

## 6. 리액트와 상품 API 서버 연동

상품 API의 경우 파일 데이터가 추가된다는 것을 제외하면 JSON을 이용하는 데이터 처리 방식과 유사하다. 하지만 파일 데이터가 추가되기 때문에 처리 과정에 걸리는 시간에 맞는 모달창을 보여주는 등의 추가적인 부분이 필요하다.

### 6.1 상품 관련 React-Router 설정

개발하려는 상품 기능은 목록화면에서 새로운 상품을 등록할 수 있고, 조회화면에서는 수정/삭제 화면으로 이동이 가능하다. 상품등록 시에는 상품의 이미지들을 함께 추가해서 등록한다.

상품 목록 페이지는 상품들의 이미지를 같이 보여주고, 페이지 처리가 가능하다.

특정 상품을 선택하면 상품의 조회 페이지가 출력되고 해당 상품의 모든 이미지가 출력된다.

해당 상품의 'Modify'를 선택하면 상품 수정이 가능하다. 이 때 상품 정보와 같이 새로운 이미지를 추가하거나 삭제할 수 있다.

상품의 수정/삭제 후에는 모달창에서 수정/삭제 결과를 확인하고 수정시에는 조회 페이지로, 삭제시에는 목록 페이지로 이동한다.

이전에 개발한 리액트의 라우터를 사용해서 페이지나 내용에 해당하는 컴포넌트를 추가하는 방식으로 개발한다.

#### src/router/Root.jsx

```
import React, { Suspense, lazy } from 'react';
import { createBrowserRouter } from 'react-router-dom';
import Loading from '../pages/Loading';
const MainPage = lazy(() => import('../pages/MainPage'));
const About = lazy(() => import('../pages/AboutPage'));
const ListPage = lazy(() => import('../pages/todo/ListPage'));
const ReadPage = lazy(() => import('../pages/todo/ReadPage'));
const AddPage = lazy(() => import('../pages/todo/AddPage'));
const ModifyPage = lazy(() => import('../pages/todo/ModifyPage'));
const ProductListPage = lazy(() =>
import('../pages/product/ListPage'));
const ProductAddPage = lazy(() =>
import('../pages/product/AddPage'));
const ProductReadPage = lazy(() =>
import('../pages/product/ReadPage'));
const ProductModifyPage = lazy(() => import('../pages/product/ModifyPage'));

const root = createBrowserRouter([
  {
    path: '/',
    element: (
      <Suspense fallback=<Loading />>
        <MainPage />
      </Suspense>
    ),
  },
  {
    path: '/about',
```

```

element: (
  <Suspense fallback={<Loading />}>
    <About />
  </Suspense>
),
},
{
  path: '/todo/list',
  element: (
    <Suspense fallback={<Loading />}>
      <ListPage />
    </Suspense>
  ),
},
{
  path: '/todo/read/:tno',
  element: (
    <Suspense fallback={<Loading />}>
      <ReadPage />
    </Suspense>
  ),
},
{
  path: '/todo/add',
  element: (
    <Suspense fallback={<Loading />}>
      <AddPage />
    </Suspense>
  ),
},
{
  path: '/todo/modify/:tno',
  element: (
    <Suspense fallback={<Loading />}>
      <ModifyPage />
    </Suspense>
  ),
},
{
  path: '/product/list',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductListPage />
    </Suspense>
  ),
},
{
  path: '/product/add',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductAddPage />
    </Suspense>
  ),
},
},

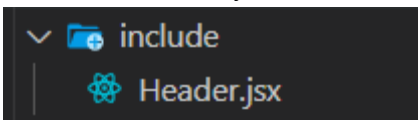
```

```

{
  path: '/product/read/:pno',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductReadPage />
    </Suspense>
  ),
},
{
  path: '/product/modify/:pno',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductModifyPage />
    </Suspense>
  ),
},
]);
export default root;

```

src/include/Header.jsx에서는 브라우저의 상단 메뉴에서 '/products/' 경로로 이동하는 링크를 추가한다.



#### src/include/Header.jsx

```

import React from 'react';
import { Container, Nav, Navbar, NavDropdown } from 'react-bootstrap';
import { useSelector } from 'react-redux';
import { Link } from 'react-router-dom';

export default function Header() {
  const loginState = useSelector((state) => state.loginSlice);

  return (
    <Navbar
      collapseOnSelect
      bg="primary"
      data-bs-theme="dark"
      className="bg-body-primary"
    >
      <Container>
        <Navbar.Toggle aria-controls="responsive-navbar-nav" />
        <Navbar.Collapse id="responsive-navbar-nav">
          <Nav className="me-auto">
            <Nav.Link href="/">MAIN</Nav.Link>
            <Nav.Link href="/about">ABOUT</Nav.Link>
            <NavDropdown
              title="TODO"
              className="bg-body-primary"
              bg="primary"
            >

```

```

    >
    <NavDropdown.Item href="/todo/list">LIST</NavDropdown.Item>
    <NavDropdown.Item href="/todo/add">ADD</NavDropdown.Item>
    <NavDropdown.Divider />

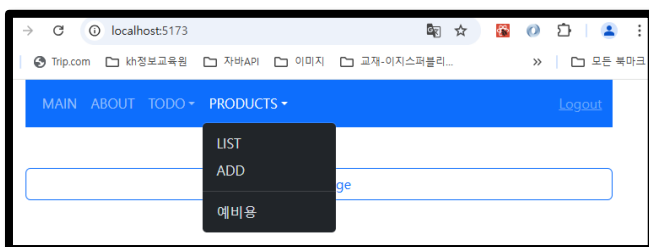
    <NavDropdown.Item href="#action/3.4">예비용</NavDropdown.Item>
  </NavDropdown>
  <NavDropdown
    title="PRODUCTS"
    className="bg-body-primary"
    bg="primary"
  >
    <NavDropdown.Item href="/product/list">LIST</NavDropdown.Item>
    <NavDropdown.Item href="/product/add">ADD</NavDropdown.Item>
    <NavDropdown.Divider />

    <NavDropdown.Item href="#action/3.4">예비용</NavDropdown.Item>
  </NavDropdown>
</>
</Nav>
<Nav>
  <Link to={'/member/login'}>Login</Link>
</Nav>
</Navbar.Collapse>
</Container>
</Navbar>
);
}

```

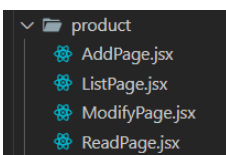
설정된 프로젝트를 실행한다. 화면 상단에 Product 메뉴를 보여지고 list 클릭하면 빈화면으로 연결된다.

<http://localhost:5173/product/list>



## 1)ListPage

상품의 목록을 보여주는src/product/ 폴더에 ListPage.jsx 파일을 생성하고 라우팅 실행을 진행한 다.



### src/product/ListPage.jsx

```

import React from 'react';
import { Container } from 'react-bootstrap';

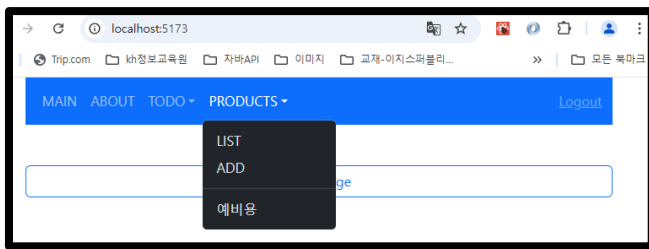
```

```
import Header from '../include/Header';
import ListComponent from '../components/product/ListComponent';

export default function ListPage() {
  return (
    <Container>
      <Header />
      <ListComponent />
    </Container>
  );
}
```

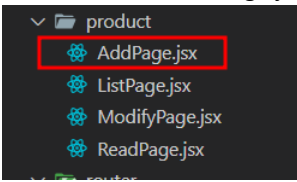
만들어진 ListPage는 router/root.jsx에서는 '/product/list' 경로를 호출할 때 자동으로 ListPage를 보이도록 설정한다.

<http://localhost:5173/product/list>



## 6.2 등록 페이지와 컴포넌트 처리

등록 작업은 AddPage.jsx 이름의 페이지 컴포넌트를 추가하고 라우터를 설정한다.



### src/product/AddPage.jsx

```
import React from 'react';
import { Container } from 'react-bootstrap';
import Header from '../include/Header';
import AddComponent from '../components/product/AddComponent';
export default function AddPage() {
  return (
    <Container>
      <Header />
      <AddComponent />
    </Container>
  );
}
```

### 1)라우팅 설정

router/root.jsx에는 AddPage에 대한 라우팅 설정을 추가한다.

### src/router/root.jsx

```
import React, { Suspense, lazy } from 'react';
import { createBrowserRouter } from 'react-router-dom';
import Loading from '../pages/Loading';
const MainPage = lazy(() => import('../pages/MainPage'));
const About = lazy(() => import('../pages/AboutPage'));
const ListPage = lazy(() => import('../pages/todo/ListPage'));
const ReadPage = lazy(() => import('../pages/todo/ReadPage'));
const AddPage = lazy(() => import('../pages/todo/AddPage'));
const ModifyPage = lazy(() => import('../pages/todo/ModifyPage'));
const ProductListPage = lazy(() =>
import('../pages/product/ListPage'));
const ProductAddPage = lazy(() =>
import('../pages/product/AddPage'));
const ProductReadPage = lazy(() =>
import('../pages/product/ReadPage'));
const ProductModifyPage = lazy(() => import('../pages/product/ModifyPage'));
const root = createBrowserRouter([
  {
    path: '/',
    element: (
      <Suspense fallback={<Loading />}>
        <MainPage />
      </Suspense>
    )
  }
])
```

```

    },
  },
  {
    path: '/about',
    element: (
      <Suspense fallback={<Loading />}>
        <About />
      </Suspense>
    ),
  },
  {
    path: '/todo/list',
    element: (
      <Suspense fallback={<Loading />}>
        <ListPage />
      </Suspense>
    ),
  },
  {
    path: '/todo/read/:tno',
    element: (
      <Suspense fallback={<Loading />}>
        <ReadPage />
      </Suspense>
    ),
  },
  {
    path: '/todo/add',
    element: (
      <Suspense fallback={<Loading />}>
        <AddPage />
      </Suspense>
    ),
  },
  {
    path: '/todo/modify/:tno',
    element: (
      <Suspense fallback={<Loading />}>
        <ModifyPage />
      </Suspense>
    ),
  },
  {
    path: '/product/list',
    element: (
      <Suspense fallback={<Loading />}>
        <ProductListPage />
      </Suspense>
    ),
  },
},

```

```

{
  path: '/product/add',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductAddPage />
    </Suspense>
  ),
},
{
  path: '/product/read/:pno',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductReadPage />
    </Suspense>
  ),
},
{
  path: '/product/modify/:pno',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductModifyPage />
    </Suspense>
  ),
},
]);
export default root;

```

http://localhost:5173/product/add

The screenshot shows a web browser at the URL http://localhost:5173/product/add. The page has a blue header with navigation links: MAIN, ABOUT, TODO, and PRODUCTS. The PRODUCTS dropdown menu is open, showing options: LIST, ADD (highlighted with a red box), and 예비용. Below the header, there is a form with the following fields:
 

- Product Name: A text input field with the placeholder text 'Enter pname'.
- Product Description: A large text area.
- Price: A text input field with the value '0'.
- Files: A file selection interface showing '파일 선택' (Select file) and '선택된 파일 없음' (No files selected).

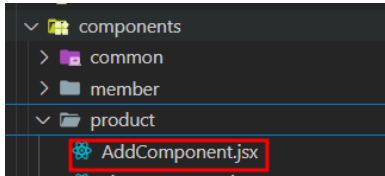
 At the bottom of the form is a blue button labeled '저장' (Save).

## 2)상품의 AddComponent와 API 호출

AddPage 컴포넌트에서 실제 화면의 내용을 구성하는 작업은 components 폴더에 products 폴더를 만들고



AddComponent.jsx는 결과를 확인할 정도로만 작성한다.



#### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';

export default function AddComponent() {

  return (
    <Container className="p-5">
      <h1>상품 추가 컴포넌트</h1>
    </Container>
  );
}
```

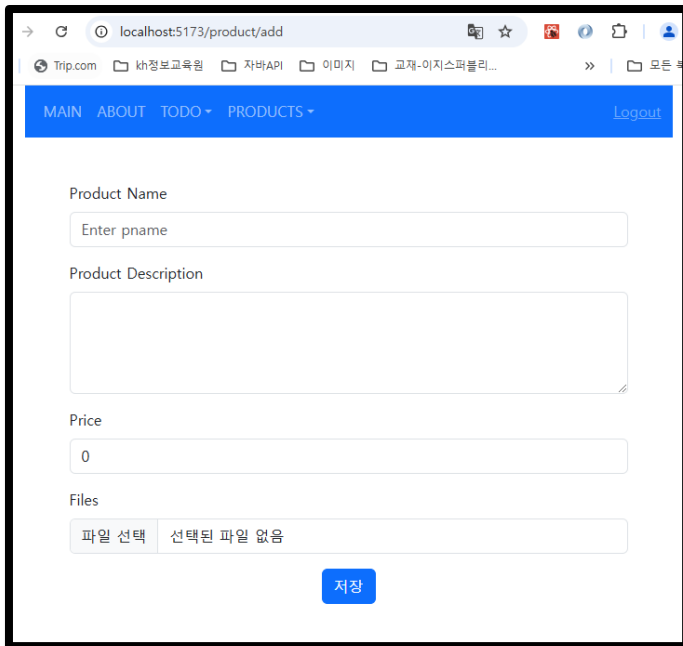
/src/pages/product/AddPage.jsx에 AddComponent를 import한다.

#### src/pages/product/AddPage.jsx

```
import React from 'react';
import { Container } from 'react-bootstrap';
import Header from '../include/Header';
import AddComponent from '../components/product/AddComponent';

export default function AddPage() {
  return (
    <Container>
      <Header />
      <AddComponent />
    </Container>
  );
}
```

http://localhost:5173/product/add



product의 AddComponent의 내용은 Todo의 AddComponent를 참고해서 첨부파일을 추가할 수 있는 구성으로 작성하는데 먼저 화면 구성을 완성한다. 첨부파일은 useRef()를 사용해서 처리한다.

#### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';

const initState = { pname: "", pdesc: "", price: 0, files: [] };

export default function AddComponent() {
  const [product, setProduct] = useState({ ...initState });
  const uploadRef = useRef(); //type="file" 위치
  const handleChangeProduct = (e) => {
    product[e.target.name] = e.target.value;
    setProduct({ ...product });
  };
  const handleClickAdd = (e) => {
    console.log(product)
  }

  return (
    <Container className="p-5">
      <Form>
        <Form.Group className="mb-3">
          <Form.Label>Product Name</Form.Label>
          <Form.Control
            name="pname"

```

```

        type="text"
        value={product.pname}
        onChange={handleChangeProduct}
        placeholder="Enter pname"
      />
    </Form.Group>

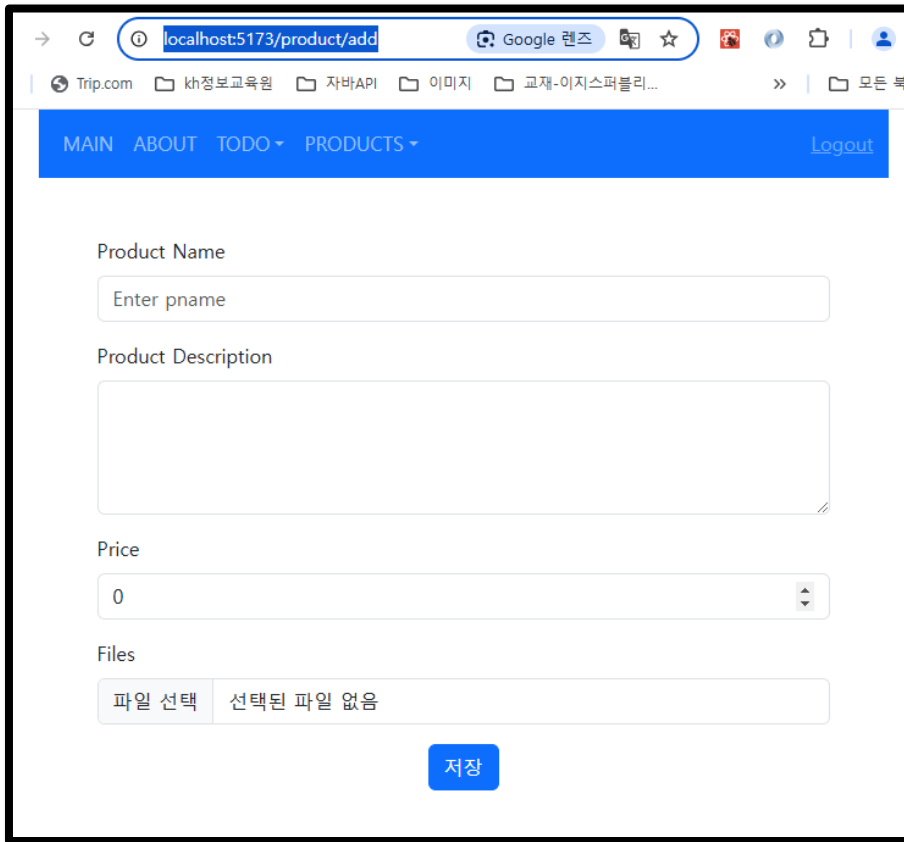
    <Form.Group className="mb-3">
      <Form.Label>Product Description</Form.Label>
      <Form.Control
        name="pdesc"
        value={product.pdesc}
        as="textarea"
        rows={4}
        onChange={handleChangeProduct}
      />
    </Form.Group>

    <Form.Group className="mb-3">
      <Form.Label>Price</Form.Label>
      <Form.Control
        name="price"
        type="number"
        value={product.price}
        onChange={handleChangeProduct}
        placeholder="Enter price"
      />
    </Form.Group>

    <Form.Group className="mb-3">
      <Form.Label>Files</Form.Label>
      <Form.Control ref={uploadRef} type="file" multiple="true" />
    </Form.Group>
  </Form>
  <div className="d-flex justify-content-center gap-2 ">
    <Button variant="primary" type="button" onClick={handleClickAdd}>
      저장
    </Button>
  </div>
</Container>
);
}

```

<http://localhost:5173/product/add>

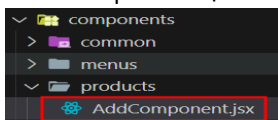


### 3)useRef()와 FormData

useRef()는 기존의 자바스크립트에서 document.getElementById()와 유사한 역할을 한다. 리액트의 컴포넌트는 태그의 id 속성을 활용하면 나중에 동일한 컴포넌트를 여러번 사용해서 화면에 문제가 생기기 때문에 useRef()를 이용해서 처리한다.

‘ADD’를 클릭했을 때 첨부파일에 선택된 정보를 읽어와서 첨부파일의 정보를 파악하고 이를 Ajax 전송에 사용하는 FormData 객체로 구성해야 한다. FormData 타입으로 처리된 내용은 Axios를 이용해서 서버를 호출할 때 사용하게 된다.

AddComponent의 ‘Add’ 버튼의 처리는 다음과 같이 수정한다.



#### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';

const initState = { pname: "", pdesc: "", price: 0, files: [] };

export default function AddComponent() {
  const [product, setProduct] = useState({ ...initState });
  const uploadRef = useRef();
  const handleChangeProduct = (e) => {
```

```

    product[e.target.name] = e.target.value;
    setProduct({ ...product });
  };
  const handleClickAdd = (e) => {
    const files = uploadRef.current.files
    const formData = new FormData()
    for (let i = 0; i < files.length; i++) {
      formData.append("files", files[i]);
    }

    //other data
    formData.append("pname", product.pname)
    formData.append("pdesc", product.pdesc)
    formData.append("price", product.price)
    console.log(formData)
  }
  return (
    생략...
  );
}

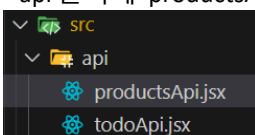
```

useRef를 이용할 때는 current라는 속성을 활용해서 현재 DOM 객체를 참조하게 된다. Ajax를 전송할 때는 FormData 객체를 통해서 모든 내용을 담아서 전송하게 된다.

#### 4)productsAPI의 개발

Axios와의 통신은 api 폴더내에 productApi.js 파일을 작성해서 처리한다. 'multipart/form-data' 방식으로 서버를 호출해야 한다.

api 폴더에 productsApi.js 파일을 생성하고 다음과 같이 작성한다.



##### src/api/productsApi.jsx

```

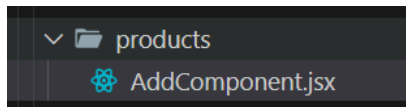
import axios from 'axios';
export const API_SERVER_HOST = 'http://localhost:8080'; //서버 주소
const host = `${API_SERVER_HOST}/api/products`;

export const postAdd = async (product) => {
  //파일업로드 할때에는 기본값인 'Content-Type': 'application/json'을 'multipart/form-data' 변경해야됨
  const header = { headers: { 'Content-Type': 'multipart/form-data' } };
  const res = await axios.post(`${host}/`, product, header);
  return res.data;
};

```

postAdd()에서 주의해야 하는 점은 Axios가 기본적으로 'Content-Type'을 'application/json'을 이 용하기 때문에 파일 업로드를 같이할 때는 'multipart/form-data' 헤더 설정을 추가해 주어야 한다.

AddComponent에서는 버튼을 클릭했을 때 postAdd()를 호출하도록 코드를 수정한다. 전송해야 하는 모든 데이터는 FormData 객체로 처리한다.



#### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';
import { postAdd } from "../../api/productsApi";

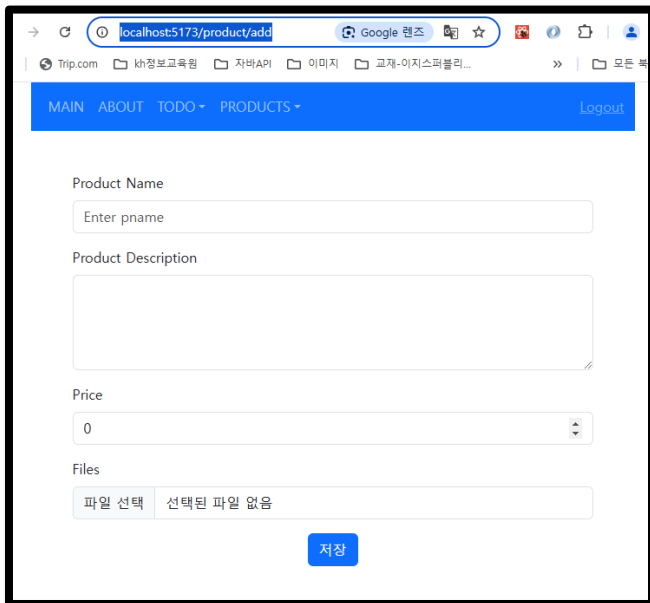
const initState = { pname: '', pdesc: '', price: 0, files: [] };

export default function AddComponent() {
  const [product, setProduct] = useState({ ...initState });
  const uploadRef = useRef();
  const handleChangeProduct = (e) => {
    { product[e.target.name] = e.target.value;
      setProduct({ ...product });
    };
  };
  const handleClickAdd = (e) => {
    const files = uploadRef.current.files
    const formData = new FormData()
    for (let i = 0; i < files.length; i++) {
      formData.append("files", files[i]);
    }

    //other data
    formData.append("pname", product.pname)
    formData.append("pdesc", product.pdesc)
    formData.append("price", product.price)
    console.log(formData)
    postAdd(formData)
  }
  return (
    생략...
  );
}
```

작성된 AddComponent는 '/product/add' 링크에서 확인하고 스프링 부트로 구성된 서버를 실행한 후에 상품을 추가한다.

<http://localhost:5173/product/add>

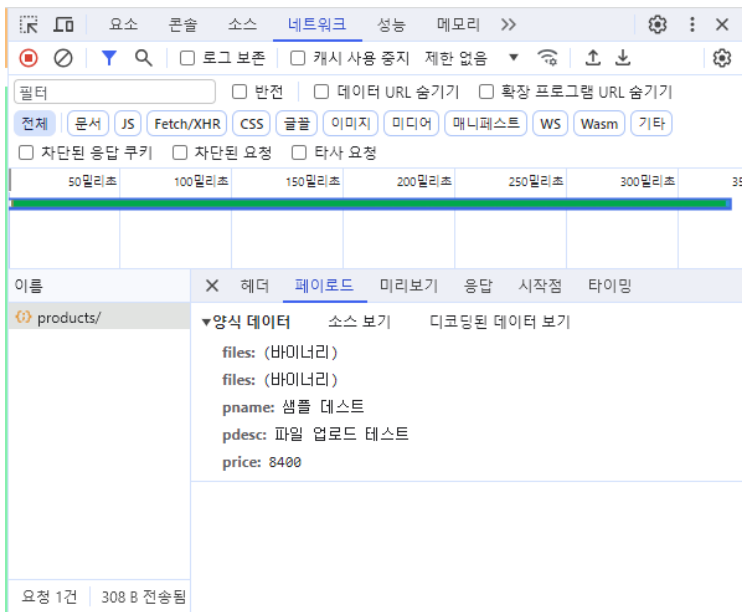


The screenshot shows a web browser at the URL `http://localhost:5173/product/add`. The page has a blue header with navigation links: `MAIN`, `ABOUT`, `TODO`, and `PRODUCTS`, along with a `Logout` link. The main content area contains a form with the following fields:

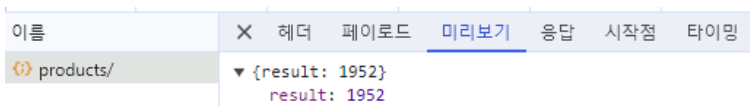
- Product Name:** A text input field with the placeholder text "Enter pname".
- Product Description:** A large text area.
- Price:** A numeric input field with the value "0".
- Files:** A file selection interface showing "파일 선택" (Select file) and "선택된 파일 없음" (No files selected).

A blue button labeled "저장" (Save) is located at the bottom of the form.

ADD 버튼을 클릭한 후에 브라우저의 개발자 도구에서 전달되는 내용(payload)을 확인했을 때 다음과 같이 파일 데이터들은 binary 데이터로 전달되는 것을 확인해야 한다.



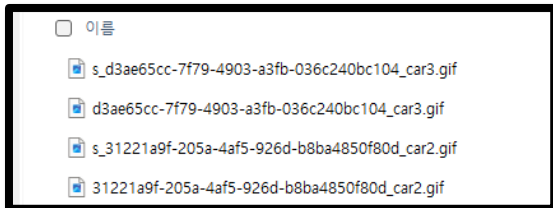
브라우저에서는 서버가 정상적으로 상품을 추가 했는지를 응답(response) 데이터로 확인할 수 있다.



서버에서는 정상적으로 파일이 업로드 되었는지 업로드 폴더를 확인하고 데이터베이스에 해당 번호의 상품 이미지들이 정상적으로 처리되었는지 확인한다.

13	1002	0	테스트 프로덕트 설명	테스트 프로덕트	9000
14	1952	1	이파일 업로드 테스트	샘플 테스트	8400

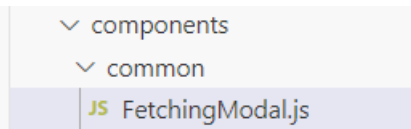
29	1952	31221a9f-205a-4af5-926d-b8ba4850f80d	car2.gif	0
30	1952	d3ae65cc-7f79-4903-a3fb-036c240bc104	car3.gif	1



섬네일 포함해 4개의 파일이 생성되었다. 진행 모달창과 결과 모달창

API 서버와 통신이 필요한 모든 기능은 서버에서 데이터를 가져오는(fetch) 시간을 고려해야 한다. 흔히 '처리중입니다.' 혹은 '로딩중...'과 같은 메시지가 보이는 모달창을 통해 처리하는 경우가 많은데 파일 업로드와 같은 경우 단순한 텍스트를 이용할 때 보다 많은 시간이 걸리기 때문에 진행에 대한 모달창을 보여주고 처리가 끝나면 결과를 모달창으로 보여준다.

#### 5)진행 모달창을 작성하기 위해 componenets의 commons 폴더에 FetchingModal을 추가



##### src/components/common/FetchingModal.jsx

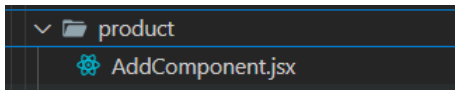
```
import React from 'react';
import Modal from 'react-bootstrap/Modal';

export default function FetchingModal() {
  return (
    <div className="modal show" style={{ display: 'block', position: 'initial' }}>
      <Modal.Dialog>
        <Modal.Header closeButton>
          <Modal.Title>MESSAGE</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          <p>Loading.....</p>
        </Modal.Body>
      </Modal.Dialog>
    </div>
  );
}
```

FetchingModal은 ResultModal과 유사하지만 외부에서 보이거나 사라지도록 제어되기 때문에 속성으로 전달받아야 하는 데이터가 없는 점이 다르다.



AddComponent는 서버와의 통신상태를 fetching이라는 상태를 useState()를 통해서 제어한다. API 서버를 호출할 때는 fetching 상태를 true로 해주고 데이터를 가져온 후에는 false로 변경해서 화면에서 사라지도록 한다.



#### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';
import { postAdd } from "../../api/productsApi";
import FetchingModal from "../../common/FetchingModal";

const initState = { pname: "", pdesc: "", price: 0, files: [] };

export default function AddComponent() {
  const [product, setProduct] = useState({ ...initState });
  const uploadRef = useRef();

  //FetchingModal 보이거나, 사라지게하는 flag역할
  const [fetching, setFetching] = useState(false)
  const handleChangeProduct = (e) =>
  { product[e.target.name] = e.target.value;
    setProduct({ ...product });
  };

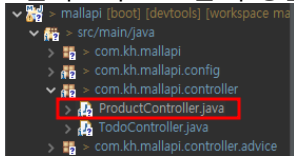
  const handleClickAdd = (e) => {
    const files = uploadRef.current.files
    const formData = new FormData()
    for (let i = 0; i < files.length; i++) {
      formData.append("files", files[i]);
    }

    //other data
    formData.append("pname", product.pname)
    formData.append("pdesc", product.pdesc)
    formData.append("price", product.price)
    console.log(formData)
    setFetching(true)
    postAdd(formData).then(data=> {
      setFetching(false)
    })
  }

  return (
    <Container className="p-5">
      {fetching? <FetchingModal/> :<></>}
      생략...
    </Container>
  );
}
```

ADD 버튼을 클릭하면 FetchingModal이 보였다가 서버 통신이 끝나면 빠르게 사라진다. FetchingModal 시간이걸리도록 API 서버의 TodoController에서는 등록작업 처리시에 Thread.sleep()을 이용해 약간의 시간이

걸리도록 코드를 수정한다. 백엔드의 MALLRESTFUL에서 수정

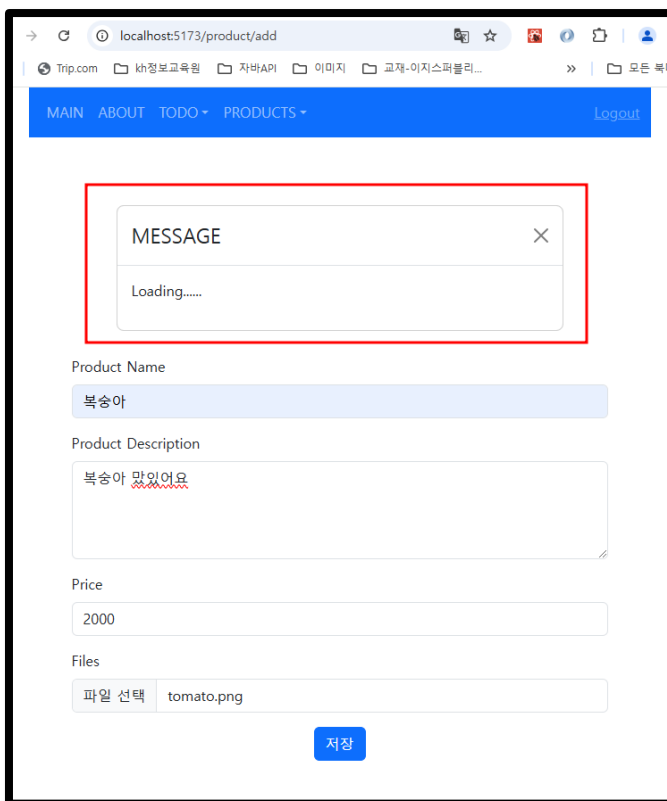


## ProductController.java

생략...

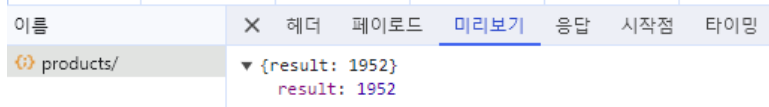
```
@PostMapping("/")
public Map<String, Long> register(ProductDTO productDTO)
{
    log.info("register: " + productDTO);
    List<MultipartFile> files = productDTO.getFiles();
    List<String> uploadFileNames = fileUtil.saveFiles(files);
    productDTO.setUploadFileNames(uploadFileNames);
    log.info(uploadFileNames);
    Long pno = productService.register(productDTO);
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return Map.of("result", pno);
}
```

생략...



## 6)결과 모달창

FetchingModal은 서버와의 통신 과정에서만 보이는데 등록/수정/삭제 처리에서도 보여줄 필요가 있다. 이 경우 ResultModal을 이용해서 처리한다. API 서버에서 상품이 등록되면 전송되는 JSON 데이터는 result라는 속성을 가진다.



AddComponent에서는 이를 활용해서 화면에 결과를 보여주도록 수정한다.

### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';
import { postAdd } from "../../api/productsApi";
import FetchingModal from "../../common/FetchingModal";
import ResultModal from "../../common/ResultModal";

const initState = { pname: "", pdesc: "", price: 0, files: [] };

export default function AddComponent() {
  const [product, setProduct] = useState({ ...initState });
  const uploadRef = useRef();
  const [fetching, setFetching] = useState(false);
  const [result, setResult] = useState(null);

  const handleChangeProduct = (e) => {
    { product[e.target.name] =
      e.target.value;
      setProduct({ ...product });
    };
  };

  const handleClickAdd = (e) => {
    { const files =
      uploadRef.current.files const
      formData = new FormData() for
      (let i = 0; i < files.length; i++)
      { formData.append("files", files[i]);
      }

      //other data
      formData.append("pname", product.pname)
      formData.append("pdesc", product.pdesc)
      formData.append("price", product.price)
      console.log(formData)
      setFetching(true)
      postAdd(formData).then(data=> {
        setFetching(false)
        setResult(data.result)
      })
    }

    const closeModal = () => { //ResultModal 종료
      setResult(null)
    }

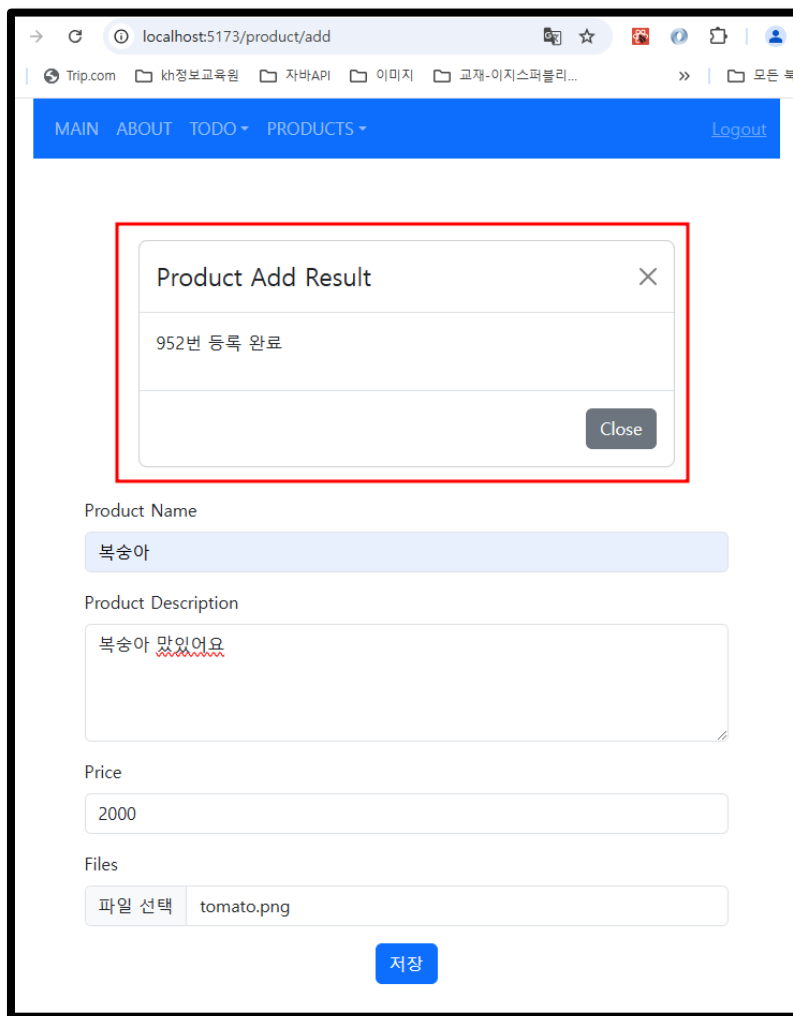
    return (
```

```

<Container className="p-5">
  {fetching? <FetchingModal/> :<></>}
  {result?
    <ResultModal title={"Product Add Result"}
    content={`${result}번 등록 완료`}
    callbackFn={closeModal}
    />
    :<></>
  }
  생략...
};
}

```

FeatchingModal과 ResultModal이 적용되면 'ADD' 버튼을 클릭했을 때 FetchingModal이 빠르게 보여지고 서버와의 통신이 끝나면 ResultModal이 동작한다.



## 7)등록 후 목록 페이지 이동

등록에서 ResultModal이 닫히면 목록 페이지로 이동한다. 라우팅과 관련된 작업은 이미 hooks 폴더에 있는 useCustomMove()을 활용한다.

AddComponent의 경우 useCustomMove의 실행 결과에서 moveToList만 필요하다. ResultModal이 닫히는 closeModal 함수에서 moveToList()를 호출한다.

### src/hooks/useCustomMove.jsx

```
import { useState } from 'react';
import { createSearchParams, useNavigate, useSearchParams, } from 'react-router-dom';

const getNum = (param, defaultValue) => {
  if (!param) {
    return defaultValue;
  }
  return parseInt(param);
};

const useCustomMove = () => {
  const navigate = useNavigate();
  const [refresh, setRefresh] = useState(false);
  const [queryParams] = useSearchParams();
  const page = getNum(queryParams.get('page'), 1);
  const size = getNum(queryParams.get('size'), 10);

  const queryDefault = createSearchParams({ page, size }).toString();

  const moveToList = (pageParam) => {
    let queryStr = "";
    if (pageParam) {
      const pageNum = getNum(pageParam.page, page);
      const sizeNum = getNum(pageParam.size, size);
      queryStr = createSearchParams({
        page: pageNum,
        size: sizeNum,
      }).toString();
    } else {
      queryStr = queryDefault;
    }

    navigate({
      pathname: `../todo/list`,
      search: queryStr,
    });

    setRefresh(!refresh); //추가
  };

  const moveToProductList = (pageParam) => {
    let queryStr = "";
    if (pageParam) {
      const pageNum = getNum(pageParam.page, page);
      const sizeNum = getNum(pageParam.size, size);
      queryStr = createSearchParams({
        page: pageNum,
        size: sizeNum,
      }).toString();
    } else {
      queryStr = queryDefault;
    }
  }
}
```

```

navigate({
  pathname: `../product/list`,
  search: queryStr,
});

setRefresh(!refresh); //추가
};

const moveToModify = (num) => {
  console.log(queryDefault);
  navigate({
    pathname: `../modify/${num}`,
    search: queryDefault, //수정시에 기존의 쿼리 스트링 유지를 위해
  });
};

const moveToProductModify = (num) => {
  console.log(queryDefault);
  navigate({
    pathname: `../product/modify/${num}`,
    search: queryDefault, //수정시에 기존의 쿼리 스트링 유지를 위해
  });
};

const moveToRead = (num) => {
  console.log(queryDefault);
  navigate({
    pathname: `../todo/read/${num}`,
    search: queryDefault,
  });
};

const moveToProductRead = (num) => {
  console.log(queryDefault);
  navigate({
    pathname: `../product/read/${num}`,
    search: queryDefault,
  });
};

return {
  moveToProductList, moveToList, moveToModify, moveToProductModify, moveToProductRead, moveToRead,
  page, size, refresh, }; //moveToModify 추가
};
export default useCustomMove;

```

### src/components/products/AddComponent.jsx

```
import React, { useRef, useState } from 'react';
import { Button, Container, Form } from 'react-bootstrap';
import { postAdd } from "../../api/productsApi";
import FetchingModal from "../../common/FetchingModal";
import ResultModal from "../../common/ResultModal";
import useCustomMove from "../../hooks/useCustomMove";

const initState = { pname: "", pdesc: "", price: 0, files: [] };

export default function AddComponent() {
  const [product, setProduct] = useState({ ...initState });
  const uploadRef = useRef();
  const [fetching, setFetching] = useState(false)
  const [result, setResult] = useState(null)

  const handleChangeProduct = (e) =>
    { product[e.target.name] =
      e.target.value;
      setProduct({ ...product });
    };
  const handleClickAdd = (e) =>
    { const files =
      uploadRef.current.files const
      formData = new FormData() for
      (let i = 0; i < files.length; i++)
      { formData.append("files", files[i]);
      }

      //other data
      formData.append("pname", product.pname)
      formData.append("pdesc", product.pdesc)
      formData.append("price", product.price)
      console.log(formData)
      setFetching(true)
      postAdd(formData).then(data=>
      { setFetching(false)
      setResult(data.result)
      })
    }

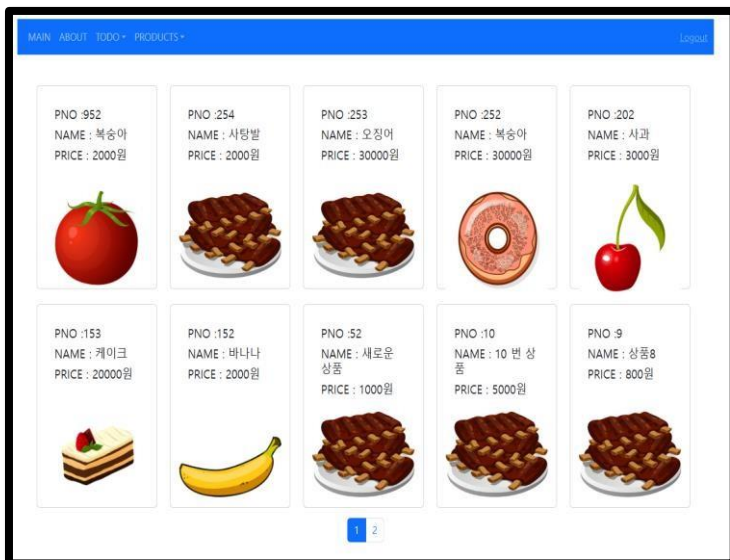
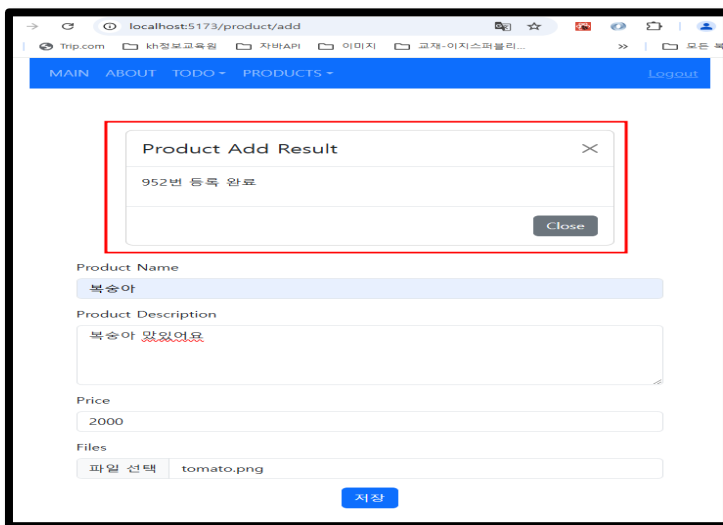
  const closeModal = () => { //ResultModal 종료
    setResult(null)
    useCustomMove({page:1}) //모달 창이 닫히면 이동
  }
  return (
    <Container className="p-5">
      {fetching? <FetchingModal/> :<></>}
      {result?
        <ResultModal title='Product Add Result'
        content={`$${result}번 등록 완료`}
      }
    </Container>
  )
}
```

```

    callbackFn ={closeModal}
  />
  :<></>
}
생략...
);
}

```

등록된 상품의 번호가 보이고 ResultModal 창을 닫은 후에는 자동으로 '/products/list' 경로로 이동 하는 것을 확인한다.





### 6.3 목록 페이지와 목록 컴포넌트

useCustomMove를 이용하면 이동과 관련된 처리를 쉽게 처리할 수 있고 FetchingModal을 이용해서 서버와 통신 과정을 보여주도록 처리한다. api/productsApi.jsx에는 서버에서 목록 데이터를 가져오기 위한 함수를 추가한다.

#### src/api/productsApi.jsx

```
export const API_SERVER_HOST = 'http://localhost:8080';
const host = `${API_SERVER_HOST}/api/products`;

export const postAdd = async (product) => {
  const header = { headers: { 'Content-Type': 'multipart/form-data' } };
  const res = await axios.post(`${host}/`, product, header);
  return res.data;
};

export const getList = async (pageParam) => {
  const { page, size } = pageParam;
  const res = await axios.get(`${host}/list`, {
    params: { page: page, size: size },
  });
  return res.data;
};
```

#### 1)ListComponent 처리

목록 화면 내용을 보여줄 ListComponent를 components/products/ 폴더에 추가한다.

#### src/components/products/ListComponent.jsx

```
import { useEffect, useState } from "react";
import { getList } from "../../api/productsApi";
import { Container } from 'react-bootstrap';
import useCustomMove from "../../hooks/useCustomMove";
import FetchingModal from "../../common/FetchingModal";

const initState = {
  dtoList:[],
  pageNumList:[],
  pageRequestDTO: null,
```

```

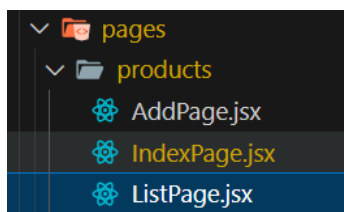
    prev: false,
    next: false,
    totoalCount: 0,
    prevPage: 0,
    nextPage: 0,
    totalPage: 0,
    current: 0
  }
  const ListComponent = () => {
    const { page, size, moveToProductList, moveToProductRead, refresh } = useCustomMove();
    const [serverData, setServerData] = useState(initState)
    //for FetchingModal
    const [fetching, setFetching] = useState(false)
    useEffect(() => {
      setFetching(true)
      getList({page,size}).then(data =>{
        console.log(data)
        setServerData(data)
        setFetching(false)
      })
    }, [page, size, refresh])

    return (
      <Container className="px-5 justify-content-center mb-5">
        <h1>Products List Component</h1>
        {fetching ? <FetchingModal/> : <></>}
      </Container>
    );
  }

  export default ListComponent

```

pages/products 폴더의 ListPage에 ListComponent를 추가한다.



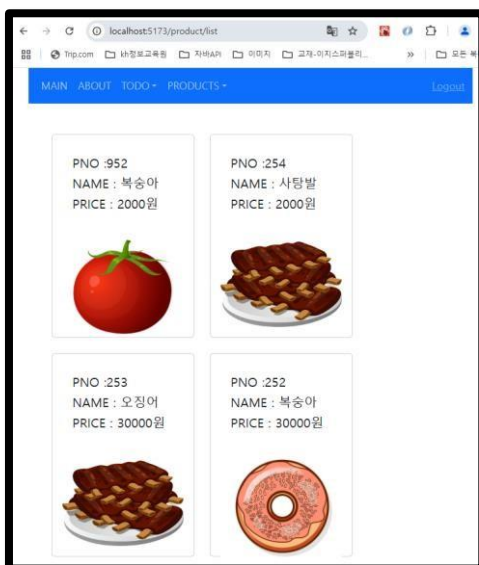
#### src/pages/product/ListPage.jsx

```
import React from 'react';
import { Container } from 'react-bootstrap';
import Header from '../include/Header';
import ListComponent from '../components/product/ListComponent';

export default function ListPage()
{ return (
  <Container>
    <Header />
    <ListComponent />
  </Container>
);
}
```

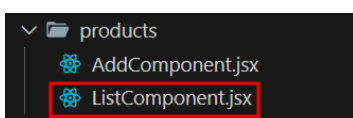
모달창이 빠르게 사라진다.

<http://localhost:5173/product/list>



## 2) 목록 데이터 출력

ListComponent는 API 서버에서 가져온 목록 데이터는 이미지 파일의 이름이 포함되어 있으므로 이를 화면에 출력해 줄 때 서버의 경로를 이용해야 한다. 화면에 데이터를 출력하는 부분을 작성한다. 목록 데이터 출력시에 useCustomMove를 통해 만든 moveToProductRead를 이용해서 상품 조회 페이지로 이동이 가능하도록 구성한다.



### src/components/products/ListComponent.jsx

```
import { useEffect, useState } from 'react';
import { getList } from '../api/productApi';
import useCustomMove from '../hooks/useCustomMove';
import FetchingModal from '../common/FetchingModal';
import { API_SERVER_HOST } from '../api/todoApi';
import { Container, Card, Row } from 'react-bootstrap';

const host = API_SERVER_HOST;

const initState = {
  dtoList: [],
  pageNumList: [],
  pageRequestDTO: null,
  prev: false,
  next: false,
  totoalCount: 0,
  prevPage: 0,
  nextPage: 0,
  totalPage: 0,
  current: 0,
};

const ListComponent = () => {
  const { page, size, moveToProductList, moveToProductRead, refresh } = useCustomMove();
  const [serverData, setServerData] = useState(initState);
  //for FetchingModal
  const [fetching, setFetching] = useState(false);

  useEffect(() => {
    setFetching(true);
    getList({ page, size })
      .then((data) => {
        console.log(data);
        setServerData(data);
        setFetching(false);
      })
      .catch((err) => exceptionHandle(err));
  }, [page, size, refresh]);

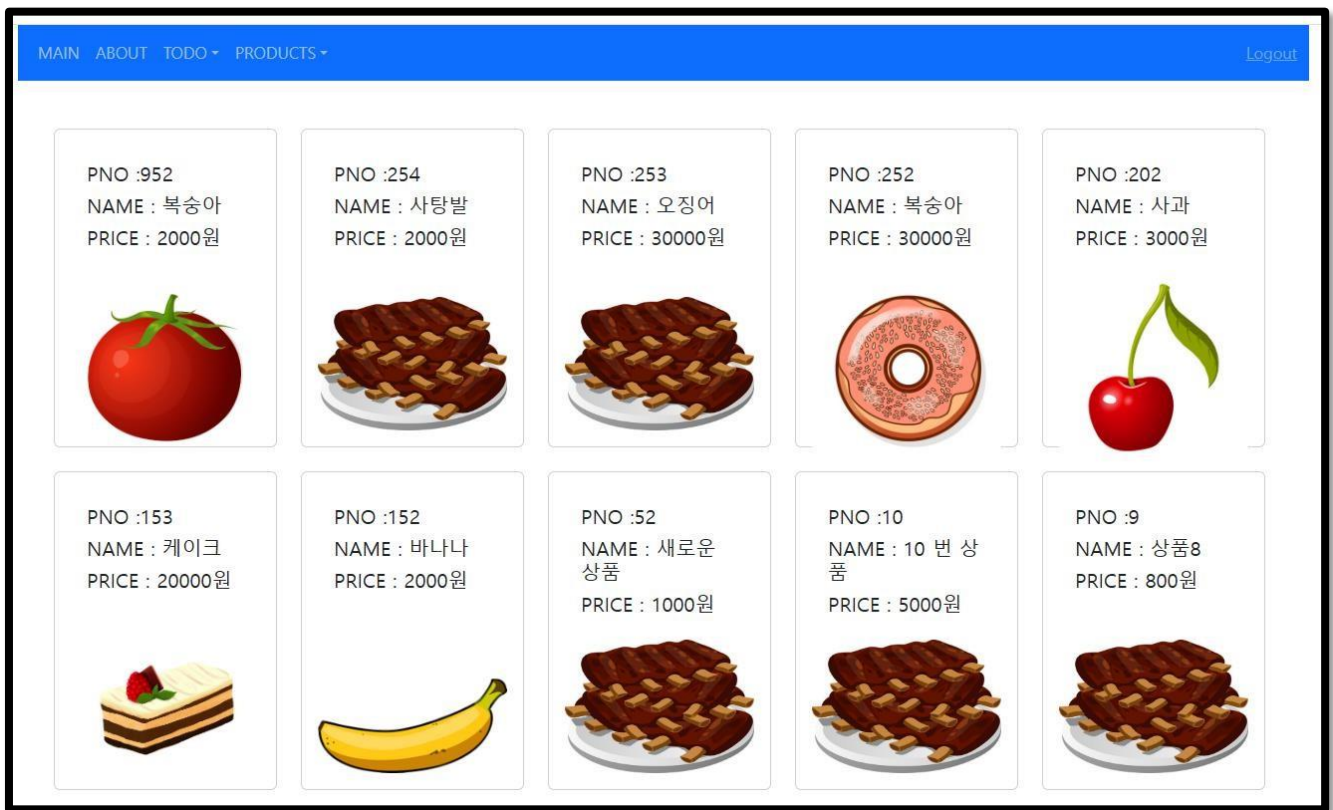
  return (
    <>
    <Container className="px-5 justify-content-center mb-5">
      {fetching ? <FetchingModal /> : <></>}
      <Row className="display-content-around mt-5 gap-4">
        {serverData.dtoList.map((product) => (
          <>
          <Card
            className="p-3"
            style={{ width: '14rem', height: '20rem' }}
            key={product.pno}
            onClick={() => moveToProductRead(product.pno)}
          >
            </>
          </>
        ))}
      </Row>
    </Container>
    </>
  );
}
```

```

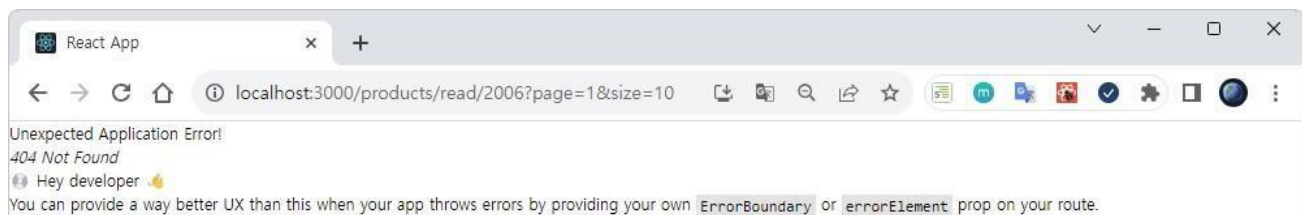
<Card.Body>
  <Card.Title>PNO :{product.pno}</Card.Title>
  <Card.Title>NAME : {product.pname}</Card.Title>
  <Card.Title>PRICE : {product.price}원</Card.Title>
  <Card.Text></Card.Text>
</Card.Body>
<img alt="product" src={` ${host}/api/products/view/s_${product.uploadFileNames[0]} ` />
</Card>
</>
)}}
</Row>
</Container>
</>
);
};
export default ListComponent;

```

http://localhost:5173/product/list?page=1&size=10



상품 조회에 대한 구현이 완성되지 않아 상품을 클릭하면 에러 화면이 보인다.



### 3)페이지 컴포넌트 생성

Src/components/common/PageComponent.jsx

```
import { Container } from 'react-bootstrap';
import Pagination from 'react-bootstrap/Pagination';

const PageComponent = ({ serverData, moveToList }) => {
  return (
    <Container className="d-flex justify-content-center mt-3">
      <Pagination size="md">
        {serverData.prev ? (
          <Pagination.Prev onClick={() => { moveToList({ page: serverData.prevPage }); }} />
        ) : (
          <></>
        )}
        {serverData.pageNumList.map((pageNum) =>
          serverData.current === pageNum ? (
            <Pagination.Item key={pageNum} active onClick={() => { moveToList({ page: pageNum }); }} >
              {pageNum}
            </Pagination.Item>
          ) : (
            <Pagination.Item key={pageNum} onClick={() => { moveToList({ page: pageNum }); }} >
              {pageNum}
            </Pagination.Item>
          )
        )}
        {serverData.next ? (
          <Pagination.Next onClick={() => { moveToList({ page: serverData.nextPage }); }} />
        ) : (
          <></>
        )}
      </Pagination>
    </Container>
  );
};

export default PageComponent;
```

#### 4)페이지 이동

useCustomMove()를 이용해서 moveToList()를 이용할 수 있고, 페이징 처리 역시 common 폴더에 PageComponent를 활용하면 쉽게 구현할 수 있다.

**-src/components/products/ListComponent.jsx**

```
import { useEffect, useState } from 'react';
import { getList } from '../../api/productApi';
import useCustomMove from '../../hooks/useCustomMove';
import FetchingModal from '../../common/FetchingModal';
import { API_SERVER_HOST } from '../../api/todoApi';
import { Container, Card, Row } from 'react-bootstrap';
import PageComponent from '../../common/PageComponent';

const host = API_SERVER_HOST;
const initState = {
  dtoList: [], pageNumList: [], pageRequestDTO: null, prev: false, next: false, totoalCount: 0, prevPage: 0, nextPage: 0,
  totalPage: 0, current: 0;};

const ListComponent = () => {
  const { page, size, moveToProductList, moveToProductRead, refresh } = useCustomMove();
  //serverData 는 나중에 사용
  const [serverData, setServerData] = useState(initState);
  //for FetchingModal
  const [fetching, setFetching] = useState(false);

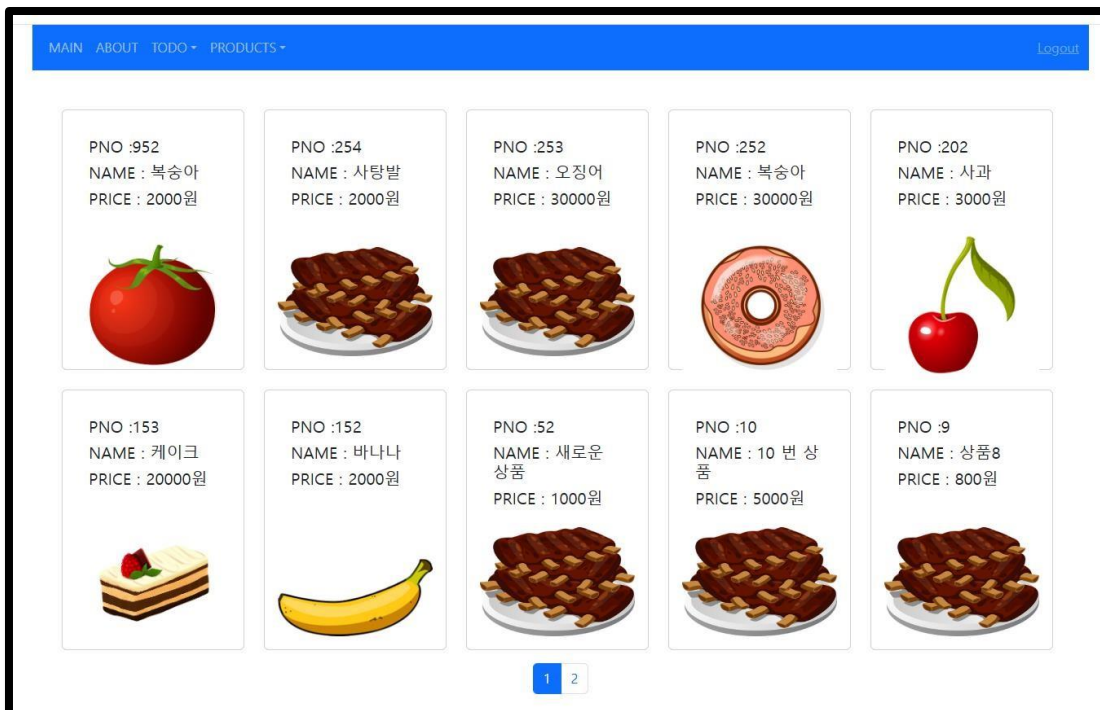
  useEffect(() => {
    setFetching(true);
    getList({ page, size })
      .then((data) => {
        console.log(data);
        setServerData(data);
        setFetching(false);
      }).catch((err) => exceptionHandle(err));
  }, [page, size, refresh]);

  return (
    <>
    <Container className="px-5 justify-content-center mb-5">
      {fetching ? <FetchingModal /> : <></>}
      <Row className="display-content-around mt-5 gap-4">
```

```

{serverData.dtoList.map((product) => (
  <>
  <Card
    className="p-3"
    style={{ width: '14rem', height: '20rem' }}
    key={product.pno}
    onClick={() => moveToProductRead(product.pno)}
  >
    <Card.Body>
      <Card.Title>PNO :{product.pno}</Card.Title>
      <Card.Title>NAME : {product.pname}</Card.Title>
      <Card.Title>PRICE : {product.price}원</Card.Title>
      <Card.Text></Card.Text>
    </Card.Body>
    <img alt="product" src={`/${host}/api/products/view/s_${product.uploadFileNames[0]} ` />
  </Card>
</>
)))
</Row>
<PageComponent serverData={serverData} moveToList={moveToProductList}></PageComponent>
</Container>
</>
);
};
export default ListComponent;

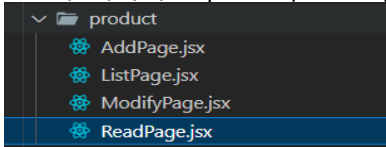
```



#### 6.4 조회 페이지와 조회 컴포넌트



조회 페이지는 `product/ReadPage.jsx`로 작성한다.



#### **src/pages/product/ReadPage.jsx**

```
import React from 'react';
import { useCallback } from 'react';
import { Container } from 'react-bootstrap';
import Header from '../include/Header';
import { createSearchParams, useNavigate, useParams, useSearchParams, } from 'react-router-dom';
import ReadComponent from '../components/product/ReadComponent';

export default function ReadPage() {
  const { pno } = useParams();
  return (
    <Container>
      <Header />
      <ReadComponent pno={pno} />
    </Container>
  );
}
```

router/Root.jsx에 ReadPage에 대한 설정을 추가한다.

#### **Src/router/Root.jsx**

```
import React, { Suspense, lazy } from 'react';
import { createBrowserRouter } from 'react-router-dom';
import Loading from '../pages/Loading';
....
const ProductReadPage = lazy(() => import('../pages/product/ReadPage'));
....
const root = createBrowserRouter([
  {
    path: '/product/list',
    element: (
      <Suspense fallback={<Loading />}>
        <ProductListPage />
      </Suspense>
    ),
  },
  {
    path: '/product/add',
    element: (
      <Suspense fallback={<Loading />}>
```

```

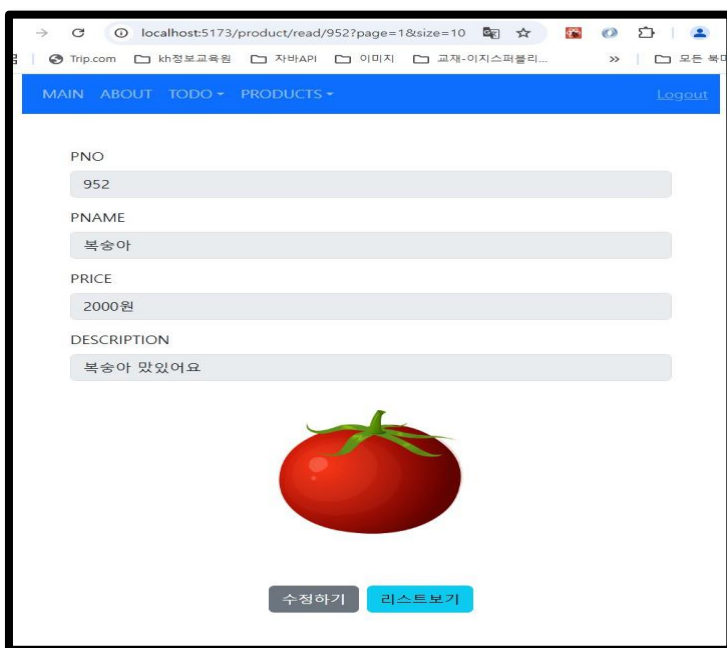
    <ProductAddPage />
  </Suspense>
),
},
{
  path: '/product/read/:pno',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductReadPage />
    </Suspense>
  ),
},
{
  path: '/product/modify/:pno',
  element: (
    <Suspense fallback={<Loading />}>
      <ProductModifyPage />
    </Suspense>
  ),
},
]);
export default root;

```

설정을 완료한 후에는 목록 페이지에서 조회 페이지로 이동이 가능한지 확인한다. 특정 상품의 번호를 클릭하면 '/product/read/상품번호?page=페이지번호&size=10'으로 이동한다.

http://localhost:5173/product/list 에서 이미지 클릭

http://localhost:5173/product/read/952?page=1&size=10



## 1)ReadComponent 처리

ReadComponent의 개발은 api/productsApi.jsx에서 Axios로 특정한 상품 데이터를 조회하는 작업에서 시작한다.

### src/api/productsApi.jsx

```
import axios from "axios"
import { API_SERVER_HOST } from "../todoApi"

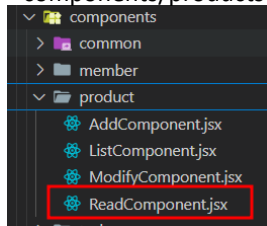
const host = `${API_SERVER_HOST}/api/products`

export const postAdd = async (product) => {
  const header = {headers: {"Content-Type": "multipart/form-data"}}
  //경로 뒤 '/' 주의
  const res = await axios.post(`${host}/`, product, header)
  return res.data
}

export const getList = async ( pageParam ) => { const
  {page,size} = pageParam
  const res = await axios.get(`${host}/list`, {params: {page:page,size:size}}) return res.data
}

export const getOne = async (tno) => {
  const res = await axios.get(`${host}/${tno}` )
  return res.data
}
```

components/products 폴더에는 ReadComponent.jsx를 추가한다.



ReadComponent는 번호(pno)에 해당하는 상품을 서버에서 조회한다. 이를 위해서 속성으로 pno를 전달 받고 useEffect()와 useCustomMove(), FetchingModal을 사용한다.

#### src/components/products/ReadComponent.jsx

```
import { useEffect, useState } from 'react';
import { getOne } from '../api/productApi';
import { API_SERVER_HOST } from '../api/todoApi';
import useCustomMove from '../hooks/useCustomMove';
import FetchingModal from '../common/FetchingModal';
import { Container, Form } from 'react-bootstrap';

const initState = { pno: 0, pname: "", pdesc: "", price: 0, uploadFileNames: [] };
const host = API_SERVER_HOST;

const ReadComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);
  const { moveToProductList, moveToProductModify } = useCustomMove(); //화면 이동용 함수
  const [fetching, setFetching] = useState(false); //fetching

  useEffect(() => {
    setFetching(true);
    getOne(pno).then((data) => {
      setProduct(data);
      setFetching(false);
    });
  }, [pno]);

  return (
    <Container className="p-5">
      {fetching ? <FetchingModal /> : <></>}
      <Form>
        <Form.Group className="mb-3">
          <Form.Label>PNO</Form.Label>
          <Form.Control
            value={pno}
            type="text"
            placeholder="Enter pno"
            disabled
          />
        </Form.Group>

        <Form.Group className="mb-3">
```

```

<Form.Label>PNAME</Form.Label>
<Form.Control
  value={product.pname}
  type="text"
  placeholder="Enter name"
  disabled
/>
</Form.Group>

<Form.Group className="mb-3">
<Form.Label>PRICE</Form.Label>
<Form.Control
  type="text"
  value={product.price + '원'}
  placeholder="Enter price"
  disabled
/>
</Form.Group>
<Form.Group className="mb-3">
<Form.Label>DESCRIPTION</Form.Label>
<Form.Control
  type="text"
  value={product.pdesc}
  placeholder="Enter price"
  disabled
/>
</Form.Group>
<Form.Group className="mb-3 d-flex justify-content-center">
{product.uploadFileNames.map((imgFile, i) => (
  <img
    alt="product"
    key={i}
    style={{ width: '14rem', height: '14rem' }}
    src={` ${host}/api/products/view/s_ ${imgFile}`}
  />
))}
</Form.Group>
</Form>
<div className="d-flex justify-content-center gap-2 mt-5">
<button className="btn btn-secondary" type="button"
  onClick={() => {
    moveToProductModify(pno);
  }}
>
  수정하기
</button>
<button className="btn btn-info" type="button"
  onClick={() => {
    moveToProductList();
  }}
>

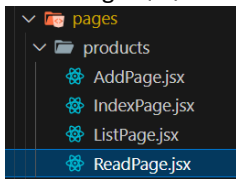
```

```

    리스트보기
  </button>
</div>
</Container>
);
};
export default ReadComponent;

```

ReadPage에서는 ReadComponents를 import하고 추가해서 결과를 확인한다.



#### src/pages/products/ReadPage.jsx

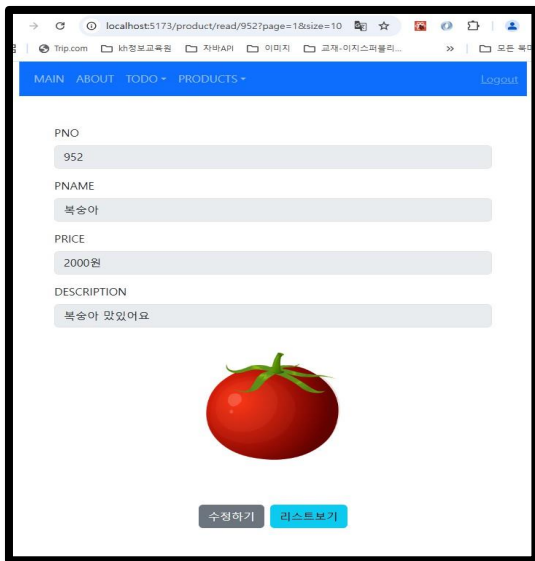
```

import React from 'react';
import { useCallback } from 'react';
import { Container } from 'react-bootstrap';
import Header from '../include/Header';
import { createSearchParams, useNavigate, useParams, useSearchParams, } from 'react-router-dom';
import ReadComponent from '../components/product/ReadComponent';

export default function ReadPage() { const
  { pno } = useParams();
  return (
    <Container>
      <Header />
      <ReadComponent pno={pno} />
    </Container>
  );
}

```

<http://localhost:5173/product/read/952?page=1&size=10>

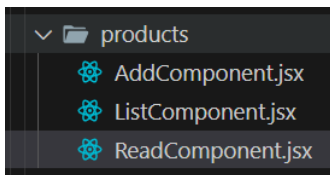


개발자 도구에서 해당 번호의 상품 데이터가 정상적으로 도착했는지 확인한다.



## 2)데이터 출력과 이동

ReadComponent에서는 화면에 데이터를 출력하고, useCustomMove()를 사용해서 다시 목록 화면으로 이동하거나 수정/삭제 화면으로 이동할 수 있도록 코드를 추가한다.



### src/components/products/ReadComponent.jsx

```
import { useEffect, useState } from 'react';
import { getOne } from '../api/productApi';
import { API_SERVER_HOST } from '../api/todoApi';
import useCustomMove from '../hooks/useCustomMove';
```

```

import FetchingModal from '../common/FetchingModal';
import { Container } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';

const initState = {
  pno: 0,
  pname: "",
  pdesc: "",
  price: 0,
  uploadFileNames: [],
};

const host = API_SERVER_HOST;

const ReadComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);
  const { moveToProductList, moveToProductModify } = useCustomMove();
  const [fetching, setFetching] = useState(false);

  useEffect(() => {
    setFetching(true);
    getOne(pno).then((data) => {
      setProduct(data);
      setFetching(false);
    });
  }, [pno]);

  return (
    <Container className="p-5">
      {fetching ? <FetchingModal /> : <></>}
      <Form>
        <Form.Group className="mb-3">
          <Form.Label>PNO</Form.Label>
          <Form.Control
            value={pno}
            type="text"
            placeholder="Enter pno"
            disabled
          />
        </Form.Group>

        <Form.Group className="mb-3">
          <Form.Label>PNAME</Form.Label>
          <Form.Control
            defaultValue={product.pname}
            type="text"
            placeholder="Enter name"
            disabled
          />
        </Form.Group>
      </Container>
    )
  );
};

```



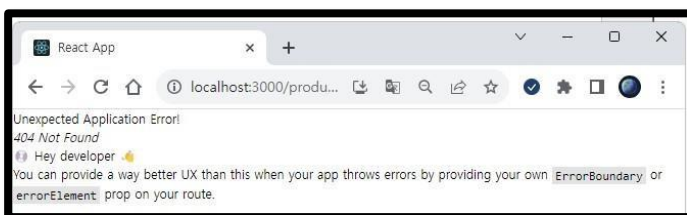
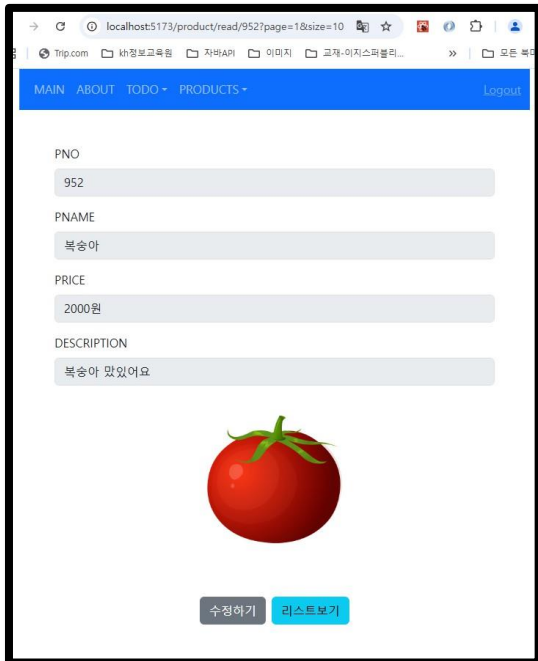
```

<Form.Group className="mb-3">
  <Form.Label>PRICE</Form.Label>
  <Form.Control
    type="text"
    value={product.price + '원'}
    placeholder="Enter price"
    disabled
  />
</Form.Group>
<Form.Group className="mb-3">
  <Form.Label>DESCRIPTION</Form.Label>
  <Form.Control
    type="text"
    defaultValue={product.pdesc}
    placeholder="Enter price"
    disabled
  />
</Form.Group>
<Form.Group className="mb-3 d-flex justify-content-center">
  {product.uploadFileNames.map((imgFile, i) => (
    <img
      alt="product"
      key={i}
      style={{ width: '14rem', height: '14rem' }}
      src={` ${host}/api/products/view/s_${imgFile}`}
    />
  ))}
</Form.Group>
</Form>
<div className="d-flex justify-content-center gap-2 mt-5">
  <button
    className="btn btn-secondary"
    type="button"
    onClick={() =>
      { moveToProductModify(pno);
    }}
  >
    수정하기
  </button>
  <button
    className="btn btn-info"
    type="button"
    onClick={() =>
      { moveToProductList();
    }}
  >
    리스트보기
  </button>
</div>
</Container>
);

```

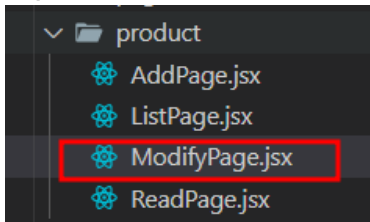
```
};  
export default ReadComponent;
```

조회 화면에서는 상품 정보와 첨부된 모든 이미지가 출력된다. 만일 이미지가 2개 이상인 경우에는 모든 이미지를 출력한다. 조회 화면의 아래쪽에는 'Modify' 버튼과 'List' 버튼이 보이고 'List' 버튼을 클릭하면 원래 페이지로 이동이 가능하다(Modify는 아직 구현되지 구현되지 않아 에러).



## 6.5 수정/삭제 페이지와 컴포넌트

1)수정/삭제를 위한 페이지는 `pages/product/ModifyPage.jsx` 파일에 작성한다.



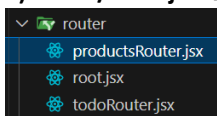
### src/pages/product/ModifyPage.jsx

```
import React from 'react';
import { Container } from 'react-bootstrap';
import Header from '../include/Header';
import { useParams } from 'react-router-dom';
import ModifyComponent from '../components/product/ModifyComponent';

export default function ModifyPage() {
  const { pno } = useParams();

  return (
    <Container>
      <Header />
      <ModifyComponent pno={pno} />
    </Container>
  );
}
```

2)router/Root.jsx에서 ModifyPage을 추가해서 라우팅을 설정한다.



### src/router/Root.jsx

```
import React, { Suspense, lazy } from 'react';
import { createBrowserRouter } from 'react-router-dom';
import Loading from '../pages/Loading';
....
const ProductModifyPage = lazy(() => import('../pages/product/ModifyPage'));
....
const root = createBrowserRouter([
  {
    path: '/product/list',
    element: (
      <Suspense fallback={<Loading />}>
        <ProductListPage />
      </Suspense>
    )
  }
]);
```

```

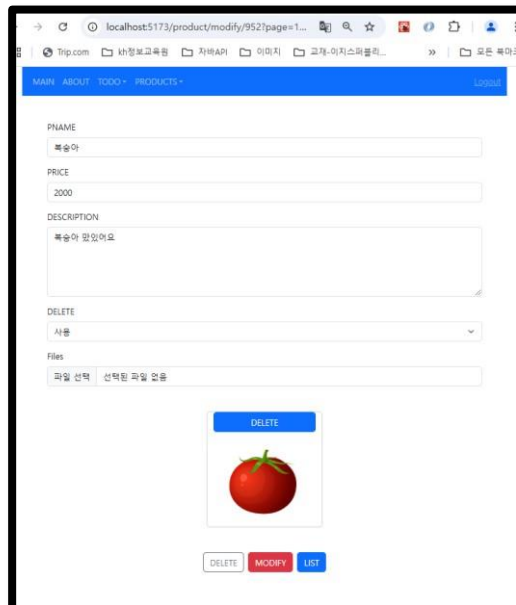
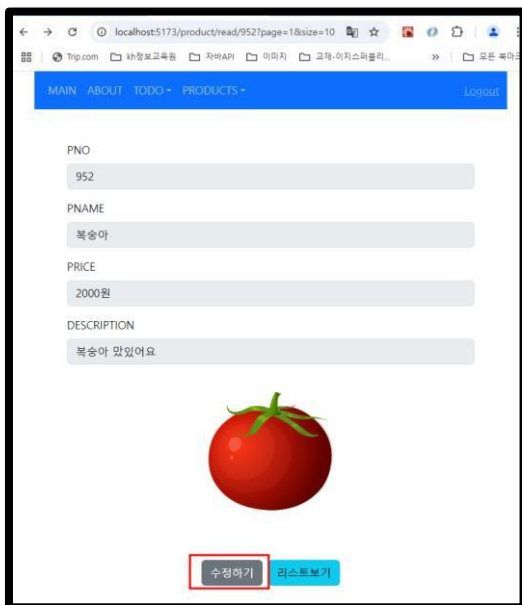
    },
  },
  {
    path: '/product/read/:pno',
    element: (
      <Suspense fallback={<Loading />}>
        <ProductReadPage />
      </Suspense>
    ),
  },
  {
    path: '/product/modify/:pno',
    element: (
      <Suspense fallback={<Loading />}>
        <ProductModifyPage />
      </Suspense>
    ),
  },
];
export default root;

```

상품 조회 화면에서 Modify 버튼을 클릭하면 수정/삭제 화면으로 이동한다.

<http://localhost:5173/products/read/2006?page=1&size=10> 에서 Modify 클릭

<http://localhost:5173/products/modify/2006?page=1&size=10>



### 3) ModifyComponent 처리

수정 작업은 등록과 같이 첨부 파일이 존재하기 때문에 'multipart/form-data' 헤더를 설정해서 전송 처리하고 실제 작업은 해당 상품 번호만 전달해서 처리한다.

api/productsApi.jsx에 수정/삭제 함수를 추가한다.

#### src/api/productsApi.jsx

```
import axios from "axios"
import { API_SERVER_HOST } from "../todoApi"

const host = `${API_SERVER_HOST}/api/products`

export const postAdd = async (product) => {
  const header = {headers: {"Content-Type": "multipart/form-data"}}
  // 경로 뒤 '/' 주의
  const res = await axios.post(`${host}/`, product, header)
  return res.data
}

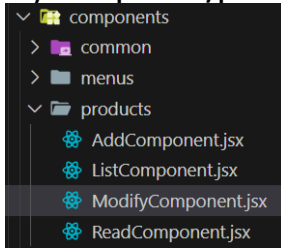
export const getList = async (pageParam) => { const
  {page,size} = pageParam
  const res = await axios.get(`${host}/list`, {params: {page:page,size:size}})
  return res.data
}

export const getOne = async (tno) => {
  const res = await axios.get(`${host}/${tno}`) return res.data
}

export const putOne = async (pno, product) => {
  const header = {headers: {"Content-Type": "multipart/form-data"}}
  const res = await axios.put(`${host}/${pno}`, product, header)
  return res.data
}

export const deleteOne = async (pno) => {
  const res = await axios.delete(`${host}/${pno}`)
  return res.data
}
```

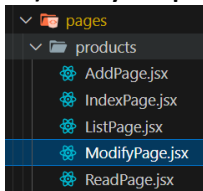
4) components/products 폴더에 ModifyComponent.jsx를 추가한다.



src/components/products/ModifyComponent.jsx

```
const ModifyComponent = ({pno}) => { return (  
  <Container className="p-5">  
    Product Modify Component  
  </Container>  
);  
}  
  
export default ModifyComponent;
```

5) ModifyComponent는 ModifyPage에서 import 한다. 이때 현재 상품 번호를 전달한다.



Src/pages/products/ModifyPage.jsx

```
import React from 'react';  
import { Container } from 'react-bootstrap';  
import Header from '../include/Header';  
import { useParams } from 'react-router-dom';  
import ModifyComponent from '../components/product/ModifyComponent';  
  
export default function ModifyPage() {  
  const { pno } = useParams();  
  
  return (  
    <Container>  
      <Header />  
      <ModifyComponent pno={pno}/>  
    </Container>  
  );  
}
```

목록 -> 조회 -> 수정 단계로 확인한다. <http://localhost:5173/products/modify/2006?page=1&size=10>



MAIN ABOUT TODO+ PRODUCTS+ Logout

PNAME  
복숭아

PRICE  
2000

DESCRIPTION  
복숭아 맛있어요

DELETE  
사용

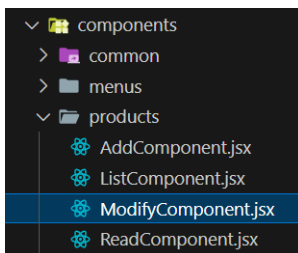
Files  
파일 선택 선택된 파일 없음

DELETE

DELETE MODIFY LIST

## 6)데이터 출력

ModifyComponent는 전달되는 pno를 속성으로 처리하고 서버와 통신 처리후 결과를 출력한다. FetchingModal을 이용해서 서버와의 통신 상태를 알 수 있게 처리하고 이미지들을 출력한다.



### src/components/products/ModifyComponent.jsx

```
import { useEffect, useState } from "react";
import { getOne } from "../../api/productsApi";
import FetchingModal from "../../common/FetchingModal";

const initState = {
  pno: 0,
  pname: "",
  pdesc: "",
  price: 0,
  delFlag: false,
  uploadFileNames: [],
};
```

```

const ModifyComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);
  const [fetching, setFetching] = useState(false);

  useEffect(() => {
    setFetching(true);
    getOne(pno).then((data) => {
      setProduct(data);
      setFetching(false);
    });
  }, [pno]);

  return (
    <Container className="p-5">
      Product Modify Component
      {fetching ? <FetchingModal /> : <></>}
    </Container>
  );
};

export default ModifyComponent;

```

브라우저에서 FetchingModal창을 통해서 서버와 통신이 이루어 졌음을 확인한다(너무 빠르게 사라짐). 서버에서 가져온 상품 정보를 화면에 출력하고 변경이 가능한 속성들을 <input>으로 작성하고 onChange()를 이용해서 변경 가능하도록 한다.

#### src/components/products/ModifyComponent.jsx

```

import { useEffect, useState, useRef } from 'react';
import { Container, Card, Row, Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';
import { deleteOne, getOne, putOne } from '../api/productApi';
import FetchingModal from '../common/FetchingModal';
import { API_SERVER_HOST } from '../api/todoApi';

const initState =
{
  pno: 0,
  pname: '',
  pdesc: '',
  price: 0,
  delFlag: false,
  uploadFileNames: [],
};

const host = API_SERVER_HOST;

const ModifyComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);

```



```
const [fetching, setFetching] = useState(false);
const uploadRef = useRef(); //파일위치저장
```

```
useEffect(() => {
  setFetching(true);
  getOne(pno).then((data) => {
    setProduct(data);
    setFetching(false);
  });
}, [pno]);
```

```
const handleChangeProduct = (e) => {
  product[e.target.name] = e.target.value;
  setProduct({ ...product });
};
```

```
const deleteOldImages = (imageName) => {
};
```

```
return (
  <Container className="p-5">
    {fetching ? <FetchingModal /> : <></>}
```

```
    <Form>
      <Form.Group className="mb-3">
        <Form.Label>PNAME</Form.Label>
        <Form.Control
          name="pname"
          value={product.pname}
          type="text"
          placeholder="Enter name"
          onChange={handleChangeProduct}
        />
      </Form.Group>
```

```
      <Form.Group className="mb-3">
        <Form.Label>PRICE</Form.Label>
        <Form.Control
          name="price"
          type="number"
          value={product.price}
          placeholder="Enter price"
          onChange={handleChangeProduct}
        />
      </Form.Group>
```

```
      <Form.Group className="mb-3">
        <Form.Label>DESCRIPTION</Form.Label>
```

```

<Form.Control
  name="pdesc"
  defaultValue={product.pdesc}
  as="textarea"
  rows={5}
  onChange={handleChangeProduct}
/>
</Form.Group>
<Form.Group className="mb-3">
  <Form.Label>DELETE</Form.Label>
  <Form.Select
    name="delFlag"
    value={product.delFlag ? '사용' : '삭제'}
    onChange={handleChangeProduct}
  >
    <option value={false}>사용</option>
    <option value={true}>삭제</option>
  </Form.Select>
</Form.Group>
<Form.Group className="mb-3">
  <Form.Label>Files</Form.Label>
  <Form.Control ref={uploadRef} type="file" multiple="true" />
</Form.Group>
</Form>
<Row className="d-flex justify-content-center mt-5 gap-4">
  {product.uploadFileNames.map((imgFile, i) => (
    <>
      <Card style={{ width: '14rem', height: '14rem' }} key={i}>
        <Button
          variant="primary"
        >
          DELETE
        </Button>
        <Card.Body>
          <img
            alt="img"
            style={{ width: '10rem' }}
            src={`/${host}/api/products/view/s_${imgFile}`}
          />
        </Card.Body>
      </Card>
    </>
  ))}
</Row>

<div className="d-flex justify-content-center gap-2 mt-5">
  <button
    className="btn btn-outline-secondary"

```

```

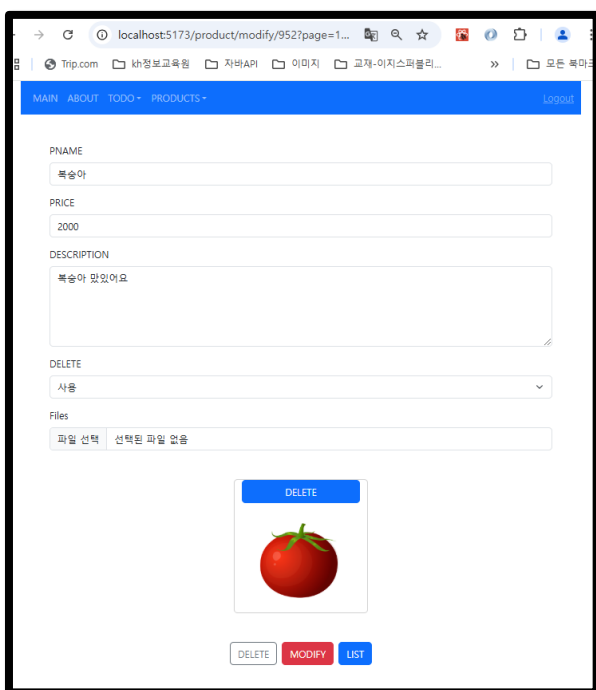
    type="button"
    onClick={handleClickDelete}
  >
    DELETE
  </button>
  <button
    className="btn btn-danger"
    type="button"
    onClick={handleClickModify}
  >
    MODIFY
  </button>
  <button
    className="btn btn-primary"
    type="text"
    onClick={moveToProductList}
  >
    LIST
  </button>
</div>
</Container>
);
};

export default ModifyComponent;

```

해당 상품의 이미지들이 출력되고 입력창은 수정이 가능한 상태가 된다.

http://localhost:5173/product/modify/952?page=1&size=10



## 7) 기존 이미지 삭제

ModifyComponent에서 기존 이미지 삭제는 이미지 상단에 있는 'DELETE' 버튼을 클릭했을 때 상품 데이터가 가지고 있는 uploadFileNames 배열에서 해당 이미지를 삭제한다. 배열에서는 filter()를 사용해 서 해당 이미지가 아닌 이미지들만 유지하도록 한다.

src/components/products/ModifyComponent.jsx

생략...

```
const deleteOldImages = (imageName) => {  
  const resultFileNames = product.uploadFileNames.filter( fileName =>  
    fileName !== imageName)  
  product.uploadFileNames = resultFileNames  
  setProduct({...product})  
}
```

생략...

상품 이미지에 있는 버튼을 클릭해서 해당 이미지를 배열에서 제거할 수 있도록 이벤트 처리를 추가한다.

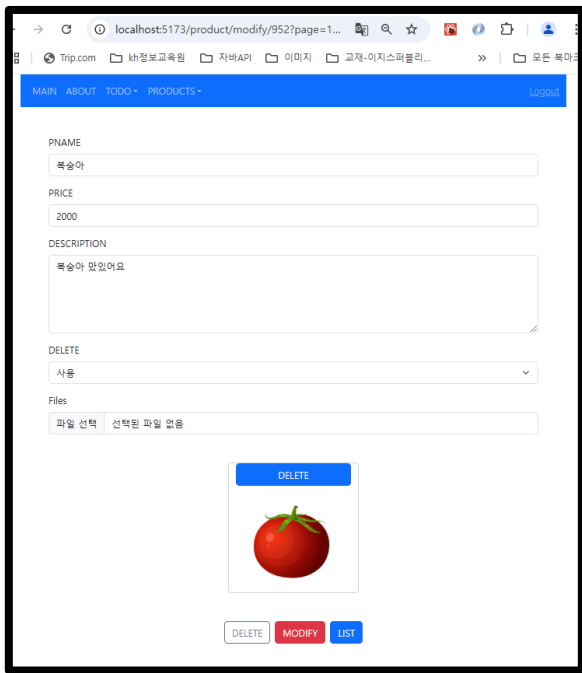
src/components/products/ModifyComponent.jsx

생략...

```
<Row className="d-flex justify-content-center mt-5 gap-4">  
  {product.uploadFileNames.map((imgFile, i) => (  
    <>  
      <Card style={{ width: '14rem', height: '14rem' }} key={i}>  
        <Button variant="primary" onClick={() => deleteOldImages(imgFile)} >  
          DELETE  
        </Button>  
        <Card.Body>  
          <img alt="img" style={{ width: '10rem' }}  
            src={` ${host}/api/products/view/s_ ${imgFile} `}/>  
        </Card.Body>  
      </Card>  
    </>  
  )  
  )  
</Row>
```

생략...

이미지의 삭제는 화면상에서만 반영되기 때문에 다시 조회하면 원래의 이미지가 보인다.



## 8) 새로운 이미지 파일의 추가와 수정

새로운 이미지 파일을 추가하는 부분은 기존의 신규 상품 등록과 동일하므로 수정 버튼을 작성하고 수정 버튼을 클릭할 때 파일 데이터를 추가한다. ModifyComponent에 이벤트 처리를 위한 handleClickModify() 함수를 추가하고 하단에 'Modify' 버튼을 추가해서 이벤트 처리를 완성한다. 마지막 부분에 버튼들을 추가하고 버튼 중에 'Modify'에 대해서 이벤트 처리를 추가한다.

### src/components/products/ModifyComponent.jsx

```
import { useEffect, useState, useRef } from 'react';
import { Container, Card, Row, Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';
import { deleteOne, getOne, putOne } from '../api/productApi';
import FetchingModal from '../common/FetchingModal';
import { API_SERVER_HOST } from '../api/todoApi';
import useCustomMove from '../hooks/useCustomMove';
import ResultModal from '../common/ResultModal';
```

```
const initState =
{
  pno: 0,
  pname: "",
  pdesc: "",
  price: 0,
  delFlag: false,
  uploadFileNames: [],
};
```

```
const host = API_SERVER_HOST;
```

```
const ModifyComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);
  const [fetching, setFetching] = useState(false);
  const uploadRef = useRef();
```

```

//결과모달(result 결과에 따라서 화면이동에 사용 result = 'Modified' or 'Deleted')
const [result, setResult] = useState(null);
//이동용 함수
const { moveToProductRead, moveToProductList } = useCustomMove();

useEffect(() => {
  setFetching(true);
  getOne(pno).then((data) => {
    setProduct(data);
    setFetching(false);
  });
}, [pno]);

const handleChangeProduct = (e) => {
  product[e.target.name] = e.target.value;
  setProduct({ ...product });
};

const deleteOldImages = (imageName) => {
  const resultFileNames =
    product.uploadFileNames.filter( fileName =>
      fileName !== imageName
    );
  product.uploadFileNames = resultFileNames;
  setProduct({ ...product });
};

const handleClickModify = () => {
  const files = uploadRef.current.files;
  const formData = new FormData();
  for (let i = 0; i < files.length; i++) {
    formData.append('files', files[i]);
  }
  //other data
  formData.append('pname', product.pname);
  formData.append('pdesc', product.pdesc);
  formData.append('price', product.price);
  formData.append('delFlag', product.delFlag);

  for (let i = 0; i < product.uploadFileNames.length; i++) {
    formData.append('uploadFileNames', product.uploadFileNames[i]);
  }
  setFetching(true)
  //수정 처리
  putOne(pno, formData).then((data) => {
    setResult('Modified');
    setFetching(false);
  });
};

const handleClickDelete = () => {

```

```

setFetching(true);
deleteOne(pno).then((data) => {
    setResult('Deleted');
    setFetching(false);
});
};

const closeModal = () => {
    if (result === 'Modified') {
        moveToProductRead(pno); // 조회 화면으로 이동
    } else if (result === 'Deleted') {
        moveToProductList({ page: 1 });
    }
    setResult(null);
};

return (
    <Container className="p-5">
        {fetching ? <FetchingModal /> : <>/>}

        <Form>
            <Form.Group className="mb-3">
                <Form.Label>PNAME</Form.Label>
                <Form.Control
                    name="pname"
                    defaultValue={product.pname}
                    type="text"
                    placeholder="Enter name"
                    onChange={handleChangeProduct}
                />
            </Form.Group>

            <Form.Group className="mb-3">
                <Form.Label>PRICE</Form.Label>
                <Form.Control
                    name="price"
                    type="number"
                    value={product.price}
                    placeholder="Enter price"
                    onChange={handleChangeProduct}
                />
            </Form.Group>
            <Form.Group className="mb-3">
                <Form.Label>DESCRIPTION</Form.Label>
                <Form.Control
                    name="pdesc"
                    defaultValue={product.pdesc}
                    as="textarea"
                    rows={5}
                    onChange={handleChangeProduct}
                />
            </Form.Group>

```

```

<Form.Group className="mb-3">
  <Form.Label>DELETE</Form.Label>
  <Form.Select
    name="delFlag"
    value={product.delFlag ? '사용' : '삭제'}
    onChange={handleChangeProduct}
  >
    <option value={false}>사용</option>
    <option value={true}>삭제</option>
  </Form.Select>
</Form.Group>
<Form.Group className="mb-3">
  <Form.Label>Files</Form.Label>
  <Form.Control ref={uploadRef} type="file" multiple="true" />
</Form.Group>
</Form>
<Row className="d-flex justify-content-center mt-5 gap-4">
  {product.uploadFileNames.map((imgFile, i) => (
    <>
      <Card style={{ width: '14rem', height: '14rem' }} key={i}>
        <Button
          variant="primary"
          onClick={() => deleteOldImages(imgFile)}
        >
          DELETE
        </Button>
        <Card.Body>
          <img
            alt="img"
            style={{ width: '10rem' }}
            src={` ${host}/api/products/view/s_${imgFile} `}
          />
        </Card.Body>
      </Card>
    </>
  )))
</Row>

<div className="d-flex justify-content-center gap-2 mt-5">
  <button className="btn btn-outline-secondary" type="button" onClick={handleClickDelete}>
    DELETE
  </button>
  <button className="btn btn-danger" type="button" onClick={handleClickModify}>
    MODIFY
  </button>
  <button className="btn btn-primary" type="text" onClick={moveToProductList}>
    LIST
  </button>
</div>
</Container>
);

```



```
};  
  
export default ModifyComponent;
```

화면에 새로운 이미지를 추가하고 'Modify' 버튼을 클릭해서 서버의 동작 여부를 확인한다(모달창 처리가 없어 화면에는 변화가 없다). 기존 이미지를 삭제하지 않는다면 '새로고침'을 통해서 추가된 이미지를 확인한다.

The screenshot shows a web browser window at localhost:5173/product/modify/952?page=1... The page has a blue header with navigation links: MAIN, ABOUT, TODO, PRODUCTS, and a Logout link. The main content area contains a form for modifying a product. The form fields are: PNAME (복숭아), PRICE (2000), and DESCRIPTION (복숭아 맛있어요). Below these is a DELETE dropdown menu with the option '사용' selected. There is also a file upload section with a '파일 선택' button and a '선택된 파일 없음' message. Below the form is a preview of a tomato image with a 'DELETE' button. At the bottom of the page are three buttons: 'DELETE', 'MODIFY', and 'LIST'.

### 9)수정 작업 후 모달창

Axios를 이용한 상품의 수정 처리가 완료되면 서버에서는 JSON 결과를 전송한다. ModifyComponent에 서는 모달창을 이용해서 결과를 보여주도록 수정한다. 모달창에서 필요한 상태와 모달창이 사라진 후 동작해야 하는 함수를 미리 정의한다. useCustomMove를 이용해서 수정은 상품의 조회 화면으로 이동하고, 상품의 삭제 후에는 목록 화면으로 이동해야 한다.

#### src/components/products/ModifyComponent.jsx

```
import { useEffect, useState, useRef } from 'react';
import { Container, Card, Row, Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';
import { deleteOne, getOne, putOne } from '../api/productApi';
import FetchingModal from '../common/FetchingModal';
import { API_SERVER_HOST } from '../api/todoApi';
import useCustomMove from '../hooks/useCustomMove';
import ResultModal from '../common/ResultModal';

const initState =
{ pno: 0,
  pname: "",
  pdesc: "",
  price: 0,
  delFlag: false,
  uploadFileNames: [],
};
const host = API_SERVER_HOST;

const ModifyComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);
  const [fetching, setFetching] = useState(false);
  const uploadRef = useRef();

  //결과 모달
  const [result, setResult] = useState(null);
  //이동용 함수
  const { moveToProductRead, moveToProductList } = useCustomMove();

  useEffect(() => {
    setFetching(true);
    getOne(pno).then((data) => {
      setProduct(data);
      setFetching(false);
    });
  }, [pno]);

  const handleChangeProduct = (e) => {
    product[e.target.name] = e.target.value;
    setProduct({ ...product });
  };

  const deleteOldImages = (imageName) => {
    const resultFileNames = product.uploadFileNames.filter(
```

```

    fileName => fileName !== imageName
  );
  product.uploadFileNames = resultFileNames;
  setProduct({ ...product });
};

const handleClickModify = () => {
  const files = uploadRef.current.files;
  const formData = new FormData();
  for (let i = 0; i < files.length; i++) {
    formData.append('files', files[i]);
  }
  //other data
  formData.append('pname', product.pname);
  formData.append('pdesc', product.pdesc);
  formData.append('price', product.price);
  formData.append('delFlag', product.delFlag);

  for (let i = 0; i < product.uploadFileNames.length; i++) {
    formData.append('uploadFileNames', product.uploadFileNames[i]);
  }
  setFetching(true)
  putOne(pno, formData).then((data) => {
    //수정 처리
    setResult('Modified');
    setFetching(false);
  });
};

const handleClickDelete = () => {
  setFetching(true);
  deleteOne(pno).then((data) => {
    setResult('Deleted');
    setFetching(false);
  });
};

const closeModal = () => {
  if (result === 'Modified') {
    moveToProductRead(pno); // 조회 화면으로 이동
  } else if (result === 'Deleted') {
    moveToProductList({ page: 1 });
  }
  setResult(null);
};

return (
  <Container className="p-5">
    {fetching ? <FetchingModal /> : <></>}
    {result ? (

```

```

<ResultModal
  title={` ${result}`}
  content={`정상적으로 처리되었습니다.`} //결과 모달창
  callbackFn={closeModal}
/>
): (
  <></>
)}

```

```

<Form>
  <Form.Group className="mb-3">
    <Form.Label>PNAME</Form.Label>
    <Form.Control
      name="pname"
      defaultValue={product.pname}
      type="text"
      placeholder="Enter name"
      onChange={handleChangeProduct}
    />
  </Form.Group>

  <Form.Group className="mb-3">
    <Form.Label>PRICE</Form.Label>
    <Form.Control
      name="price"
      type="number"
      value={product.price}
      placeholder="Enter price"
      onChange={handleChangeProduct}
    />
  </Form.Group>
  <Form.Group className="mb-3">
    <Form.Label>DESCRIPTION</Form.Label>
    <Form.Control
      name="pdesc"
      defaultValue={product.pdesc}
      as="textarea"
      rows={5}
      onChange={handleChangeProduct}
    />
  </Form.Group>
  <Form.Group className="mb-3">
    <Form.Label>DELETE</Form.Label>
    <Form.Select
      name="delFlag"
      defaultValue={product.delFlag ? '사용' : '삭제'}
      onChange={handleChangeProduct}
    >
    <option value={false}>사용</option>
    <option value={true}>삭제</option>
  </Form.Group>

```

```

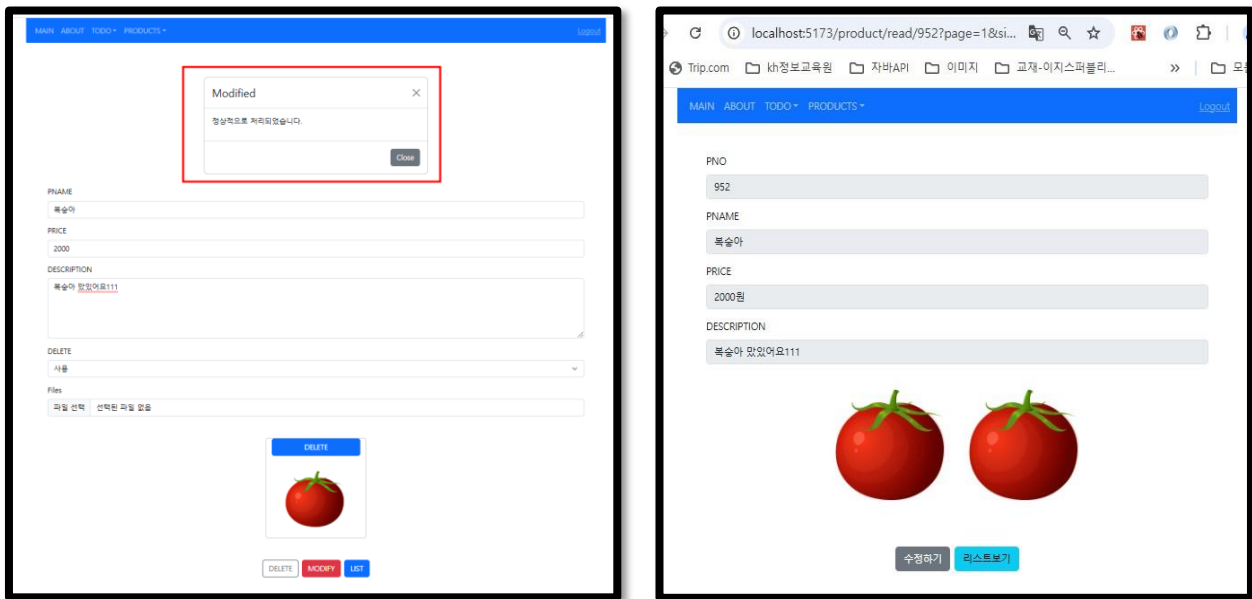
    </Form.Select>
  </Form.Group>
  <Form.Group className="mb-3">
    <Form.Label>Files</Form.Label>
    <Form.Control ref={uploadRef} type="file" multiple="true" />
  </Form.Group>
</Form>
<Row className="d-flex justify-content-center mt-5 gap-4">
  {product.uploadFileNames.map((imgFile, i) => (
    <>
      <Card style={{ width: '14rem', height: '14rem' }} key={i}>
        <Button
          variant="primary"
          onClick={() => deleteOldImages(imgFile)}
        >
          DELETE
        </Button>
        <Card.Body>
          <img
            alt="img"
            style={{ width: '10rem' }}
            src={`/${host}/api/products/view/s_${imgFile}`}
          />
        </Card.Body>
      </Card>
    </>
  ))}
</Row>

<div className="d-flex justify-content-center gap-2 mt-5">
  <button className="btn btn-outline-secondary" type="button" onClick={handleClickDelete}>
    DELETE
  </button>
  <button className="btn btn-danger" type="button" onClick={handleClickModify}>
    MODIFY
  </button>
  <button className="btn btn-primary" type="text" onClick={moveToProductList}>
    LIST
  </button>
</div>
</Container>
);
};

export default ModifyComponent;

```

상품을 수정하면 ResultModal을 보여주고 조회 화면으로 이동한다.



## 10) 삭제 버튼과 목록화면 이동

상품의 삭제는 하단의 'Delete' 버튼을 클릭하면 동작하는 함수를 작성하고 삭제 후 모달창을 보여준다.

목록 화면의 이동은 useCustomMove의 moveToList를 이용해서 버튼에 이벤트를 추가한다.

화면상에서 쿼리스트링이 유지된 상태로 목록 화면으로 이동한다.

ModifyComponent.jsx 전체 소스

```
src/components/products/ModifyComponent.jsx

import { useEffect, useState, useRef } from 'react';
import { Container, Card, Row, Button } from 'react-bootstrap';
import Form from 'react-bootstrap/Form';
import { deleteOne, getOne, putOne } from '../api/productApi';
import FetchingModal from '../common/FetchingModal';
import { API_SERVER_HOST } from '../api/todoApi';
import useCustomMove from '../hooks/useCustomMove';
import ResultModal from '../common/ResultModal';

const initState =
{
  pno: 0,
  pname: '',
  pdesc: '',
  price: 0,
  delFlag: false,
  uploadFileNames: [],
};

const host = API_SERVER_HOST;
```

```

const ModifyComponent = ({ pno }) => {
  const [product, setProduct] = useState(initState);
  const [fetching, setFetching] = useState(false);
  const uploadRef = useRef();

  //결과 모달
  const [result, setResult] = useState(null);
  //이동용 함수
  const { moveToProductRead, moveToProductList } = useCustomMove();

  useEffect(() =>
    { setFetching(true);
      getOne(pno).then((data) =>
        { setProduct(data);
          setFetching(false);
        });
    }, [pno]);

  const handleChangeProduct = (e) =>
    { product[e.target.name] = e.target.value;
      setProduct({ ...product });
    };

  const deleteOldImages = (imageName) => {
    const resultFileNames =
      product.uploadFileNames.filter( fileName =>
        fileName !== imageName
      );
    product.uploadFileNames = resultFileNames;
    setProduct({ ...product });
  };

  const handleClickModify = () =>
    { const files =
      uploadRef.current.files; const
      formData = new FormData(); for
      (let i = 0; i < files.length; i++)
      { formData.append('files', files[i]);
      }
      //other data
      formData.append('pname', product.pname);
      formData.append('pdesc', product.pdesc);
    }

```

```

formData.append('price', product.price);
formData.append('delFlag', product.delFlag);

for (let i = 0; i < product.uploadFileNames.length; i++)
  { formData.append('uploadFileNames', product.uploadFileNames[i]);
  }
//fetching setFetching(true)

putOne(pno, formData).then((data) => {
  //수정 처리
  setResult('Modified');
  setFetching(false);
});
};

const handleClickDelete = () => {
  setFetching(true);
  deleteOne(pno).then((data) => {
    setResult('Deleted');
    setFetching(false);
  });
};

const closeModal = () => {
  if (result === 'Modified') {
    moveToProductRead(pno); // 조회 화면으로 이동
  } else if (result === 'Deleted') {
    moveToProductList({ page: 1 });
  }
  setResult(null);
};

return (
  <Container className="p-5">
    {fetching ? <FetchingModal /> : <></>}
    {result ? (
      <ResultModal
        title={`${result}`}
        content={`정상적으로 처리되었습니다.`} //결과 모달창
        callbackFn={closeModal}
      />

```



```

):{
  <></>
}}

<Form>
  <Form.Group className="mb-3">
    <Form.Label>PNAME</Form.Label>
    <Form.Control
      name="pname"
      defaultValue={product.pname}
      type="text"
      placeholder="Enter name"
      onChange={handleChangeProduct}
    />
  </Form.Group>

  <Form.Group className="mb-3">
    <Form.Label>PRICE</Form.Label>
    <Form.Control
      name="price"
      type="number"
      value={product.price}
      placeholder="Enter price"
      onChange={handleChangeProduct}
    />
  </Form.Group>

  <Form.Group className="mb-3">
    <Form.Label>DESCRIPTION</Form.Label>
    <Form.Control
      name="pdesc"
      defaultValue={product.pdesc}
      as="textarea"
      rows={5}
      onChange={handleChangeProduct}
    />
  </Form.Group>

  <Form.Group className="mb-3">
    <Form.Label>DELETE</Form.Label>
    <Form.Select
      name="delFlag"
      defaultValue={product.delFlag ? '사용' : '삭제'}

```

```

    onChange={handleChangeProduct}
  >
    <option value={false}>사용</option>
    <option value={true}>삭제</option>
  </Form.Select>
</Form.Group>
<Form.Group className="mb-3">
  <Form.Label>Files</Form.Label>
  <Form.Control ref={uploadRef} type="file" multiple="true" />
</Form.Group>
</Form>
<Row className="d-flex justify-content-center mt-5 gap-4">
  {product.uploadFileNames.map((imgFile, i) => (
    <>
      <Card style={{ width: '14rem', height: '14rem' }} key={i}>
        <Button
          variant="primary"
          onClick={() => deleteOldImages(imgFile)}
        >
          DELETE
        </Button>
        <Card.Body>
          <img
            alt="img"
            style={{ width: '10rem' }}
            src={`/${host}/api/products/view/s_${imgFile}`}
          />
        </Card.Body>
      </Card>
    </>
  ))}
</Row>

<div className="d-flex justify-content-center gap-2 mt-5">
  <button className="btn btn-outline-secondary" type="button" onClick={handleClickDelete} >
    DELETE
  </button>
  <button className="btn btn-danger" type="button" onClick={handleClickModify} >
    MODIFY
  </button>
  <button className="btn btn-primary" type="text" onClick={moveToProductList} >

```

```

    LIST
  </button>
</div>
</Container>
);
};

export default ModifyComponent;

```

