

# React

# REACT -

## 목차

---

01. REACT	3
> 15. 단어장 만들기	3
> 16. 라우터 Link to 사용하기	13
BrowserRouter	13
Routes	13
Route	14
> 17. Rest Api를 만들어 보자.	20
> 26.fetch 사용법	51
설명	53
기본 문법	53
주요 개념	54
사용 예제	54
1. GET 요청	54
2. POST 요청	55
3. Error Handling	55
비동기/대기 (async/await)	56
결론	57
> 29. useReducer	57
사용자 정의 훅 (Custom Hook)	59
주요 포인트:	60
> 30.	62

# 01. REACT

## > 15. 단어장 만들기

다음과 같은 단어 장을 만들어 보자.



### 1. 프로젝트 설치하기

`npx create-react-app voca` voca폴더에 설치

### 2. index.css 를 미리 만들어 놓았다.

```
body {  
  margin: 0;  
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto",  
  "Oxygen",  
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",  
    sans-serif;  
  -webkit-font-smoothing: antialiased;  
  -moz-osx-font-smoothing: grayscale;  
  font-size: 20px;  
}
```

```
ol,
ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
    monospace;
}

a {
  text-decoration: none;
  color: #333;
}

.App {
  width: 800px;
  margin: 0 auto;
}

.header {
  position: relative;
}

.header .menu {
  position: absolute;
  top: 10px;
  right: 0;
}

.header .link {
  border: 1px solid #333;
  padding: 10px;
  margin-left: 10px;
  background-color: #efefef;
}
```

```
    font-weight: bold;
    border-radius: 4px;
}
```

```
.list_day {
    display: flex;
    flex-wrap: wrap;
}
```

```
.list_day li {
    flex: 20% 0 0;
    box-sizing: border-box;
    padding: 10px;
}
```

```
.list_day a {
    display: block;
    padding: 20px 0;
    font-weight: bold;
    color: #fff;
    text-align: center;
    border-radius: 10px;
    background-color: dodgerblue;
}
```

```
table {
    border-collapse: collapse;
    width: 100%;
}
```

```
table td {
    width: 25%;
    height: 70px;
    border: 1px solid #ccc;
    text-align: center;
    font-size: 26px;
}
```

```
table td:first-child {  
    width: 10%;  
}
```

```
.off td {  
    background: #eee;  
    color: #ccc;  
}
```

```
.btn_del {  
    margin-left: 10px;  
    color: #fff;  
    background-color: firebrick;  
}
```

```
button {  
    padding: 10px;  
    font-weight: bold;  
    font-size: 18px;  
    cursor: pointer;  
    border: 0 none;  
    border-radius: 6px;  
    padding: 10px 20px;  
    color: #fff;  
    background-color: dodgerblue;  
}
```

```
.input_area {  
    margin-bottom: 10px;  
}
```

```
.input_area label {  
    display: block;  
    margin-bottom: 10px;  
}
```

```
.input_area input {
```

```
width: 400px;
height: 40px;
font-size: 20px;
padding: 0 10px;
}
```

```
.input_area select {
width: 400px;
height: 40px;
font-size: 20px;
}
```

3. component 폴더 생성후 component 폴더에 header.js파일을 만들고 다음 코드를 작성하자.

```
export default function Header() {
  return (
    <div className="header">
      <h1>
        <a href="/">토익 영단어(고급)</a>
      </h1>
      <div className="menu">
        <a href="#x" className="link">
          단어 추가
        </a>
        <a href="#x" className="link">
          Day 추가
        </a>
      </div>
    </div>
  );
}
```

4.App.js에 다음 코드 추가하고 회면을 확인하자

```
import logo from './logo.svg';
import './App.css';
import Header from './component/Header';

function App() {
```

```

return (
  <div className="App">
    <Header/>
  </div>
);
}

```

```
export default App;
```

5. db 폴더를 생성하고 db 폴더와 data.json 데이터를 만든다. 파일을 통해서 단어장 프로그램에 사용될 단어들을 가지고 있다.

```

{
  "days": [
    { "id": 1, "day": 1 },
    { "id": 2, "day": 2 },
    { "id": 3, "day": 3 }
  ],
  "words": [
    {
      "id": 1,
      "day": 1,
      "eng": "book",
      "kor": "책",
      "isDone": false
    },
    {
      "id": 2,
      "day": 1,
      "eng": "apple",
      "kor": "사과",
      "isDone": false
    },
    {
      "id": 3,
      "day": 2,
      "eng": "car",
      "kor": "자동차",
      "isDone": false
    }
  ]
}

```



```

    },
    {
      "id": 4,
      "day": 2,
      "eng": "pen",
      "kor": "펜",
      "isDone": false
    },
    {
      "id": 5,
      "day": 3,
      "eng": "school",
      "kor": "학교",
      "isDone": false
    },
    {
      "id": 6,
      "day": 3,
      "eng": "pencil",
      "kor": "연필",
      "isDone": false
    }
  ]
}

```

## 5. component/DayList컴포넌트 만들기

DayList.js파일에 다음 코드를 추가해 보자.

더미데이터를 불러와 단어 장을 만든다.

반복되는 요소에 key가 필요하다.

```

import dummy from "../db/data.json";

export default function DayList() {
  return (
    <ul className="list_day">
      {dummy.days.map(day => (

```

```

        <li key={day.id}>
          Day {day.day}
        </li>
      )}
    </ul>
  );
}

```

App.js를 변경한다.

```

import logo from './logo.svg';
import './App.css';
import Header from './component/Header';
import DayList from './component/DayList';

function App() {
  return (
    <div className="App">
      <Header/>
      <DayList/>
    </div>
  );
}

export default App;

```

6. 특정 날을 클릭 하였을때 해당 날과 관련된 데이터가 화면에 나오는 Day 페이지를 만들어 볼 예정이다. 일단은 모든 데이터를 출력하는 페이지를 다음 코드를 이용해서 만들어 보자. component/Day.js에 만들면 된다.

```

import dummy from "../db/data.json";
export default function Day() {
  return (
    <>
      <h2>Day All</h2>
      <table>
        <tbody>
          {dummy.words.map(word => (
            <tr key={word.id}>
              <td>{word.eng}</td>
              <td>{word.kor}</td>

```

```

        </tr>
      )})
    </tbody>
  </table>
</>
);
}

```

app.js를 변경해서 화면을 확인해 보자.

토익 영단어(고급) <span>단어 추가</span> <span>Day 추가</span>	
Day 1	Day 2
Day 3	
Day All	
book	책
apple	사과
car	자동차
pen	펜
school	학교
pencil	연필

App.js에 다음을 추가하고 화면을 확인해 보자.

```

import logo from './logo.svg';
import './App.css';
import Header from './component/Header';
import DayList from './component/DayList';
import Day from './component/Day';

function App() {
  return (
    <div className="App">
      <Header/>
      <DayList/>
      <Day/>
    </div>
  );
}

export default App;

```

7. day 1 에 해당하는 1일차 데이터만 뽑아 보자.

```
import dummy from "../db/data.json";
```

```
export default function Day() {
  const day = 1;
  const wordList = dummy.words.filter(word => word.day === Number(day));

  return (
    <>
    <h2>Day {day}</h2>
    <table>
      <tbody>
        {wordList.map(word => (
          <tr key={word.id}>
            <td>{word.eng}</td>
            <td>{word.kor}</td>
          </tr>
        ))}
      </tbody>
    </table>
    </>
  );
}
```

---

## > 16. 라우터 Link to 사용하기

---

```
<BrowserRouter>
  <div className="App">
    <Header />
    <Routes>
      <Route path="/" element={<DayList />} />
      <Route path="/day/:day" element={<Day />} />
      <Route path="*" element={<EmptyPage />} />
    </Routes>
  </div>
</BrowserRouter>
```

Link to는 html a태그 처럼 다른 페이지로 이동하는 기능이다 리액트에서 싱글페이지로 동작하는 형태로 다른 페이지 이동하는 것처럼 만들때 사용한다. 라우터를 구현하기 위해서는 react-router-dom를 설치해야 한다.

```
npm install react-router-dom
```

react-router-dom은 리액트에서 SPA(Single Page Application)를 구현하기 위해 사용하는 라이브러리입니다. 이 라이브러리를 사용하면 URL에 따라 다른 컴포넌트를 렌더링할 수 있습니다. 위 코드에서는 react-router-dom의 BrowserRouter, Routes, Route 등을 사용하여 라우팅 기능을 구현하고 있습니다. 각 요소에 대한 설명은 다음과 같습니다.

### BrowserRouter

- 역할: BrowserRouter는 전체 애플리케이션을 감싸며 브라우저의 URL 변화를 감지하고 그에 따라 적절한 컴포넌트를 렌더링하는 역할을 합니다.
- 특징: HTML5의 history API를 사용하여 페이지 새로고침 없이 URL을 변경하고 해당 URL에 맞는 컴포넌트를 렌더링합니다.

### Routes

- 역할: Routes는 Route들의 집합입니다. 현재 URL에 따라 적합한 Route를 찾아 렌더링합니다.
- 특징: Routes 컴포넌트는 이전 버전에서 사용되던 Switch의 역할을 대체합니다. URL과 일치하는 첫 번째 Route를 렌더링하는 방식입니다.

## Route

- 역할: **Route**는 URL 경로에 따라 렌더링할 컴포넌트를 결정합니다. **path** 속성에는 해당 **Route**가 렌더링될 URL 경로를, **element** 속성에는 해당 경로에서 렌더링될 컴포넌트를 지정합니다.
- 동적 라우팅: **/day/:day**처럼 콜론(:)을 사용하면 동적 라우팅이 가능합니다. 여기서 **:day**는 URL의 일부로 전달되는 값이며, 해당 값을 컴포넌트에서 **useParams** 훅을 통해 접근할 수 있습니다.

### 1. EmptyPage.js 만들기

component 폴더에 EmptyPage.js 파일에 다음 코드를 붙여 넣는다.

EmptyPage는 사용자가 잘못된 경로를 요청했을 때 보여줄 페이지이다.

```
import { Link } from "react-router-dom";

export default function EmptyPage() {
  return (
    <>
      <h2>잘못된 접근입니다.</h2>
      <Link to="/">돌아가기</Link>
    </>
  );
}
```

### 2. app.js 변경하기

Router를 이용해서 여러개의 주소중 원하는 화면을 선택해서 보여주게 만들었다.

```
import Day from "../component/Day";
import DayList from "../component/DayList";
import Header from "../component/Header";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import EmptyPage from "../component/EmptyPage";

function App() {
  return (
    <BrowserRouter>
      <div className="App">
```

```

    <Header />
    <Routes>
      <Route path="/" element={<DayList />} />
      <Route path="/day/:day" element={<Day />} />
      <Route path="*" element={<EmptyPage />} />
    </Routes>
  </div>
</BrowserRouter>
);
}

```

```
export default App;
```

실행한 다음 주소창에 잘못된 주소를 넣으면 EmptyPage를 확인할 수 있다.

<http://localhost:3000/day/qwer>

3. Header.js에 홈으로 가는 Link to를 추가해 보자.

```

import { Link } from "react-router-dom";
export default function Header() {
  return (
    <div className="header">
      <h1>
        <Link to="/">토익 영단어(고급)</Link>
      </h1>
      <div className="menu">
        <a href="#x" className="link">
          단어 추가
        </a>
        <a href="#x" className="link">
          Day 추가
        </a>
      </div>
    </div>
  );
}

```

3. `daylist.js`에 다음 코드를 추가하여 각각의 날짜에 해당하는 단어를 볼수 있는 페이지 이동 `Link to`를 추가해 보자.

<http://localhost:3000/day/:day> 과 같은 형태로 1일차, 2일차, 3일차 주소를 다음과 같이 만들었다.

<http://localhost:3000/day/1>

<http://localhost:3000/day/2>

<http://localhost:3000/day/3>

```
import { Link } from "react-router-dom";
import dummy from "../db/data.json";

export default function DayList() {
  return (
    <ul className="list_day">
      {dummy.days.map(day => (
        <li key={day.id}>
          <Link to={`/day/${day.day}`}>Day {day.day}</Link>
        </li>
      ))}
    </ul>
  );
}
```

4. `Day.js`를 다음과 같이 주소 경로에 따라 다르게 처리한다.

`useParams()`를 이용해서 주소에서 원하는 값을 추출한다.

<http://localhost:3000/day/:day> 과 같은 형태로

<http://localhost:3000/day/1>

<http://localhost:3000/day/2>



<http://localhost:3000/day/3>

:day를 추출하는 방법은 다음과 같다.

```
const {day} =useParams();

import { useParams } from "react-router-dom";
import dummy from "../db/data.json";

export default function Day() {
  //const day =1;
  //const day =useParams();//useParam의 리턴값은 객체여서 에러가난다.
  const {day} =useParams();//객체의 키값중 day의 값을 변수day에 넣는다.
  const wordList = dummy.words.filter(word => word.day === Number(day));
  return (
    <>
    <h2>Day {day}</h2>
    <table>
      <tbody>
        {wordList.map(word => (
          <tr key={word.id}>
            <td>{word.eng}</td>
            <td>{word.kor}</td>
          </tr>
        ))}
      </tbody>
    </table>
    </>
  );
}
```

// Day.jsx 변경하기

```
import { Link } from "react-router-dom";
import dummy from "../db/data.json";
export default function DayList(){
  return(
    <ul className="list_day">
      {
```

```

        dummy.days.map(day=>(
            <li key={day.id}>
                <Link
to={` /day/${day.day}`}>Day{day.day}</Link>
                </li>
            ))
        }
    </ul>
);
}

```

word꾸미기

## 토익 영단어(고급)

단어 추가

Day 추가

### Day 2



car	자동차
pen	펜

word.js

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

```

export default function Word({ word }) {
    const [isShow, setIsShow] = useState(false);
    const [isDone, setIsDone] = useState(word.isDone);

```

```

    function toggleShow() {
        setIsShow(!isShow);
    }

```

```

function toggleDone() {
  setIsDone(!isDone);
}

return (
  <tr className={isDone ? "off" : ""}>
    <td>
      <input type="checkbox" checked={isDone} onChange={toggleDone} />
    </td>
    <td>{word.eng}</td>
    <td>{isShow && word.kor}</td>
    <td>
      <button onClick={toggleShow}>뜻 {isShow ? "숨기기" : "보기"}</button>
      <button className="btn_del">삭제</button>
    </td>
  </tr>
);
}

```

day.js에서 word를 추가한다.

day.js

```

import dummy from "../db/data.json";
import { useParams } from "react-router-dom";
import Word from "./Word";

export default function Day() {
  const { day } = useParams();
  const wordList = dummy.words.filter(word => word.day === Number(day));

  return (
    <>
      <h2>Day {day}</h2>
      <table>

```

```
    <tbody>
      {wordList.map(word => (
        <Word word={word} key={word.id} />
      ))}
    </tbody>
  </table>
</>
);
}
```

---

## > 17. Rest Api를 만들어 보자.

---

json-server모듈은 json파일을 가지고 REST API를 자동으로 만들어서 사용해보자.

```
npm install -g json-server
```

상단을 이용해서 모듈을 설치한다.

관리자 권한을 이용해서 cmd창을 실행하고 해당 경로로 이동해서 .json파일 서버를 실행한다.

```
json-server --watch ./src/db/data.json --port 3010
```

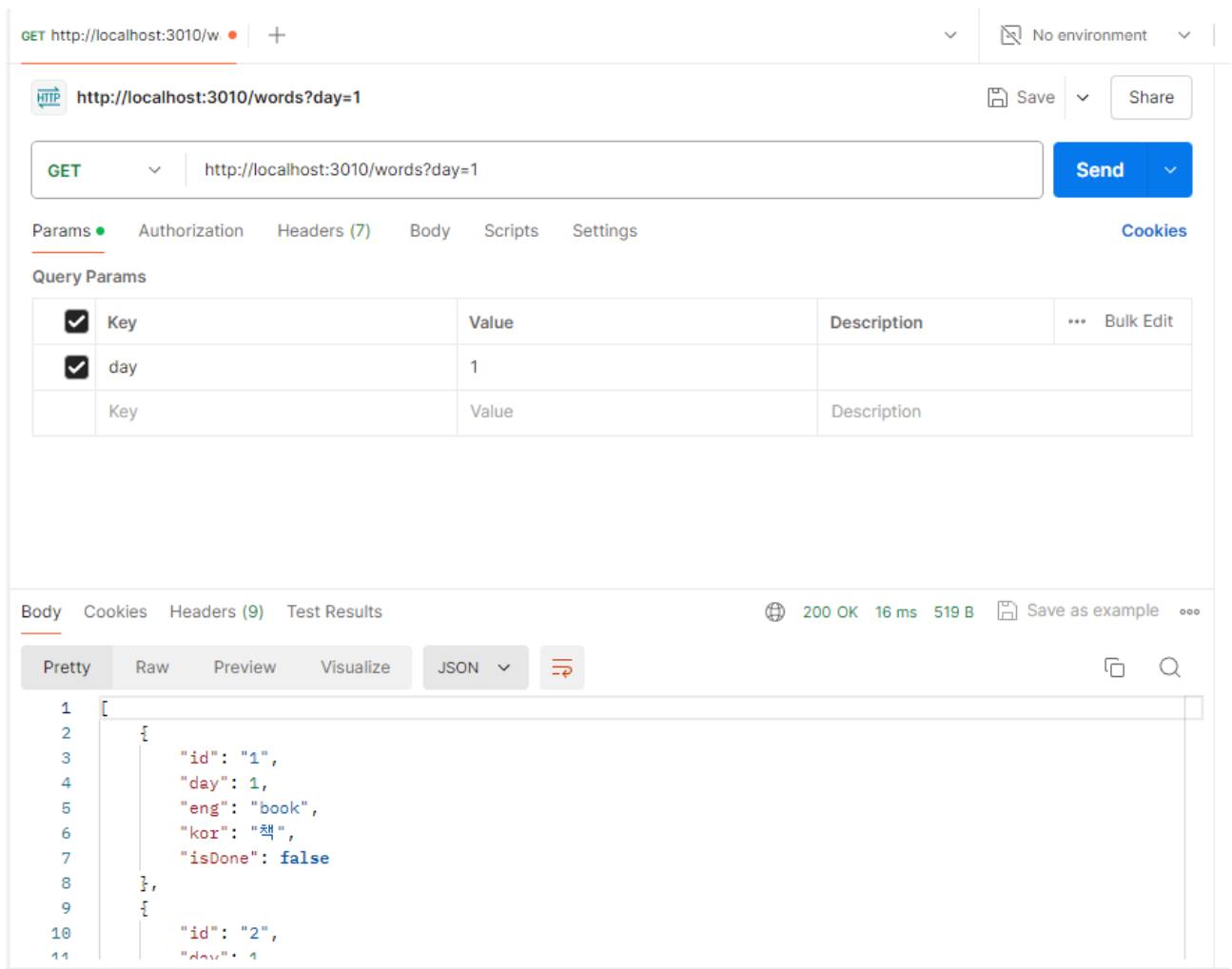
브라우저에서 다음 주소를 치면 **rest**서버가 제대로 동작하는지 확인할 수 있다.

<http://localhost:3010/days>

<http://localhost:3010/words>

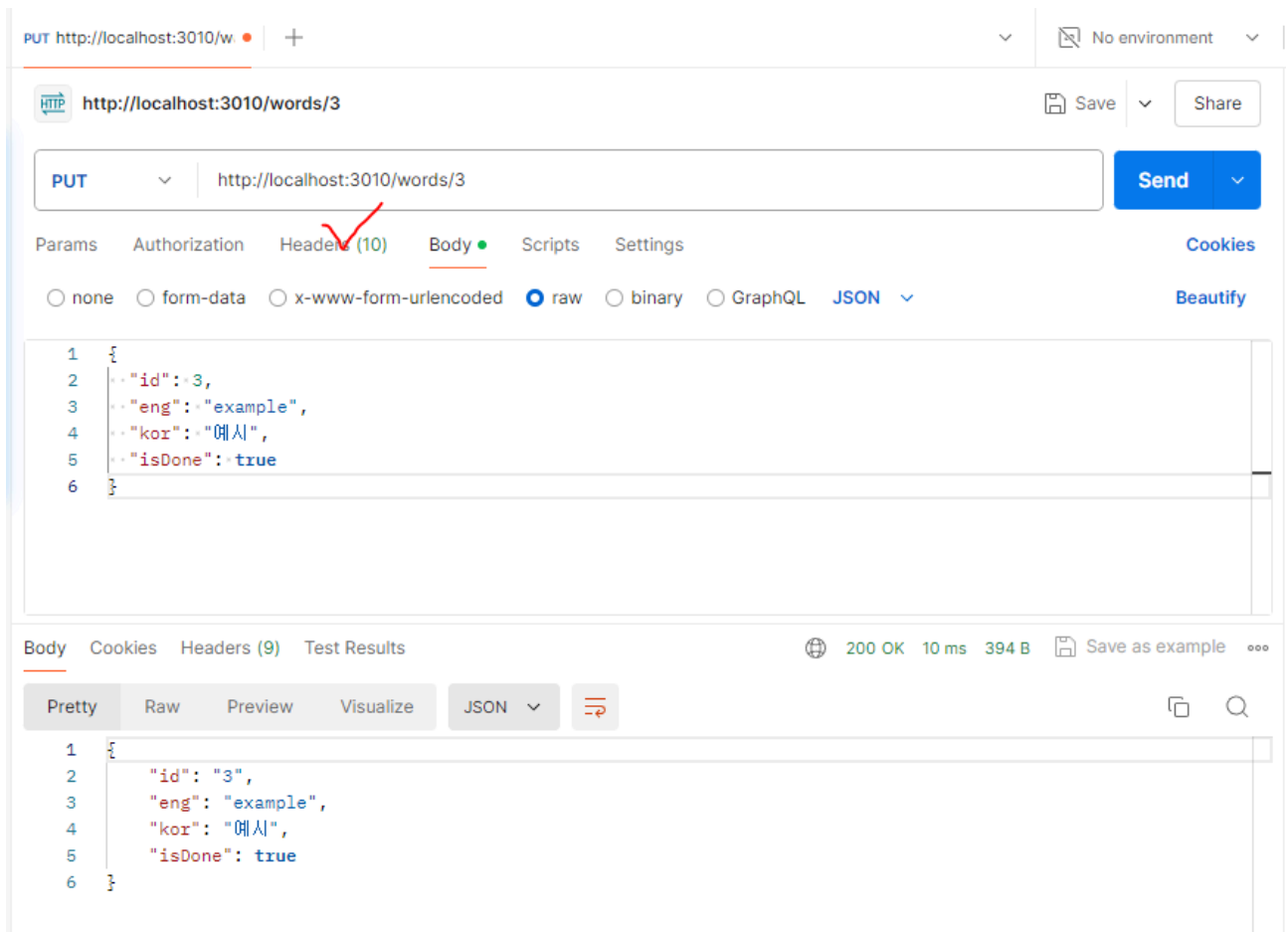
**postman**를 실행시키고 다음을 입력하여 1일차 단어를 검색해서 받아 보자.

<http://localhost:3010/words?day=1>



데이터 수정 방법

- <http://localhost:3010/words/3>



### 1. HTTP 메서드 설정

- **Method:** PUT

### 2. URL 입력

- **URL:** `http://localhost:3001/words/{word.id}`
  - 여기서 `{word.id}`는 실제 `word.id` 값으로 대체되어야 합니다. 예를 들어 `word.id`가 3이라면 URL은 `http://localhost:3001/words/3`이 됩니다.

### 3. Headers 설정

- **Content-Type:** `application/json`

Postman의 Headers 탭에 다음과 같이 설정:

makefile

코드 복사

**Key:** Content-Type

**Value:** `application/json`

○

#### 4. Body 입력

- **Body** 탭에서 **raw** 옵션을 선택하고, 데이터 형식을 **JSON**으로 지정합니다.
- **JSON** 형식으로 요청 본문을 작성합니다. 요청 본문의 내용은 **word** 객체의 속성들과 **isDone** 상태를 포함해야 합니다.

예시:

```
{
  "id": 3,
  "day": 2,
  "eng": "example",
  "kor": "예시",
  "isDone": true
}
```

- 이 예시에서는 **word.id**가 3, **word.eng**가 "example", **word.kor**가 "예시", 그리고 **isDone** 상태가 **true**로 되어 있습니다. 실제 **word** 객체의 내용에 맞게 변경해야 합니다.

해당 데이터를 새로운 항목으로 서버에 \*\*입력(생성)\*\*하려면 **HTTP POST** 메서드를 사용해야 합니다. **POST**는 서버에 새로운 리소스를 추가하는 데 사용됩니다.

Postman에서 데이터를 입력하는 방법:

1. **HTTP** 메서드 선택: **POST**
2. **URL** 입력: **http://localhost:3001/words**
  1. 이 URL은 **words** 엔드포인트로 새로운 데이터를 추가할 때 사용됩니다.
3. **Headers** 설정:

1. **Content-Type:** `application/json`
  - **Key:** `Content-Type`
  - **Value:** `application/json`
4. **Body** 설정:
  1. **Body** 탭을 클릭합니다.
  2. **raw** 옵션을 선택합니다.
  3. **raw** 옆의 드롭다운 메뉴에서 **JSON**을 선택합니다.
  4. 아래의 **JSON** 데이터를 입력합니다.

입력할 JSON Body 예시:

json

코드 복사

```
{  
  "id": "3",  
  "day": 2,  
  "eng": "bike",  
  "kor": "자전거",  
  "isDone": false  
}
```

5. **Send** 버튼 클릭: 모든 설정을 마친 후 **Send** 버튼을 클릭하여 요청을 서버로 전송합니다.

Postman에서 삭제하는 방법:

1. **HTTP** 메서드 선택: `DELETE`
2. **URL** 입력: `http://localhost:3010/words/3`
  - 여기서 **3**은 삭제하려는 데이터의 `id` 값입니다.
3. **Headers**:
  - 일반적으로 `DELETE` 요청에는 추가적인 **Body**나 **Headers** 설정이 필요하지 않습니다. 다만, 특정 **API**에서 인증이 필요한 경우, `Authorization` 헤더 등을 추가해야 할 수 있습니다.
4. **Body** 설정: `DELETE` 요청에는 보통 **Body**가 필요 없으므로 설정하지 않아도 됩니다.



5. **Send** 버튼 클릭: 모든 설정을 마친 후 **Send** 버튼을 눌러 요청을 전송합니다.

요약:

- **Method:** DELETE
- **URL:** <http://localhost:3001/words/3>
- **Headers:** 보통 필요 없음 (특정한 인증 요구 사항이 없는 경우)
- **Body:** 필요 없음

이렇게 하면 서버에서 **id: 3**인 데이터가 삭제됩니다.

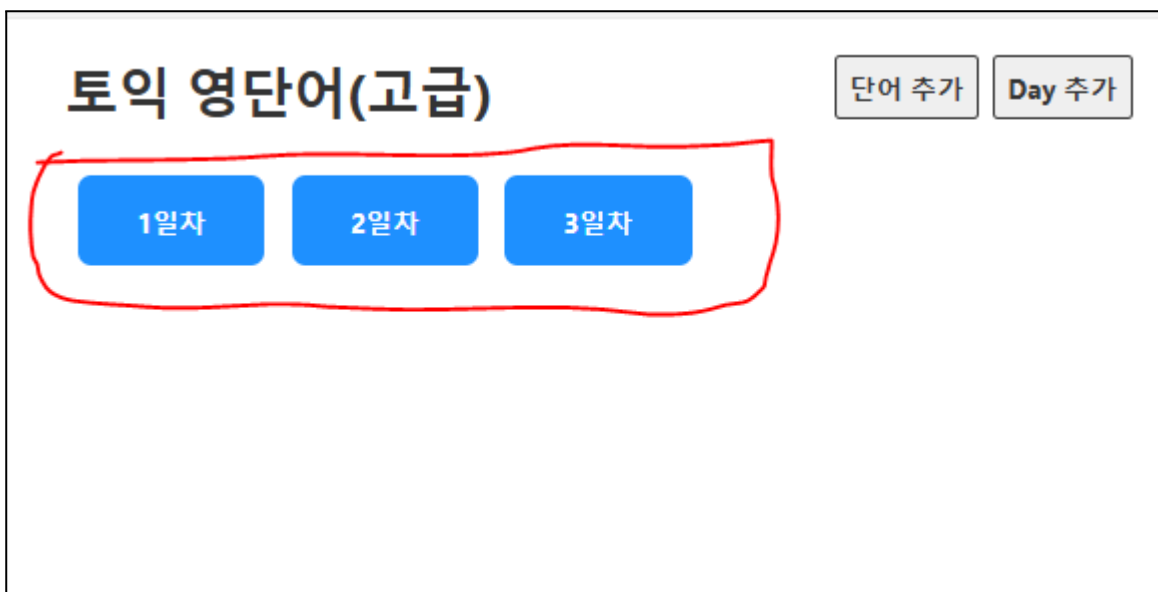
웃○

6.

---

> 18.

---



다음 이미지는 버튼을 클릭하면 **onclick** 메소드가 실행되고 **useState**의 **count**값이 변경되어 화면이 갱신된 후 **useEffect**가 실행된다.

json에서 읽어 오는 형태로 day.js와 daylist.js를 변경해 보자.

dayList.js

```
import { Link } from "react-router-dom";
import dummy from "../db/data.json";

export default function DayList() {
  return (
    <ul className="list_day">
      {dummy.days.map(day => (
        <li key={day.id}>
          <Link to={` /day/${day.day}`}>Day {day.day}</Link>
        </li>
      ))}
    </ul>
  );
}
```

변경후

```
import { useEffect, useState } from "react";
import { Link } from "react-router-dom";
//import useFetch from "../hooks/useFetch";

export default function DayList() {
  //const days = useFetch("http://localhost:3001/days");
  const [days, setDays] = useState([]);

  useEffect(() => {
```

```

fetch("http://localhost:3010/days")
  .then(res => {
    return res.json();
  })
  .then(data => {
    setDays(data);
  });
}, []);

return (
  <ul className="list_day">
    {days.map(day => (
      <li key={day.id}>
        <Link to={` /day/${day.day}`}>Day {day.day}</Link>
      </li>
    ))}
  </ul>
);
}

```

## 토익 영단어(고급)

단어 추가

Day 추가

Day 2

day.js

<input type="checkbox"/>	pen		뜻 보기	삭제
<input type="checkbox"/>	bike		뜻 보기	삭제

day.js

```

import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import Word from "../Word";
//import useFetch from "../hooks/useFetch";

```

```

export default function Day() {
  const { day } = useParams();

```

```

//const words = useFetch(`http://localhost:3001/words?day=${day}`);
const [words, setWords] = useState([]);

useEffect(() => {
  fetch(`http://localhost:3010/words?day=${day}`)
    .then(res => {
      return res.json();
    })
    .then(data => {
      setWords(data);
    });
}, [day]);

return (
  <>
    <h2>Day {day}</h2>
    <table>
      <tbody>
        {words.map(word => (
          <Word word={word} key={word.id} />
        ))}
      </tbody>
    </table>
  </>
);
}

```

---

## > 19. 사용자 훅 만들기

---

다음에 만든 `useFetch` 훅은 이미 존재하는 훅이 아니고 사용자가 만든 훅으로 반복적으로 사용되는 훅을 사용자가 정의해서 만들 수 있다.

hooks/useFetch.js 의 코드 내용

```

import { useEffect, useState } from "react";

export default function useFetch(url) {
  const [data, setData] = useState([]);

```

```

useEffect(() => {
  fetch(url)
    .then(res => {
      return res.json();
    })
    .then(data => {
      setData(data);
    });
}, [url]);

return data;
}

```

변경내용

```

export default function DayList() {
  const days = useFetch("http://localhost:3001/days");
  // const [days, setDays] = useState([]);

  // useEffect(() => {
  //   fetch("http://localhost:3001/days")
  //     .then(res => {
  //       return res.json();
  //     })
  //     .then(data => {
  //       setDays(data);
  //     });
  // }, []);

export default function Day() {
  const { day } = useParams();
  const words = useFetch(`http://localhost:3001/words?day=${day}`);
  // const [words, setWords] = useState([]);

```

```
// useEffect(() => {
//   fetch(`http://localhost:3001/words?day=${day}`)
//     .then(res => {
//       return res.json();
//     })
//     .then(data => {
//       setWords(data);
//     });
// }, [day]);
```

---

## > 20. 단어 수정 삭제하기

---

수정 작업을 해보자.

체크를 하면 `isDone`를 변경하는 작업을 진행할 예정이다.



`isDone`수정 작업

```
function toggleDone() {
  fetch(`http://localhost:3001/words/${word.id}`, {
```

```

method: "PUT",
headers: {
  "Content-Type": "application/json",
},
body: JSON.stringify({
  ...word,
  isDone: !isDone,
}),
}).then(res => {
  if (res.ok) {
    setIsDone(!isDone);
  }
});
}

```

삭제

word.js

화면 갱신을 위한 word useState추가

```

export default function Word({ word:w }) {
  const [word,setWord] = useState(w);

```

버튼 이벤트 추가

```

<button onClick={del} className="btn_del">삭제</button>

```

이벤트 핸들러 추가

```

function del() {
  if (window.confirm("삭제 하시겠습니까?")) {
    fetch(`http://localhost:3010/words/${word.id}`, {
      method: "DELETE",
    }).then(res => {
      if (res.ok) {
        setWord({ id: 0 });
      }
    });
  }
}
}

```

```
if (word.id === 0) {  
  return null;  
}
```

삭제 업그레이드

```
<button onClick={del} className="btn_del">삭제</button>
```

```
const [loading, setLoading] = useState(false);
```

```
function del() {  
  if (window.confirm("삭제 하시겠습니까?")) {  
    setLoading(true);  
    fetch(`http://localhost:3001/words/${word.id}`, {  
      method: "DELETE",  
    })  
      .then((res) => {  
        if (res.ok) {  
          setWord({ id: 0 });  
        } else {  
          alert("삭제 요청이 실패했습니다.");  
        }  
      })  
      .catch((error) => {  
        console.error("Error:", error);  
        alert("서버와의 통신에 문제가 발생했습니다.");  
      })  
      .finally(() => setLoading(false)); // 요청 완료 후 로딩 상태 해제  
  }  
}
```

```
if (word.id === 0) {  
  return null; // 삭제된 경우 아무것도 렌더링하지 않음  
}
```

```
return (
```



```

<div>
  <p>{word.name}</p>
  <button onClick={del} className="btn_del" disabled={loading}>
    {loading ? "삭제 중..." : "삭제"}
  </button>
</div>
);

```

---

> 21.

---

app.js 라우터

```

import Day from "../component/Day";
import DayList from "../component/DayList";
import Header from "../component/Header";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import EmptyPage from "../component/EmptyPage";

function App() {
  return (
    <BrowserRouter>
    <div className="App">
      <Header />
      <Routes>
        <Route path="/" element={<DayList />} />
        <Route path="/day/:day" element={<Day />} />
        <Route path="*" element={<EmptyPage />} />
        <Route path="/create_word" element={<CreateWord />} />
        <Route path="/create_day" element={<CreateDay />} />
        <Route path="*" element={<EmptyPage />} />
      </Routes>
    </div>
    </BrowserRouter>
  );
}

```

```
export default App;
```

Header.js

```
import { Link } from "react-router-dom";

export default function Header() {
  return (
    <div className="header">
      <h1>
        <Link to="/">토크 영단어(고급)</Link>
      </h1>
      <div className="menu">
        <Link to="/create_word" className="link">
          단어 추가
        </Link>
        <Link to="/create_day" className="link">
          Day 추가
        </Link>
      </div>
    </div>
  );
}
```

단어와 일 입력

CreateDay.js

```
import { useNavigate } from 'react-router-dom';
import useFetch from "../hooks/useFetch";

export default function CreateDay() {
  const days = useFetch("http://localhost:3001/days");
  const navigate = useNavigate();

  function addDay() {
    fetch(`http://localhost:3001/days/`, {
```

```

    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      day: days.length + 1,
    }),
  }).then(res => {
    if (res.ok) {
      alert("생성이 완료 되었습니다");
      navigate(`/`);
    }
  });
}
return (
  <div>
    <h3>현재 일수 : {days.length}일</h3>
    <button onClick={addDay}>Day 추가</button>
  </div>
);
}

```

CreateWord.js

```

import { useState } from "react";
import { useNavigate } from 'react-router-dom';
import useFetch from "../hooks/useFetch";

export default function CreateWord() {
  const days = useFetch("http://localhost:3010/days");
  const navigate = useNavigate();

  const [eng, setEng] = useState("");
  const [kor, setKor] = useState("");
  const [day, setDay] = useState("");

  function onSubmit(e) {

```

```

e.preventDefault();

fetch(`http://localhost:3010/words/`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    day,
    eng,
    kor,
    isDone: false,
  }),
}).then(res => {
  if (res.ok) {
    alert("생성이 완료 되었습니다");
    navigate(`/day/${day}`);
  }
});
}

return (
  <form onSubmit={onSubmit}>
    <div className="input_area">
      <label>Eng</label>
      <input
        type="text"
        placeholder="computer"
        value={eng}
        onChange={(e) => setEng(e.target.value)}
      />
    </div>
    <div className="input_area">
      <label>Kor</label>
      <input
        type="text"
        placeholder="컴퓨터"
        value={kor}
        onChange={(e) => setKor(e.target.value)}
      />
    </div>
    <div className="input_area">

```

```

<label>Day</label>
<select
  value={day}
  onChange={(e) => setDay(e.target.value)}
>
  <option value="">-- 선택하세요 --</option>
  {days.map(dayItem => (
    <option key={dayItem.id} value={dayItem.day}>
      {dayItem.day}
    </option>
  ))}
</select>
</div>
<button>저장</button>
</form>
);
}

```

---

> 22.

---



---

> 23.

---



---

> 24.

---

json 요청하기

<https://jsonplaceholder.typicode.com/users> 에 들어가 보면 test json 데이터를 확인할 수 있다.

<https://www.data.go.kr/> 국가에서 관리하는 데이터베이스

```
import React, { useState, useEffect } from 'react';

// DataFetcher 컴포넌트 정의
const DataFetcher = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // 데이터 fetch
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error('Error fetching data:', error));
  }, []); // 빈 배열을 의존성 배열로 전달하면, 컴포넌트가 처음 렌더링될 때만 실행됩니다.

  return (
    <div>
      {data ? (
        <div>
          {data.map(user => (
            <p key={user.id}>{user.name}</p>
          ))}
        </div>
      ) : (
        <p>Loading...</p>
      )}
    </div>
  );
};

// App 컴포넌트 정의
const App = () => {
  return (
    <div style={{ padding: '20px' }}>
      <h1>User List</h1>
      <DataFetcher />
    </div>
  );
};
```

```
        </div>
    );
};

export default App;
```

## 설명

### 1. **data** 배열의 사용:

- **data**는 여러 사용자 객체로 이루어진 배열입니다. 따라서 **data.name**이 아닌 **data.map(user => user.name)**을 사용해야 합니다.

### 2. **map** 메소드:

- **data.map(user => ...)**을 사용하여 배열의 각 사용자 객체에 대해 **<p>** 요소를 생성합니다.
- **key={user.id}**를 추가하여 각 **<p>** 요소에 고유한 키를 부여합니다. 이는 **React**가 각 요소를 효율적으로 추적하는 데 필요합니다.

### 3. 에러 처리:

- **.catch(error => console.error('Error fetching data:', error))**를 추가하여 데이터를 가져오는 동안 발생할 수 있는 오류를 콘솔에 출력합니다.

이 수정된 코드로 **data** 배열의 각 사용자 이름을 웹 페이지에 렌더링할 수 있으며, 데이터가 로드되는 동안 **"Loading..."** 메시지를 표시합니다.

## fetch 관련 자바 스크립트 예제

**fetch**는 웹 브라우저에서 HTTP 요청을 보내고 응답을 받기 위한 API입니다. **fetch**는 **JavaScript**에서 비동기적으로 데이터를 가져오는 데 사용되며, 주로 API 호출, 데이터 요청, 서버와의 상호작용 등에 사용됩니다.

### 기본 문법

```
fetch(url, options)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json(); // 또는 response.text() 등
  })
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('There has been a problem with your fetch operation:',
error);
  });
```

### 주요 개념

1. **URL**: 요청을 보낼 서버의 URL입니다.
2. 옵션 객체: HTTP 메서드(GET, POST, PUT, DELETE 등), 헤더, 본문 데이터 등을 설정할 수 있습니다.
3. 응답 객체: 서버에서 반환된 응답을 나타내는 객체입니다. **response.json()**은 JSON 형태로 데이터를 파싱합니다.

### 사용 예제

#### 1. GET 요청

서버로부터 데이터를 가져오는 기본적인 GET 요청 예제입니다.

```
fetch('https://api.example.com/data')
```



```
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
.then(data => {
  console.log('Data:', data);
})
.catch(error => {
  console.error('Fetch error:', error);
});
```

- 설명: `fetch`를 호출하여 서버에서 데이터를 요청합니다. 응답을 `JSON`으로 변환하고, 데이터를 콘솔에 출력합니다. 오류가 발생하면 `catch` 블록에서 처리합니다.

## 2. POST 요청

서버에 데이터를 전송하는 `POST` 요청 예제입니다.

```
fetch('https://api.example.com/data', {
  method: 'POST', // 요청 메서드
  headers: {
    'Content-Type': 'application/json' // 요청 헤더
  },
  body: JSON.stringify({ key: 'value' }) // 요청 본문
})
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
.then(data => {
  console.log('Response Data:', data);
});
```

```
  })  
  .catch(error => {  
    console.error('Fetch error:', error);  
  });
```

- 설명: `method`를 'POST'로 설정하고, `body`에 JSON 문자열을 포함시킵니다. 이 데이터를 서버에 전송하고, 응답을 처리합니다.

### 3. PUT 요청

서버에서 기존 데이터를 업데이트하는 PUT 요청 예제입니다.

javascript

복사편집

```
fetch('https://api.example.com/data/1', {  
  
  method: 'PUT', // 요청 메서드  
  
  headers: {  
  
    'Content-Type': 'application/json' // 요청 헤더  
  
  },  
  
  body: JSON.stringify({ key: 'newValue' }) // 요청 본문  
  
})  
  
  .then(response => {  
  
    if (!response.ok) {  
  
      throw new Error('Network response was not ok');  
  
    }  
  
    return response.json();  
  
  })
```

```
.then(data => {  
    console.log('Updated Data:', data);  
})  
  
.catch(error => {  
    console.error('Fetch error:', error);  
});
```

설명: **method**를 'PUT'으로 설정하고, **body**에 업데이트할 데이터를 JSON 문자열로 포함시켜 서버에 전송합니다. 서버의 기존 데이터를 업데이트하고, 응답을 처리합니다.

---

#### 4. DELETE 요청

서버에서 데이터를 삭제하는 DELETE 요청 예제입니다.

javascript

복사편집

```
fetch('https://api.example.com/data/1', {  
    method: 'DELETE', // 요청 메서드  
})  
  
.then(response => {  
    if (!response.ok) {  
        throw new Error('Network response was not ok');  
    }  
  
    console.log('Data deleted successfully');  
})
```

```
.catch(error => {  
  
    console.error('Fetch error:', error);  
  
});
```

설명: `method`를 'DELETE'로 설정하여 서버에서 특정 데이터를 삭제합니다. 삭제 성공 시 메시지를 출력하고, 오류가 발생하면 `catch` 블록에서 처리합니다.

### 3. Error Handling

`fetch` 호출 시 발생할 수 있는 오류를 처리하는 예제입니다.

javascript

코드 복사

```
fetch('https://api.example.com/data')  
    .then(response => {  
        if (!response.ok) {  
            throw new Error('Network response was not ok');  
        }  
        return response.json();  
    })  
    .then(data => {  
        console.log('Data:', data);  
    })  
    .catch(error => {  
        console.error('Fetch error:', error.message);  
    });
```

- 설명: `response.ok`를 사용하여 응답 상태가 성공적인지 확인하고, 오류가 발생하면 `catch` 블록에서 처리합니다.

비동기/대기 (async/await)

`fetch`와 `async/await`를 함께 사용하여 코드의 가독성을 높일 수 있습니다.

javascript

코드 복사

```
const fetchData = async () => {  
  try {  
    const response = await fetch('https://api.example.com/data');  
    if (!response.ok) {  
      throw new Error('Network response was not ok');  
    }  
    const data = await response.json();  
    console.log('Data:', data);  
  } catch (error) {  
    console.error('Fetch error:', error.message);  
  }  
};  
  
fetchData();
```

- 설명: `fetch`를 `await`하여 응답을 기다리고, `JSON`으로 변환 후 데이터를 처리합니다. 오류가 발생하면 `catch` 블록에서 처리합니다.

## `useReducer`

- 설명: `useState`보다 복잡한 상태 로직을 관리할 때 사용되는 훅입니다. 상태 업데이트 로직을 별도의 `reducer` 함수로 분리할 수 있습니다.

`useReducer`는 상태 관리 로직을 리듀서 함수로 분리하여 복잡한 상태 업데이트를 간단하고 예측 가능하게 처리할 수 있는 `React` 훅입니다.

기본 문법

javascript

복사편집

```
const [state, dispatch] = useReducer(reducer, initialState);
```

- `state`: 현재 상태 값
- `dispatch`: 상태를 변경할 액션을 전달하는 함수
- `reducer`: 상태 변경 로직을 담당하는 함수

- `initialState`: 초기 상태 값



다음과 같은 화면은 3개의 메소드로 `count`값을 조작하는 형태로 되어 있다.

이렇게 여러개의 데이터와 여러개의 메소드가 같은 데이터를 조작하는 용도로 사용되면 `userReducer`로 묶어서 표현할 수 있다.

```
import React, { useReducer } from 'react';

// 1. 리듀서 함수 정의
const reducer = (state, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    case 'RESET':
      return { count: 0 };
    default:
      return state;
  }
};

// 2. 초기 상태
const initialState = { count: 0 };
const Counter = () => {
  // 3. useReducer를 사용하여 상태와 디스패치 함수를 정의
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
```

```

    {/* 4. 변경된 데이터를 그때그때 보여줄 state변수 기술*/}
    <p>Count: {state.count}</p>
    {/* 5. 디스패치 함수를 사용하여 액션을 보냄 */}
    <button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>
    <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>
    <button onClick={() => dispatch({ type: 'RESET' })}>Reset</button>
  </div>
);
};

export default Counter;

```

`const [state, dispatch] = useReducer(reducer, initialState);` 에서  
`state`는 현재 상태를 나타내는 변수입니다.  
`dispatch`는 상태를 업데이트하기 위해 액션을 보내는 함수입니다

1. `useReducer`에서 사용할 로직들을 구현한다.
2. 값변경시 화면에 변경된 값을 바로바로 표현해줄 변수를 생성한다.
3. 로직과 변수로 `useReducer`메소드를 이용해서 `state`와`dispatch`를 얻는다.
4. 4. 변경된 데이터를 그때그때 보여줄 `state`변수 기술
5. 디스패치 함수를 사용하여 액션을 보냄



## 사용자 정의 훅 (Custom Hook)

사용자 정의 훅을 통해 공통된 로직을 재사용할 수 있습니다.

많이 사용하는 기능을 Hook으로 만들어서 필요할때 사용한다. 다음예제는 토글 기능을 가지고 있는 `useToggle` Hook를 만들어 보았다.

`useToggle`은 불리언 상태를 간단하게 토글할 수 있는 사용자 정의 훅입니다.

주요 포인트:

- 목적: 불리언 값을 `true`와 `false`로 쉽게 변경하는 기능을 제공합니다.
- 구성: 상태 값과 상태를 반전시키는 함수(토글 함수)를 반환합니다.
- 장점:
  - 코드 간결성: 상태를 간단하게 관리할 수 있음.
  - 재사용성: 여러 컴포넌트에서 반복 사용 가능.
  - 초기 값 설정: 초기 상태를 `true` 또는 `false`로 설정 가능.

### App Component

Off

Toggle1

Off

Toggle2

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

```
// 사용자 정의 훅: true/false 값을 토글
```

```
const useToggle = (initialValue = false) => {

  const [value, setValue] = useState(initialValue);

  const toggle = () => {

    setValue(prevValue => !prevValue);

  };

  return [value, toggle];

};

// Toggle 버튼을 포함한 컴포넌트

const ToggleComponent = () => {

  const [isToggled1, toggle1] = useToggle(); // 사용자 정의 훅 사용

  const [isToggled2, toggle2] = useToggle(); // 사용자 정의 훅 사용

  return (<>

    <div>

      <h1>{isToggled1 ? 'On' : 'Off'}</h1>

      <button onClick={toggle1}>Toggle1</button>

    </div>

    <div>

      <h1>{isToggled2 ? 'On' : 'Off'}</h1>

      <button onClick={toggle2}>Toggle2</button>

    </div>

  </>);
}
```

```
    </>);  
  
};  
  
// App 컴포넌트  
  
const App = () => {  
  
  return (  
  
    <div>  
  
      <h1>App Component</h1>  
  
      <ToggleComponent /> { /* ToggleComponent를 포함 */}  
  
    </div>  
  
  );  
  
};  
  
export default App;
```

---

## > 26.fetch 사용법

---

json 요청하기

<https://jsonplaceholder.typicode.com/users> 에 들어가 보면 test json 데이터를 확인할 수 있다.

```
import React, { useState, useEffect } from 'react';
```

```
// DataFetcher 컴포넌트 정의
const DataFetcher = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // 데이터 fetch
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error('Error fetching data:', error));
  }, []); // 빈 배열을 의존성 배열로 전달하면, 컴포넌트가 처음 렌더링될 때만 실행됩니다.
```

```
  return (
    <div>
      {data ? (
        <div>
          {data.map(user => (
            <p key={user.id}>{user.name}</p>
          ))}
        </div>
      ) : (
        <p>Loading...</p>
      )}
    </div>
  );
};
```

```
// App 컴포넌트 정의
const App = () => {
  return (
    <div style={{ padding: '20px' }}>
      <h1>User List</h1>
      <DataFetcher />
    </div>
  );
};
```

```
export default App;
```

## 설명

### 1. `data` 배열의 사용:

- `data`는 여러 사용자 객체로 이루어진 배열입니다. 따라서 `data.name`이 아닌 `data.map(user => user.name)`을 사용해야 합니다.

### 2. `map` 메소드:

- `data.map(user => ...)`을 사용하여 배열의 각 사용자 객체에 대해 `<p>` 요소를 생성합니다.
- `key={user.id}`를 추가하여 각 `<p>` 요소에 고유한 키를 부여합니다. 이는 `React`가 각 요소를 효율적으로 추적하는 데 필요합니다.

### 3. 에러 처리:

- `.catch(error => console.error('Error fetching data:', error))`를 추가하여 데이터를 가져오는 동안 발생할 수 있는 오류를 콘솔에 출력합니다.

이 수정된 코드로 `data` 배열의 각 사용자 이름을 웹 페이지에 렌더링할 수 있으며, 데이터가 로드되는 동안 "Loading..." 메시지를 표시합니다.

## fetch관련 자바 스크립트 예제

`fetch`는 웹 브라우저에서 HTTP 요청을 보내고 응답을 받기 위한 API입니다. `fetch`는 JavaScript에서 비동기적으로 데이터를 가져오는 데 사용되며, 주로 API 호출, 데이터 요청, 서버와의 상호작용 등에 사용됩니다.

## 기본 문법

```
fetch(url, options)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json(); // 또는 response.text() 등
  })
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('There has been a problem with your fetch operation:',
error);
  });
```

## 주요 개념

1. **URL**: 요청을 보낼 서버의 URL입니다.
2. **옵션 객체**: HTTP 메서드(GET, POST, PUT, DELETE 등), 헤더, 본문 데이터 등을 설정할 수 있습니다.
3. **응답 객체**: 서버에서 반환된 응답을 나타내는 객체입니다. `response.json()`은 JSON 형태로 데이터를 파싱합니다.

## 사용 예제

### 1. GET 요청

서버로부터 데이터를 가져오는 기본적인 GET 요청 예제입니다.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
  })
```

```
    return response.json();
  })
  .then(data => {
    console.log('Data:', data);
  })
  .catch(error => {
    console.error('Fetch error:', error);
  });
```

- 설명: `fetch`를 호출하여 서버에서 데이터를 요청합니다. 응답을 `JSON`으로 변환하고, 데이터를 콘솔에 출력합니다. 오류가 발생하면 `catch` 블록에서 처리합니다.

## 2. POST 요청

서버에 데이터를 전송하는 `POST` 요청 예제입니다.

```
fetch('https://api.example.com/data', {
  method: 'POST', // 요청 메서드
  headers: {
    'Content-Type': 'application/json' // 요청 헤더
  },
  body: JSON.stringify({ key: 'value' }) // 요청 본문
})
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Response Data:', data);
  })
  .catch(error => {
    console.error('Fetch error:', error);
  });
```

- 설명: `method`를 'POST'로 설정하고, `body`에 JSON 문자열을 포함시킵니다. 이 데이터를 서버에 전송하고, 응답을 처리합니다.

### 3. Error Handling

`fetch` 호출 시 발생할 수 있는 오류를 처리하는 예제입니다.

javascript

코드 복사

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Data:', data);
  })
  .catch(error => {
    console.error('Fetch error:', error.message);
  });
```

- 설명: `response.ok`를 사용하여 응답 상태가 성공적인지 확인하고, 오류가 발생하면 `catch` 블록에서 처리합니다.

### 비동기/대기 (`async/await`)

`fetch`와 `async/await`를 함께 사용하여 코드의 가독성을 높일 수 있습니다.

javascript

코드 복사

```
const fetchData = async () => {
```



```
try {
  const response = await fetch('https://api.example.com/data');
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  const data = await response.json();
  console.log('Data:', data);
} catch (error) {
  console.error('Fetch error:', error.message);
}
};

fetchData();
```

- 설명: `fetch`를 `await`하여 응답을 기다리고, `JSON`으로 변환 후 데이터를 처리합니다. 오류가 발생하면 `catch` 블록에서 처리합니다.

## 결론

`fetch`는 웹에서 데이터를 가져오거나 서버에 데이터를 보내는 데 유용한 **API**입니다. 비동기적으로 동작하며, `Promise` 기반의 처리를 지원하여 코드의 가독성을 높입니다. `async/await`와 함께 사용하면 더욱 깔끔한 코드를 작성할 수 있습니다.

```
async function func() {  
  
  console.log("1");  
  
  await new Promise(resolve => setTimeout(resolve, 1000));  
  
  console.log("2");  
  
}
```


```
console.log("시작");
```

```
func();
```

```
console.log("끝");
```

출력 순서:

시작    1        끝        (1초 후)    2

 흐름 설명

`func()`는 비동기 함수라 호출하면 **Promise**를 반환.

`console.log("1")`은 즉시 실행됨.

`await`를 만나면 → JS는 그 작업이 끝날 때까지 기다리고,

그 사이에 다음 코드 (`console.log("끝")`)로 넘어감.

1초 후, **Promise**가 완료되면 → `console.log("2")` 실행됨.

 요약

구문    의미    실행    흐름

**async** 이 함수는 **Promise**를 반환한다    비동기 함수 정의

`await Promise`가 끝날 때까지 기다린다 일시 중단 후 다른 코드 실행됨 (즉, 전체는 비동기 흐름)

! 그래서 결론

✅ `await`는 동기처럼 보이게 만드는 비동기 처리 방식입니다.

✅ `await` 줄에서 멈추긴 하지만, 그 함수는 비동기로 동작하며 다음 코드로 넘어갑니다.

`await`가 없으면 다음과 같이 실행됨

```
async function func() {  
  
  console.log("1");  
  
  // await new Promise(resolve => setTimeout(resolve, 1000));  
  
  console.log("2");  
  
}
```

```
console.log("시작");
```

```
func();
```

```
console.log("끝");
```

✅ 실행 결과

복사

편집

시작

1

2

끝

✅ 왜 이렇게 나오는가?

`console.log("시작")` → 즉시 출력.

`func()` 호출:

`async` 함수지만, 내부에 `await`가 없기 때문에 즉시 실행되고,

`console.log("1")`, `console.log("2")` 모두 동기적으로 실행됨.

그 다음 `console.log("끝")` 출력.

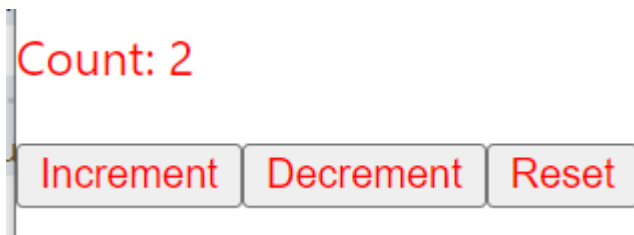
---

## > 29. `useReducer`

---

- 설명: `useState`보다 복잡한 상태 로직을 관리할 때 사용되는 hook입니다. 상태 업데이트 로직을 별도의 `reducer` 함수로 분리할 수 있습니다.

`useReducer`는 상태 관리 로직을 리듀서 함수로 분리하여 복잡한 상태 업데이트를 간단하고 예측 가능하게 처리할 수 있는 `React` hook입니다.



다음과 같은 화면은 3개의 메소드로 `count`값을 조작하는 형태로 되어 있다.

이렇게 여러개의 데이터와 여러개의 메소드가 같은 데이터를 조작하는 용도로 사용되면 `useReducer`로 묶어서 표현할 수 있다.

```
import React, { useReducer } from 'react';
```

```
// 1. 리듀서 함수 정의
```

```
const reducer = (state, action) => {  
  switch (action.type) {  
    case 'INCREMENT':  
      return { count: state.count + 1 };  
    case 'DECREMENT':  
      return { count: state.count - 1 };  
    case 'RESET':  
      return { count: 0 };  
    default:  
      return state;  
  }  
}
```

```

    }
  };
  // 2. 초기 상태
  const initialState = { count: 0 };
  const Counter = () => {
    // 3. useReducer를 사용하여 상태와 디스패치 함수를 정의
    const [state, dispatch] = useReducer(reducer, initialState);

    return (
      <div>
        {/* 4. 변경된 데이터를 그때그때 보여줄 state변수 기술*/}
        <p>Count: {state.count}</p>
        {/* 5. 디스패치 함수를 사용하여 액션을 보냄 */}
        <button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>
        <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>
        <button onClick={() => dispatch({ type: 'RESET' })}>Reset</button>
      </div>
    );
  };
};

export default Counter;

```

`const [state, dispatch] = useReducer(reducer, initialState);` 에서 `state`는 현재 상태를 나타내는 변수입니다.

`dispatch`는 상태를 업데이트하기 위해 액션을 보내는 함수입니다

1. `useReducer`에서 사용할 로직들을 구현한다.
2. 값변경시 화면에 변경된 값을 바로바로 표현해줄 변수를 생성한다.
3. 로직과 변수로 `useReducer`메소드를 이용해서 `state`와`dispatch`를 얻는다.
4. 4. 변경된 데이터를 그때그때 보여줄 `state`변수 기술
5. 디스패치 함수를 사용하여 액션을 보냄

## 사용자 정의 훅 (Custom Hook)

사용자가 훅을 만들 수 있다.

사용자 정의 훅을 통해 공통된 로직을 재사용할 수 있습니다.

많이 사용하는 기능을 Hook으로 만들어서 필요할 때 사용한다. 다음 예제는 토글 기능을 가지고 있는 `useToggle` Hook를 만들어 보았다.

`useToggle`은 불리언 상태를 간단하게 토글할 수 있는 사용자 정의 훅입니다.

주요 포인트:

- 목적: 불리언 값을 `true`와 `false`로 쉽게 변경하는 기능을 제공합니다.
- 구성: 상태 값과 상태를 반전시키는 함수(토글 함수)를 반환합니다.
- 장점:
  - 코드 간결성: 상태를 간단하게 관리할 수 있음.
  - 재사용성: 여러 컴포넌트에서 반복 사용 가능.
  - 초기 값 설정: 초기 상태를 `true` 또는 `false`로 설정 가능.

### App Component

Off

Toggle1

Off

Toggle2

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

```
// 사용자 정의 훅: true/false 값을 토글
```

```
const useToggle = (initialValue = false) => {

  const [value, setValue] = useState(initialValue);

  const toggle = () => {

    setValue(prevValue => !prevValue);

  };

  return [value, toggle];

};

// Toggle 버튼을 포함한 컴포넌트

const ToggleComponent = () => {

  const [isToggled1, toggle1] = useToggle(); // 사용자 정의 훅 사용

  const [isToggled2, toggle2] = useToggle(); // 사용자 정의 훅 사용

  return (<>

    <div>

      <h1>{isToggled1 ? 'On' : 'Off'}</h1>

      <button onClick={toggle1}>Toggle1</button>

    </div>

    <div>

      <h1>{isToggled2 ? 'On' : 'Off'}</h1>

      <button onClick={toggle2}>Toggle2</button>

    </div>

  </>);

};
```



```

    </>);

};

// App 컴포넌트

const App = () => {

  return (

    <div>

      <h1>App Component</h1>

      <ToggleComponent /> { /* ToggleComponent를 포함 */}

    </div>

  );

};

export default App;

```

---

> 30.

---

```
json-server --watch ./src/db/data.json --port 3010
```

<http://localhost:3010/days> 엔드포인트에서 CRUD 작업을 수행하는 방법을 설명해 드리겠습니다.

#### 1. 모든 Day 조회 (Read - GET)

http

복사편집

GET <http://localhost:3010/days>

📌 모든 **day** 데이터를 가져옵니다.

응답 예시:

json

복사편집

```
[
  {
    "id": "1",
    "day": 1
  },
  {
    "id": "2",
    "day": 2
  },
  {
    "id": "3",
    "day": 3
  },
  {
    "id": "4",
    "day": 4
  },
  {
    "id": "5",
    "day": 5
  }
]
```

]

---

## 2. 특정 Day 조회 (Read - GET)

http

복사편집

GET <http://localhost:3010/days/1>

 ID가 1인 Day를 조회합니다.

응답 예시:

json

복사편집

```
{  
  "id": "1",  
  "day": 1  
}
```

---


## 3. Day 추가 (Create - POST)

http

복사편집

POST <http://localhost:3010/days>

Content-Type: application/json

 새로운 Day 데이터를 추가합니다.

요청 바디 예시:

json

복사편집

```
{  
  "id": "6",  
  "day": 6  
}
```

응답 예시:

json

복사편집

```
{  
  "id": "6",  
  "day": 6  
}
```

---

#### 4. Day 수정 (Update - PUT/PATCH)

http

복사편집

PATCH <http://localhost:3010/days/2>

Content-Type: application/json

 ID가 2인 Day 데이터를 수정합니다.

요청 바디 예시:

json

복사편집

```
{  
  "day": 20  
}
```

응답 예시:

json

복사편집

```
{  
  "id": "2",  
  "day": 20  
}
```

---

## 5. Day 삭제 (Delete)

http

복사편집

DELETE http://localhost:3010/days/3

 ID가 3인 Day 데이터를 삭제합니다.

응답 예시:

json

복사편집

```
{  
  "message": "Day 3 deleted"  
}
```

---

## 추가 설명

- **GET** 요청은 데이터를 조회하는 데 사용됩니다.
- **POST** 요청은 새 데이터를 추가하는 데 사용됩니다.
- **PATCH** 또는 **PUT** 요청은 기존 데이터를 수정하는 데 사용됩니다.
- **DELETE** 요청은 데이터를 삭제하는 데 사용됩니다.

**http://localhost:3010/words** 엔드포인트에서 **CRUD** 작업을 수행하는 방법을 설명해 드리겠습니다.

---

### 1. 모든 단어 조회 (Read - GET)

http

복사편집

**GET http://localhost:3010/words**

 모든 단어 데이터를 가져옵니다.

응답 예시:

json

복사편집

```
[
  {
    "id": "1",
    "day": 1,
    "eng": "book",
    "kor": "책",
    "isDone": false
  },
  {
    "id": "2",
    "day": 1,
    "eng": "apple",
    "kor": "사과",
    "isDone": false
  },
  {
    "id": "4",
    "day": 2,
    "eng": "pen",
    "kor": "펜",
    "isDone": true
  }
]
```


---

## 2. 특정 단어 조회 (Read - GET)

http

복사편집

GET <http://localhost:3010/words/1>

 ID가 1인 단어를 조회합니다.

응답 예시:

json

복사편집

```
{  
  "id": "1",  
  "day": 1,  
  "eng": "book",  
  "kor": "책",  
  "isDone": false  
}
```

---


## 3. 단어 추가 (Create - POST)

http

복사편집

POST <http://localhost:3010/words>

Content-Type: application/json

 새로운 단어를 추가합니다.



요청 바디 예시:

json

복사편집

```
{  
  "id": "7",  
  "day": 1,  
  "eng": "table",  
  "kor": "테이블",  
  "isDone": false  
}
```

응답 예시:

json

복사편집

```
{  
  "id": "7",  
  "day": 1,  
  "eng": "table",  
  "kor": "테이블",  
  "isDone": false  
}
```

---


#### 4. 단어 수정 (Update - PUT/PATCH)

http

복사편집

PATCH <http://localhost:3010/words/2>

Content-Type: application/json

 ID가 2인 단어를 수정합니다.

요청 바디 예시 (예: **isDone** 상태 변경)

json

복사편집

```
{  
  "isDone": true  
}
```

응답 예시:

json

복사편집

```
{  
  "id": "2",  
  "day": 1,  
  "eng": "apple",  
  "kor": "사과",  
  "isDone": true  
}
```

---

## 5. 단어 삭제 (Delete)

http

복사편집

DELETE http://localhost:3010/words/6

📌 ID가 6인 단어를 삭제합니다.

응답 예시:

json

복사편집

```
{  
  "message": "Word 6 deleted"  
}
```

---

📌 추가 설명

- GET 요청: 단어 목록 및 특정 단어 조회
- POST 요청: 새로운 단어 추가
- PATCH 또는 PUT 요청: 단어 정보 수정
- DELETE 요청: 단어 삭제

이제 Thunder Client에서 위 요청을 실행하여 CRUD 작업을 수행해 보세요! 🚀

## 1. 특정 요일의 완료된 단어만 조회 (GET)

http

복사편집

GET http://localhost:3010/words?day=3&isDone=true

📌 day=3인 단어 중에서 완료된 (isDone=true) 단어만 필터링해서 조회합니다.

응답 예시:

json

복사편집

```
[
  {
    "id": "6",
    "day": 3,
    "eng": "pencil",
    "kor": "연필",
    "isDone": true
  }
]
```

---

2. 모든 단어의 isDone 상태를 false로 초기화 (PATCH)

http

복사편집

PATCH http://localhost:3010/words/reset

Content-Type: application/json

📌 모든 단어의 isDone을 false로 초기화하는 엔드포인트를 만든다고 가정합니다.

요청 바디 (선택 사항):

json

복사편집

```
{  
  "isDone": false  
}
```

응답 예시:

json

복사편집

```
{  
  "message": "All words have been reset"  
}
```

---

### 3. 특정 요일의 단어 개수 조회 (GET)

http

복사편집

GET <http://localhost:3010/words/count?day=2>

 **day=2**인 단어가 몇 개인지 조회합니다.

응답 예시:

json

복사편집

```
{  
  "day": 2,  
  "count": 2  
}
```

---

#### 4. 랜덤으로 단어 하나 추천받기 (GET)

http

복사편집

GET <http://localhost:3010/words/random>

 랜덤한 단어 하나를 추천받습니다.

응답 예시:

json

복사편집

```
{  
  "id": "5",  
  "day": 3,  
  "eng": "school",  
  "kor": "학교",  
  "isDone": false  
}
```

---


#### 5. 특정 요일의 단어를 한 번에 여러 개 추가 (POST)

http

복사편집

POST <http://localhost:3010/words/bulk>

Content-Type: application/json

 특정 요일(**day=4**)에 여러 단어를 한 번에 추가합니다.

요청 바디 예시:

json

복사편집

```
{
  "day": 4,
  "words": [
    { "eng": "notebook", "kor": "노트북", "isDone": false },
    { "eng": "eraser", "kor": "지우개", "isDone": false },
    { "eng": "bag", "kor": "가방", "isDone": false }
  ]
}
```

응답 예시:

json

복사편집

```
{
  "message": "3 words added for day 4"
}
```

---

## 6. 완료된 단어들을 한 번에 삭제 (DELETE)

http

복사편집

DELETE <http://localhost:3010/words/done>

📌 완료된 (**isDone=true**) 단어들을 한 번에 삭제합니다.

응답 예시:

json

복사편집

```
{  
  "message": "Deleted 2 completed words"  
}
```

---

📌 추가 설명

- 쿼리 파라미터 활용 → **?day=3&isDone=true** 같은 필터링
- 랜덤 요소 추가 → 단어를 랜덤으로 추천
- 배치 처리 → 여러 개의 단어를 한 번에 추가 또는 삭제
- 특정 조건에 따라 상태 변경 → 모든 단어의 **isDone** 초기화

이런 작업들을 **Thunder Client**에서 직접 실행해 보면서 확장하면 더 재미있을 거예요! 🚀



---

## > 30. 피자 애니메이션 효과

---

피자

<https://www.youtube.com/watch?v=CjGEuLgt4nw>

<https://www.youtube.com/watch?v=k3Vfj-e1Ma4&t=5629s>

피자관련

<https://www.youtube.com/watch?v=QwarZBtFoFA&list=WL&index=1&t=3169s>

코드 주소

<https://github.com/machadop1407/react-website-tutorial>

노드 버전 실행 방법

과거 버전 돌리는 방법

`npm install`

```
set NODE_OPTIONS=--openssl-legacy-provider
```

```
npm install react-scripts@latest
```

```
npm start
```

새로 설치하는 방법

```
npm install react-router-dom
```

```
npm install @mui/icons-material@7.0.0
```

```
npm install @emotion/react@^11.11.1 @emotion/styled@^11.11.0
```

아이콘 확인사이트

-- <https://mui.com/material-ui/material-icons/>

-- 애니메이션 효과

```
npm install framer-motion@11
```

12는 아직 불안정하여 11를 설치해야 한다.

설치 한것을 삭제하려면 `npm uninstall` 설치된 모듈이름 하면된다.

✨ 주요 기능 요약

- `motion` 컴포넌트로 HTML 요소를 애니메이션 가능하게 만들
- 애니메이션 상태 전환 (`animate`, `initial`, `exit`)
- 드래그 지원 (`drag`)
- 제스처 애니메이션 (`whileHover`, `whileTap`)

- AnimatePresence로 컴포넌트 진입/퇴장 애니메이션 가능

## ✓ 기본 사용법

### 1. 요소에 애니메이션 적용

```
import React from "react";
```

```
import { motion } from "framer-motion";
```

```
function MyComponent() {  
  
  return (  
  
    <motion.div  
  
      initial={{ opacity: 0 }}  
  
      animate={{ opacity: 1 }}  
  
      transition={{ duration: 1 }}  
  
    >  
  
      안녕하세요!  
  
    </motion.div>  
  
  );  
  
}
```

```
function App() {  
  
  return (  
  
    <div>  
  
      <MyComponent />  
  
    </div>  
  
  );  
  
}
```

```
    </div>

  );
}

export default App;
```

- **initial**: 시작 상태
  - **animate**: 애니메이션 도착 상태
  - **transition**: 시간 등 애니메이션 설정
- 

## 2. Hover 시 애니메이션

```
import React from "react";

import { motion } from "framer-motion";

function App() {

  return (

    <div>

      <motion.button

        whileHover={{ scale: 1.2 }} // 마우스 오버 시 크기 확대

        whileTap={{ scale: 0.9 }} // 클릭 시 크기 축소

        style={{
```

```
padding: "10px 20px",

fontSize: "16px",

cursor: "pointer",

border: "none",

backgroundColor: "#007BFF",

color: "white",

borderRadius: "5px",

}}

>

클릭해보세요

</motion.button>

</div>

);

}
```

```
export default App;
```

---

### 3. 드래그 가능 요소

```
import React from "react";

import { motion } from "framer-motion";

function App() {

  return (
```

```

    <motion.div
      drag

      dragConstraints={{ left: 0, right: 300, top: 0, bottom: 200 }}

      style={{ width: 100, height: 100, background: 'skyblue' }}
    >

      드래그 가능

    </motion.div>

  );
}

export default App;

```

**drag:** 이 속성은 요소가 드래그 가능하도록 설정합니다.

**dragConstraints:** 드래그할 수 있는 영역의 범위를 설정합니다. 여기서는 **left: 0, right: 300, top: 0, bottom: 200**으로 설정되어 있어, 드래그 가능한 영역이 좌측 0부터 우측 300까지, 상단 0부터 하단 200까지로 제한됩니다.

**style:** **motion.div** 요소의 크기와 배경 색상, 텍스트 스타일을 지정하여 보기 좋게 꾸몄습니다.

---

#### 4. 컴포넌트가 사라질 때 애니메이션

```

import React, { useState } from 'react';

import { AnimatePresence, motion } from 'framer-motion';

```

```

function ToggleBox() {

  const [show, setShow] = useState(true);

  return (

    <>

      <button onClick={() => setShow(!show)}>Toggle</button>

      <AnimatePresence>

        {show && (

          <motion.div

            initial={{ opacity: 0, y: -50 }}

            animate={{ opacity: 1, y: 0 }}

            exit={{ opacity: 0, y: 50 }}

            transition={{ duration: 0.5 }}

          >

            등장 & 퇴장 애니메이션

          </motion.div>

        )}}

      </AnimatePresence>

    </>

  );

}

```

```

function App() {

  return (

```

```

    <div>

      <h1>프레임 애니메이션 예시</h1>

      <ToggleBox />

    </div>

  );
}

```

```
export default App;
```

---

#### 추가 학습 자료

- 공식 문서: <https://www.framer.com/motion/>
- 애니메이션 종류와 예제: **variants**, **keyframes**, **layout animations** 등도 있음

화면 갱신 확인 하기 피자 예제.

#### APP.js

```

import { BrowserRouter , Routes, Route, useLocation } from "react-router-dom";

import { AnimatePresence } from "framer-motion";

import "./App.css";

import Navbar from "./components/Navbar";

```



```
import Footer from "../components/Footer";

import Home from "../pages/Home";

import Menu from "../pages/Menu";

import About from "../pages/About";

import Contact from "../pages/Contact";


function AnimatedRoutes() {

  const location = useLocation(); // 현재 경로 감지


  return (

    <AnimatePresence mode="wait">

      <Routes location={location} key={location.pathname}>

        <Route path="/" element={<Home />} />

        <Route path="/menu" element={<Menu />} />

        <Route path="/about" element={<About />} />

        <Route path="/contact" element={<Contact />} />

      </Routes>

    </AnimatePresence>

  );
}
```

```
function App() {

  return (

    <BrowserRouter >
```

```

    <Navbar />

    <AnimatedRoutes />

    <Footer />

  </BrowserRouter>

);

}

export default App;

//home.js

import React from "react";

import { Link } from "react-router-dom";

import BannerImage from "../assets/pizza.jpeg";

import "../styles/Home.css";

import { motion } from "framer-motion";

function Home() {

  return (

    <motion.div

      initial={{ opacity: 0, x: -50 }} // 초기 상태

      animate={{ opacity: 1, x: 0 }} // 애니메이션 적용

      exit={{ opacity: 0, x: 50 }} // 페이지 나갈 때

      transition={{ duration: 0.5 }} // 지속 시간

    >

      <div className="home" style={{ backgroundImage: `url(${BannerImage})` }}>

        <div className="headerContainer">

```

```
<h1> Pedro's Pizzeria </h1>
```

```
<p> PIZZA TO FIT ANY TASTE</p>
```

```
<Link to="/menu">
```

```
  <button> ORDER NOW </button>
```

```
</Link>
```

```
</div>
```

```
</div>
```

```
</motion.div>
```

```
);
```

```
}
```

```
export default Home;
```

**AnimatedRoutes**는 React Router v6와 Framer Motion을 함께 사용하는 페이지 전환 애니메이션 구조입니다.

아래에서 구조, 용도, 그리고 사용법을 쉽게 설명드릴게요.

---

## 전체 개념

```
function AnimatedRoutes() {
```

```

const location = useLocation(); // 현재 경로 정보 가져오기

return (
  <AnimatePresence mode="wait">
    <Routes location={location} key={location.pathname}>
      {/* 경로에 따른 컴포넌트 렌더링 */}
      <Route path="/" element={<Home />} />
      <Route path="/menu" element={<Menu />} />
      <Route path="/about" element={<About />} />
      <Route path="/contact" element={<Contact />} />
    </Routes>
  </AnimatePresence>
);
}

```



## 이 코드는 무엇을 하나요?

코드 부분	설명
<code>useLocation()</code>	현재 URL 경로 정보를 가져옵니다. 예: <code>/</code> , <code>/menu</code> 등
<code>&lt;AnimatePresence mode="wait"&gt;</code>	이전 컴포넌트가 <code>exit</code> 애니메이션을 끝낼 때까지 기다립니다.
<code>location={location}</code>	경로에 따라 <code>Routes</code> 를 렌더링하게 만듭니다.
<code>key={location.pathn</code>	경로가 바뀔 때마다 컴포넌트를 새로 마운트하도록 합니다.

`ame}` 애니메이션 처리를 위해 필수입니다.

`<Route path=... />` 실제 각 경로별 페이지를 정의합니다.

---

## 사용 조건

`framer-motion` 라이브러리 설치

`bash`

복사편집

```
npm install framer-motion@12.6.2
```

- 1.
  2. React Router v6 사용 중이어야 합니다. (`react-router-dom`)
  3. 각 페이지 컴포넌트 (`Home`, `Menu` 등)는 `motion`을 사용해 애니메이션 처리해야 합니다.
- 

## 예시: **Home** 컴포넌트

`jsx`

복사편집

```
// Home.jsx

import { motion } from 'framer-motion';

function Home() {

  return (

    <motion.div

      initial={{ opacity: 0, x: 100 }}


```

```
        animate={{ opacity: 1, x: 0 }}
        exit={{ opacity: 0, x: -100 }}
        transition={{ duration: 0.5 }}
      >
        <h1>홈 페이지</h1>
      </motion.div>
    );
  }
}
```

```
export default Home;
```

위처럼 모든 페이지 컴포넌트에 `motion.div`를 감싸고 `initial`, `animate`, `exit` 속성을 정의해야 페이지 전환 애니메이션이 동작합니다.

---

## 전체 구성 예시

jsx

복사편집

```
// App.jsx

import { BrowserRouter as Router } from 'react-router-dom';

import AnimatedRoutes from './AnimatedRoutes';

function App() {
  return (
    <Router>
```

```
        <AnimatedRoutes />

    </Router>

);

}
```

---

## ✨ 결과적으로

- 페이지 전환 시 자연스럽게 이전 페이지가 사라지고, 새 페이지가 애니메이션과 함께 등장합니다.
- `AnimatePresence` + `Routes` 조합은 전환 애니메이션 처리에 필수적인 공식적인 구조입니다.



---

>4/15일

---

다음은 다운로드 받아 설치한다.

<https://github.com/iamshaunjp/Supabase-Tutorial-for-Beginners/tree/lesson-5>

```
npm install
```

```
npm start
```

<https://supabase.com/>

start your project 버튼을 누르고 새로운 프로젝트를 만든다.

### Create a new project

Your project will have its own dedicated instance and full Postgres database.  
An API will be set up so you can easily interact with your new database.

Organization

grapefoxman's Org

Project name

Supa Smoothies

Database Password

.....

Copy

This password is strong. [Generate a password](#)

Region

Northeast Asia (Seoul)

Select the region closest to your users for the best performance.

SECURITY OPTIONS >

ADVANCED CONFIGURATION >

Cancel

You can rename your project later

Create new project

다음 화면으로 이동해서 주소와 키를 얻는다.

Settings

PROJECT SETTINGS

General

Compute and Disk

Infrastructure

Integrations

Add Ons

Vault Alpha

CONFIGURATION

Database

**Data API**

Authentication

Storage

Edge Functions

Log Drains

BILLING

Subscription

Usage

### API Settings

Project URL

URL

https://ftrtpjqdzweFlzdjgnrl.s

RESTful endpoint for querying and managi

Project API Keys

Your API is secured behind an API gateway which requires an API Key for every request.  
You can use the keys below in the Supabase client libraries.

anon public

eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp1

This key is safe to use in a browser if you h  
configured policies.

service\_role secret

\*\*\*\* \*  
This key has the ability to bypass Row Leve  
new JWT secret immediately.

<https://ftrtpjqdzweflzdjgnrl.supabase.co>

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImZ0cnRwanFkendlZmx6ZGpnbmJsIiwicm9sZSI6ImFub24iLCJpYXQiOiE3NDM4NTI3ODksImV4cCI6MjA1OTQyODc4OX0.gM8MQw\_KZqEYubYz-Fs2CgIKFVIRvVVrF2R06QApbcE


.env 환경변수에 복사해 넣는다. 해당 파일은 root에 있다.















```
npm install react-router-dom
```

슈파베이스 모듈을 인서트 한다.

```
npm install @supabase/supabase-js
```

supabase.com/dashboard/project/rtrpjquzwenzajgm/api

 grapefoxman's Org Free / Supa Smoothies Enable branching



API Docs

GETTING STARTED

Introduction

AuthenticationUser Management

TABLES AND VIEWS

Introduction

STORED PROCEDURES

Introduction

GRAPHQL

GraphiQL

Connect To Your Project

All projects have a RESTful endpoint. These can be obtained from the [API](#).

You can initialize a new Supabase client to use the rest of the Supabase functionality in your ecosystem.

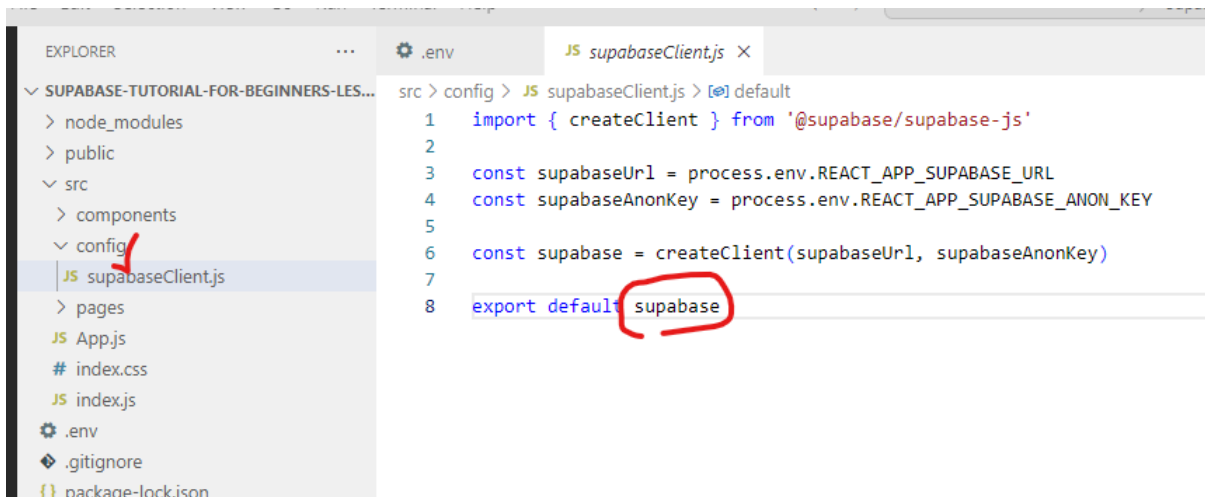
config file만들기

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = process.env.REACT_APP_SUPABASE_URL
const supabaseAnonKey = process.env.REACT_APP_SUPABASE_ANON_KEY

const supabase = createClient(supabaseUrl, supabaseAnonKey)

export default supabase
```



테이블 만들기

테이블 탭에 가서 다음과 같이 테이블을 만들자.

### Create a new table under public

Name

smoothies

Description

Optional

☐ Enable Row Level Security (RLS) Recommended

Restrict access to your table by enabling RLS and writing Postgres policies.



You are allowing anonymous access to your table

The table smoothies will be publicly writable and readable

[Documentation](#)

☐ Enable Realtime

Broadcast changes on this table to authorized subscribers

### Columns

[About data types](#)

[Import data from CSV](#)

Name ?	Type	Default Value ?	Primary
<input type="checkbox"/> id	# int8	NULL	<input checked="" type="checkbox"/>
<input type="checkbox"/> created_at	timestamp	now()	<input type="checkbox"/>
<input type="checkbox"/> title	T text	NULL	<input type="checkbox"/>
<input type="checkbox"/> method	T text	NULL	<input type="checkbox"/>
<input type="checkbox"/> rating	# int8	NULL	<input type="checkbox"/>

Cancel

Save

데이터를 넣어 보자.



## Table Editor

schema public

+ New table

Search tables...

smoothies

Filter

Sort

Insert

<input type="checkbox"/>	id int8	created_at timestampz	title text
--------------------------	---------	-----------------------	------------