    const newList = productList.concat(product);  
    setProductList(newList);  
  }  
};
```

The screenshot displays the Admin Page interface. A modal window titled '상품 정보' (Product Information) is open, allowing for the editing of a product. The modal contains input fields for '상품명:' (Product Name) with the value '짜짜면', '상품설명:' (Product Description) with the value '보통썬음', and '가격:' (Price) with the value '8500'. At the bottom of the modal are two buttons: '닫기' (Close) and '저장하기' (Save). In the background, a table lists products. The second row, which corresponds to the product being edited, is highlighted with a red dashed border. This row shows '2 짜짜면 8500 원 2023. 7. 29. 오전 8:18:20'. To the right of the table, there are '수정' (Edit) and '삭제' (Delete) buttons for each product entry. A '새 제품' (New Product) button is also visible in the top right corner of the page.

	상품명	가격	등록일	Action
1	짜장면	7000 원	2023. 7. 2. 오전 10:09:15	수정 삭제
2	짜짜면	8500 원	2023. 7. 29. 오전 8:18:20	수정 삭제
3	우동	8000 원	2023. 7. 29. 오전 4:10:34	수정 삭제

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

▶ 제품 삭제(Admin.jsx)

에러 발생시 메시지 state

```
const [errorMessage, setErrorMessage] = useState('');
```

```
const deleteProduct = (item) => {  
  if (!window.confirm('정말로 삭제하겠습니까?')) return;  
  productService.deleteProduct(item)  
    .then((_) => {  
      setProductList(productList.filter((p) => p.id !== item.id));  
    })  
    .catch((err) => {  
      setErrorMessage('삭제중 에러발생!');  
      console.log(err);  
    });  
};
```



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

- ▶ Admin.jsx
- ▶ 에러 발생시 메시지 화면에 표시

```
return (  
  <div className='container'>  
    <div className='card mt-5'>  
      {errorMessage && <div className='alert alert-danger'>{errorMessage}</div>}  
      <div className='card-header'>
```



7. Admin Page: 상품 입력, 수정, 삭제

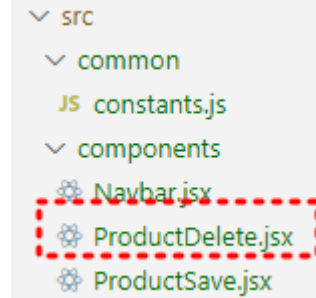
▶ Admin 페이지

▶ 삭제용 모달창 ProductDelete.jsx

```
import { forwardRef, useImperativeHandle, useState } from "react";
import { Modal } from "react-bootstrap";

const ProductDelete=forwardRef((props, ref)=>{
  const [show, setShow] = useState(false);
  useImperativeHandle(ref, () => ({
    showDeleteModal() {
      setShow(true);
    },
  }));

  const deleteProduct = () => {
    props.onConfirmed();
    setShow(false);
  };
});
```



7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지 : 삭제용 모달창 ProductDelete.jsx

```
return (  
  <Modal show={show}>  
    <div className='modal-header'>  
      <h5 className='modal-title'>Confirmation</h5>  
      <button type='button' className='btn btn-close' onClick={() => setShow(false)}></button>  
    </div>  
  
    <div className='modal-body'>정말로 이 제품을 삭제하겠습니까?</div>  
  
    <div className='modal-footer'>  
      <button type='button' className='btn btn-secondary' onClick={() => setShow(false)}>  
        취소  
      </button>  
      <button type='button' className='btn btn-danger' onClick={deleteProduct}>  
        삭제확인  
      </button>  
    </div>  
  </Modal>  
>);  
});  
export default ProductDelete;
```

7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지 : 삭제하기 커스텀 모달 적용하기 (Admin.jsx)

deleteProductRequest 함수 새로 만들기 여기서 삭제할 아이템을 현재 선택한 제품 스테이트 업데이트하기 !

```
const deleteProductRequest = (item) => {  
  console.log(item);  
  setSelectedProduct(item);  
  deleteComponent.current?.showDeleteModal();  
};
```

```
<button className='btn btn-danger' onClick={() => deleteProductRequest(item)}>삭제</button>
```

모달창을 선택할수 있도록 useRef를 사용해서 삭제컴포넌트 선언

```
const deleteComponent = useRef();
```

ProductSave 모달 컴포넌트 아래에 추가

```
<ProductDelete ref={deleteComponent} onConfirmed={() => deleteProduct()} />
```


7. Admin Page: 상품 입력, 수정, 삭제

▶ Admin 페이지

```
const deleteProduct = () => {  
  //if (!window.confirm('정말로 삭제하겠습니까?')) return;  
  productService  
    .deleteProduct(selectedProduct)  
    .then((_) => {  
      setProductList(productList.filter((p) => p.id !== selectedProduct.id));  
    }) .catch((err) => {  
      setErrorMessage('삭제중 에러발생!'); console.log(err);  
    });  
};
```

- Props로 onConfirmed이 삭제모달로 넘어가서 실행 시
- 현재 선택된 제품을 삭제한다. (제품 선택은 삭제 버튼 클릭시 바로 선택됨)
- 모달창(ProductDelete)에서 삭제확인 버튼을 클릭하면
- onConfirmed() 로 전달된 메서드 즉 () => deleteProduct() 이 실행
- 그러므로 Admin의 deleteProduct() 함수에서
- 현재 선택된 제품을 삭제한다. (함수수정필요)

8. 구매(Purchase) 처리

▶ 구매 purchase 서비스,

```
service
├── auth.service.js
├── base.service.js
├── product.service.js
├── purchase.service.js
└── user.service.js
```

```
import axios from "axios";
import { BASE_API_URL } from "../common/constants";
import { authHeader } from "../base.service";

const API_URL = BASE_API_URL + '/api/purchase';

class PurchaseService{
  savePurchaseService(purchase){
    return axios.post(API_URL, purchase, {headers:authHeader()});
  }

  getAllPerchases(){
    return axios.get(API_URL, {headers:authHeader()});
  }
}

const purchaseService = new PurchaseService();
export default purchaseService;
```

8. 구매(Purchase) 처리

▶ Home 페이지

▶ Hook추가

```
const [productList, setProductList] = useState([]);  
const [errorMessage, setErrorMessage] = useState('');  
const [infoMessage, setInfoMessage] = useState('');
```

홈화면에 제품리스트를 가져와서 보여주기 위한 productList
에러발생시 표시하기 위한 errorMessage
정보메세지 표시하기 위한 infoMessage

```
const currentUser = useSelector((state) => state.user);
```

현재 유저 정보를 스토어에서 가져오기

```
useEffect(() => {  
  productService.getAllProducts().then((response) => {  
    setProductList(response.data);  
  });  
}, []);
```

화면 시작할 때 제품리스트 가져오기

```
const purchase = (product) => {  
  if (!currentUser?.id) {  
    setErrorMessage('로그인하셔야 구매가능 합니다.');    return;  
  }  
};
```

구매버튼 클릭시 처리 함수



8. 구매(Purchase) 처리

- ▶ Home 페이지
 - ▶ 화면표시

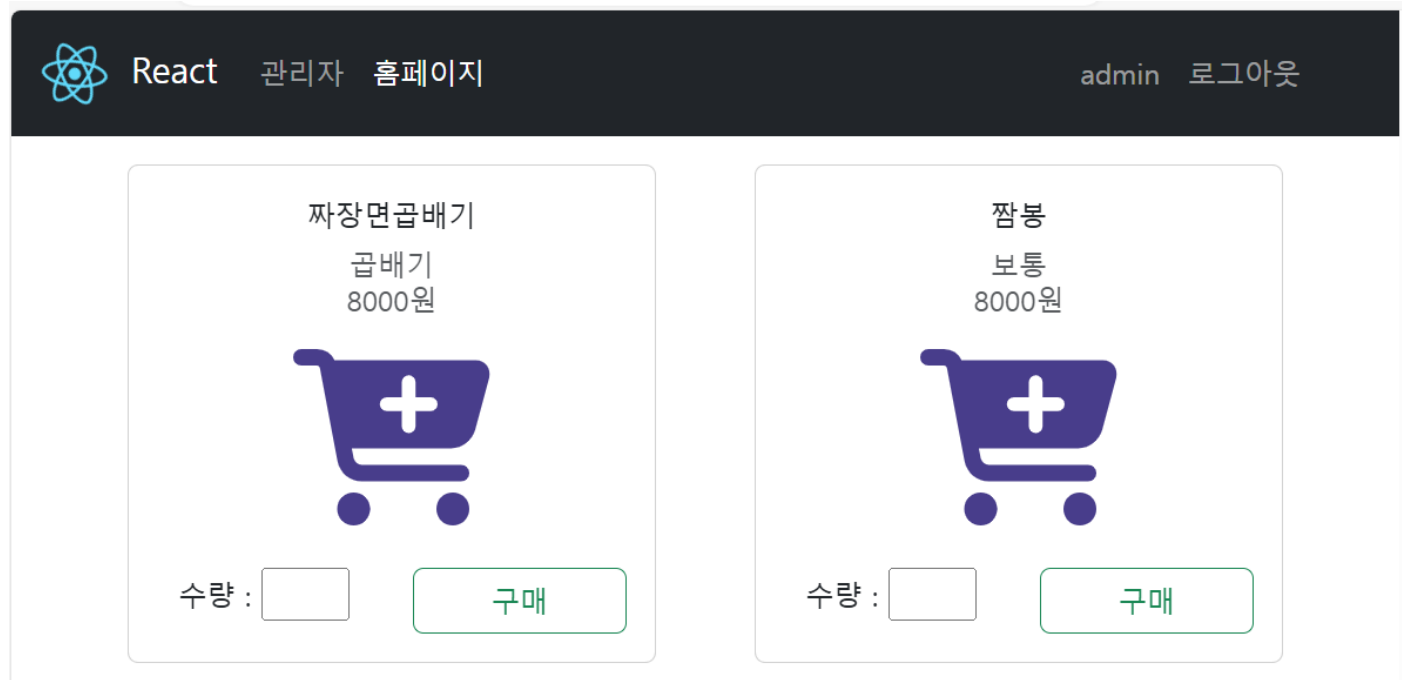
```
return(  
  <div className='mt-3'>  
    {errorMessage && <div className='alert alert-danger'>{errorMessage}</div>  
    {infoMessage && <div className='alert alert-success'>{infoMessage}</div>  
    <div className='d-flex justify-content-around flex-wrap gap-3'>  
      {productList.map((item, ind) => (  
        <div key={item.id} className='card home-card'>  
          <div className='card-body'>  
            <div className='card-title text-uppercase'>{item.name}</div>  
            <div className='card-subtitle text-muted'>{item.description}</div>  
            <div className='card-subtitle text-muted'>`$${item.price}원`</div>  
          </div>  
          <FontAwesomeIcon icon={faCartPlus} className='ms-auto me-auto product-icon' />  
          <div className='row mt-2 p-3'>  
            <div className='col-6'>  
              수량 : <input type="text" name="quantity" onChange={handleChange}/>  
            </div>  
            <div className='col-6'>  
              <button className='btn btn-outline-success w-100' onClick={() => purchase(item)}>  
                구매  
              </button>  
            </div>  
          </div>  
        </div>  
      )  
    )  
  </div>  
)  
}  
export default Home;
```

8. 구매(Purchase) 처리

▶ Home.css


```
.home-card {  
  width: 300px;  
  background-color: #f8f7f7;  
}
```

```
.product-icon {  
  font-size: 100px;  
  color: darkslateblue;  
}  
input{  
  width: 50px;  
}
```



8. 구매(Purchase) 처리


▶ 로그인 안하고 구매 클릭시

 React 홈페이지

로그인 가입하기


로그인하셔야 구매가능 합니다.

짜장면곱배기
곱배기
8000원



수량 :

짬뽕
보통
8000원



수량 :

8. 구매(Purchase) 처리

▶ Home.jsx

```
const purchase = (product) => {  
  if (!currentUser?.id) {  
    setErrorMessage('로그인하셔야 구매가능 합니다.');    return;  
  }  
  
  const purchase = new Purchase(currentUser.id, product.id, quantity);  
  console.log(purchase);  
  purchaseService.savePurchaseService(purchase)  
    .then(() => {  
      setInfoMessage('구매완료!');    })  
    .catch((err) => {  
      setErrorMessage('예상치 못한 에러가 발생했습니다.');      console.log(err);  
    });  
};
```



9. User Service

▶ 유저의 role을 변경하는 service 추가

```
▼ service
JS auth.service.js
JS base.service.js
JS product.service.js
JS purchase.service.js
JS user.service.js
```

```
import { BASE_API_URL } from '../common/constants';
import axios from 'axios';
import { authHeader } from './base.service';

const API_URL = BASE_API_URL + '/api/user';

class UserService {
  changeRole(role){
    return axios.put(API_URL+"/change/"+role, {},{headers:authHeader()})
  }
}

const userService = new UserService();
export default userService;
```



9. User Service

▶ 프로파일 Profile 페이지 (구매내역, 권한변경)

```
const [purchaseList, setPurchaseList] = useState([]);  
const [errorMessage, setErrorMessage] = useState('');
```

구매한 상품 내역리스트
에러메세지

```
const currentUser = useSelector((state) => state.user);  
const dispatch = useDispatch();
```

리덕스 스토어의 현재 유저가져오기
액션디스패치 만들기 (유저 세션 클리어)
네비게이트 객체 만들기 (페이지 이동)

페이지 시작할때 구매내역 리스트 가져오기

```
useEffect(() => {  
  purchaseService.getAllPurchaseItems().then((response) => {  
    setPurchaseList(response.data);  
  });  
}, []);
```



9. User Service

- ▶ 프로필 Profile 페이지 (구매내역, 권한변경)
 - ▶ 권한 수정 메서드

```
const changeRole = () => {  
  const newRole = currentUser.role === Role.ADMIN ? Role.USER : Role.ADMIN;  
  
  userService.changeRole(newRole)  
    .then(() => {  
      //clear session  
      dispatch(clearCurrentUser()); //유저를 클리어  
      window.location.href = '/login';  
    })  
    .catch((err) => {  
      setErrorMessage('예기치 않은 에러가 발생했습니다.');
```

console.log(err);

```
    });  
};
```



9. User Service

- ▶ 프로파일 Profile 페이지 (구매내역, 권한변경)
 - ▶ 화면에 구매상품들 테이블 표시와 현재유저 권한 표시

```
36     return(  
37         <div className="mt-5">  
38             {errorMessage} && <div className="alert alert-danger">{errorMessage}</div>  
39  
40             <div className="card">  
41                 <div className="card-header">  
42                     <div className="row">  
43                         <div className="col-6">  
44                             <h3>구매한 상품들</h3>  
45                         </div>  
46                         <div className="col-6 text-end">  
47                             현재 유저의 권한은 <strong>{currentUser?.role}</strong> 입니다.  
48                             <button onClick={changeRole} className="btn btn-primary ms-3">  
49                                 권한 변경  
50                             </button>  
51                         </div>  
52                     </div>  
53                 </div>  
            </div>  
        )  
    )
```

9. User Service

- ▶ 프로파일 Profile 페이지
(구매내역, 권한변경)
 - ▶ 화면에 구매상품들 테이블
표시와 현재유저 권한 표시

```
54 <div className="card-body">
55   <table className="table table-striped">
56     <thead>
57       <tr>
58         <th scope="col">#</th>
59         <th scope="col">상품명</th>
60         <th scope="col">수량</th>
61         <th scope="col">구매일자</th>
62       </tr>
63     </thead>
64     <tbody>
65       {purchaseList.map((item, ind) => (
66         <tr key={ind}>
67           <th scope="row">{ind + 1}</th>
68           <td>{item.name}</td>
69           <td>`${item.quantity} 개`</td>
70           <td>{new Date(item.purchaseTime).toLocaleDateString()}</td>
71         </tr>
72       ))}
73     </tbody>
74   </table>
75 </div>
76 </div>
77 </div>
78 )
```

10. 새로고침 후 로그인 유지

- ▶ 새로고침 수행 시

- ▶ state의 값이 초기값으로 변경, 로그인 상태 유지되지 않음

- ▶ **redux-persist 라이브러리 사용**

- ▶ 새로고침 후 redux state 값 유지를 위해 사용
 - ▶ 설치 : `npm install redux-persist`



10. 새 로그인 후 로그인 유지

▶ configStore.js 수정

```
import {combineReducers, legacy_createStore as createStore} from 'redux';  
import {persistStore, persistReducer} from 'redux-persist';  
//Redux Persist에서 사용할 스토리지 엔진(로컬 스토리지 사용)  
import storage from 'redux-persist/lib/storage';  
import userReducer from './reducers/user';
```

```
const persistConfig={  
  key:'root', // 로컬 스토리지에 저장되는 키  
  storage,  
}
```

```
const allReducers=combineReducers({  
  user:userReducer  
})
```

//persistConfig에 따라 상태를 로컬 스토리지에 저장하고 관리.

```
const persistedReducer=persistReducer(persistConfig, allReducers);  
const store=createStore(persistedReducer); // persistedReducer 이용해서 지속성 유지한다  
const persistor=persistStore(store); //지속성 객체를 만든다.
```

```
export {store, persistor};
```

10. 새로고침 후 로그인 유지

▶ index.js 수정

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Provider } from 'react-redux';
import { store, persistor } from './store/configStore';
import { PersistGate } from 'redux-persist/integration/react';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <App />
      </PersistGate>
    </Provider>
  </React.StrictMode>
);
```

PersistGate를 사용하여 지속성을 관리.

11. JWT 토큰 유효기간 만료시 로그인 로직 추가

▶ JWT 토큰 방식의 보안

- ▶ 시크릿 키값을 쉽게 알수 없도록 해야하고(비교적 안전)
- ▶ JWT 토큰은 탈취 당할 수 있기 때문에 토큰의 유효기간을 짧게 잡는 것이 좋다.
- ▶ 현재(840000)는 1일로 설정
- ▶ 테스트로 5초로 바꾸어 테스트
- ▶ 실제 서비스에는 30분 설정(권장).

```
app.jwt.expiration-in-ms=5000
```



11. JWT 토큰 유효기간 만료시 로그인 로직 추가

▶ 토큰 유효 기간 설정

- ▶ 백엔드의 application.properties 에서 유효기간을 5초로 변경

`app.jwt.expiration-in-ms=5000`

- ▶ 토큰의 유효기간이 지나면 JWT 토큰으로 요청시 에러가 발생.
- ▶ 유효기간 만료로 에러가 났을대 다시 로그인하게 만드는 보안 서비스를 추가
- ▶ Axios의 인터셉터 기능을 이용하여 모든 Axios 응답시 다시 로그인하는 로직을 추가
- ▶ 백엔드 서버를 다시 실행하고 로그인 후 5초 뒤 서비스를 요청하면 에러가 나는 것을 확인!



11. JWT 토큰 유효기간 만료시 로그인 로직 추가

▶ base.servise.js

- ▶ 엑시오스의 인터셉터를 이용해 에러가 났을 경우 **HTTP 상태 401(Unauthorized) 또는 403(Forbidden) 일때 => 다시 로그인 요청하기**

```
...  
  
const handleResponseWithLoginCheck = () => {  
  axios.interceptors.response.use(  
    (response) => response,  
    (error) => {  
      const currentUser = store.getState().user;  
      const isLoggedIn = currentUser?.token;  
      const status = error?.response?.status;  
      if (isLoggedIn && [401, 403].includes(status)) {  
        store.dispatch(clearCurrentUser());  
        window.location.href = '/login';  
      }  
      return Promise.reject(error);  
    }  
  );  
};  
  
export {authHeader, handleResponseWithLoginCheck};
```



11. JWT 토큰 유효기간 만료시 로그인 로직 추가

▶ index.js 수정

- ▶ root 상단에 다음의 내용을 추가하여 적용

```
handleResponseWithLoginCheck();  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
  
...
```

