230621_실습

React

Fetch API

Fetch API 란?

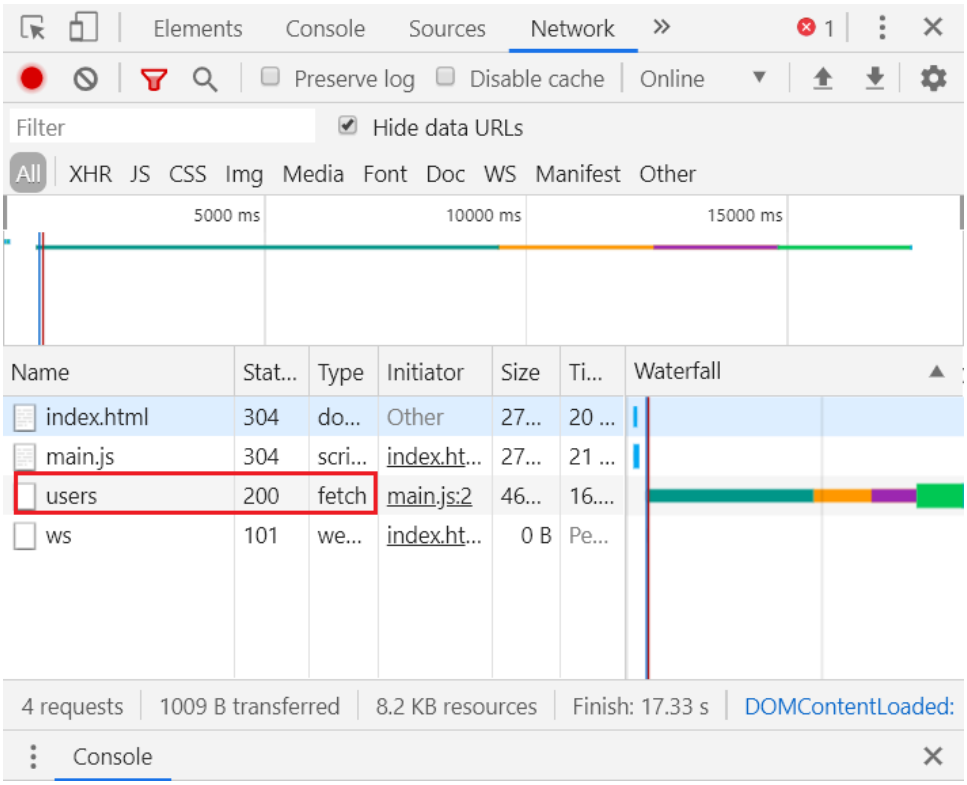
- HTTP 파이프라인을 구성하는 요청과 응답 등의 요소를 JavaScript에서 접근하고 조작할 수 있는 인터페이스를 제공한다.
- 네트워크의 리소스를 쉽게 비동기적으로 가져올 수도 있다.
- Promise 기반으로 구성되어 있다.
- JavaScript 기본 내장 기능이다.

Fetch 문법

```
fetch("https://jsonplaceholder.typicode.com/posts", option)
   .then(res => res.text())
   .then(text => console.log(text));
```

1. fetch 에는 기본적으로 첫 번째 인자에 요청할 url이 들어간다.
2. 기본적으로 http 메소드 중 GET으로 동작한다.
3. fetch를 통해 ajax를 호출 시 해당 주소에 요청을 보낸 다음, 응답 객체(Promise object Response)를 받는다.
4. 그러면 첫 번째 then에서 그 응답을 받게되고, res.text() 메서드로 파싱한 text값을 리턴한다.
5. 그러면 그 다음 then에서 리턴받은 text 값을 받고, 원하는 처리를 할 수 있게 된다.

- 개발자 도구의 네트워크 탭에 가보면 fetch를 통해 얻어온 데이터를 볼수 있다.



fetch의 response 속성

- response.status – HTTP 상태 코드(예: 200)
- response.ok – HTTP 상태 코드가 200과 299 사이일 경우 true
- response.body — 내용
- response.text() – 응답을 읽고 텍스트를 반환한다,
- response.json() – 응답을 JSON 형태로 파싱한다,
- response.formData() – 응답을 FormData 객체 형태로 반환한다.
- response.blob() – 응답을 Blob(타입이 있는 바이너리 데이터) 형태로 반환한다.

- response.arrayBuffer() – 응답을 ArrayBuffer(바이너리 데이터를 로우 레벨 형식으로 표현한 것) 형태로 반환한다.

fetch - GET Method

- **GET : 존재하는 자원을 요청**

  → 단순히 원격 API에 있는 데이터를 가져올 때 쓰임

  → fetch함수는 **디폴트로 GET 방식으로 작동**하고, 옵션 인자가 필요 없음.

  → 응답(response) 객체는 json() 메서드를 제공하고, 이 메서드를 호출하면 응답(response) 객체로부터 JSON 형태의 데이터를 자바스크립트 객체로 변환하여 얻을 수 있음.

```
JAVASCRIPT
fetch("https://jsonplaceholder.typicode.com/posts/1") // posts의 id 1인 엘리먼트를 가져옴
  .then((response) => response.json())
  .then((data) => console.log(data))
```

```
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderi
t",
  "body": "quia et suscipit↵suscipit recusandae consequuntur …strum rerum est autem su
nt rem eveniet architecto"
}
```

fetch - POST Method

- **POST : 새로운 자원 생성 요청**

  → 폼 등을 사용해서 데이터를 만들어 낼 때, 보내는 데이터의 양이 많거나, 비밀번호 등 개인정보를 보낼 때 POST 메서드 사용

  → 새로운 포스트 생성 위해서는 **method 옵션을 POST로 지정**해주고, **headers 옵션으로 JSON 포맷 사용한다고 알려줘야 함**. **body 옵션에는 요청 데이터를 JSON 포맷**으로 넣어줌.

  - method – HTTP 메서드
  - headers – 요청 헤드가 담긴 객체(제약 사항이 있음)
  - body – 보내려는 데이터(요청 본문)로 string이나 FormData, BufferSource, Blob, UrlSearchParams 객체 형태

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "POST", // POST
  headers: { // 헤더 조작
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ // 자바스크립트 객체를 json화 한다.
    title: "Test",
    body: "I am testing!",
    userId: 1,
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

fetch - PUT Method (전체수정)

- **PUT : 존재하는 자원 변경 요청**

  → API에서 관리하는 데이터의 **수정**을 위해 PUT 메서드 사용함.

  → method 옵션만 PUT으로 설정한다는 점 빼놓고는 POST 방식과 비슷

  → **아예 전체를 body의 데이터로 교체해버림.**

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    title: "Test" // 아예 title 엘리먼트로 전체 데이터를 바꿈. 마치 innerHTML같이.
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

fetch - PATCH Method (부분수정)

- **PATCH : 존재하는 자원 일부 변경 요청**

  → body의 데이터와 알맞는 **일부**만을 교체함 .

```
fetch("https://jsonplaceholder.typicode.com/posts/1", { // posts의 id 1인 엘리먼트를 수정
  method: "PATCH",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    title: "Test" // title만 바꿈. 나머지 요소는 건들지 않음.
  }),
```

```
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

fetch - DELETE Method

DELETE : 존재하는 자원 삭제 요청

→ 보낼 데이터가 없기 때문에 headers, body 옵션이 필요없음

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "DELETE",
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

Fetch - async / await 문법

```
fetch("https://jsonplaceholder.typicode.com/posts", option)
.then(res => res.text())
.then(text => console.log(text));
```

```
(async () => {
  let res = await fetch("https://jsonplaceholder.typicode.com/posts", option);
  let text = await res.text();
  console.log(text);
})() //await은 async함수내에서만 쓸수 있으니, 익명 async 바로 실행함수를 통해 활용합니다.
```

Axios

axios란?

- ajax 등의 웹 통신 기능을 제공하는 라이브러리 중 하나다.
- 브라우저, Node.js를 위한 Promise API를 활용하는 HTTP 비동기 통신 라이브러리이다.
- 백엔드랑 프론트엔드랑 통신을 쉽게하기 위해 Ajax와 더불어 사용한다.
- jquery와 비교하면, 타입스크립트도 사용이 가능하고, 요청 취소도 가능하며, 통신 기능만을 전담하므로 가볍다
- ES6 버전의 자바스크립트 문법을 사용하므로, 낮은 버전의 브라우저에서는 구동하지 않을 수도 있다(바벨, 웹팩 등으로 트랜스파일을 가하면 해결 가능)

axios 특징

- 운영 환경에 따라 브라우저의 XMLHttpRequest 객체 또는 Node.js의 http api 사용
- Promise(ES6) API 사용
- 요청과 응답 데이터의 변형
- HTTP 요청 취소
- HTTP 요청과 응답을 JSON 형태로 자동 변경

axios vs fetch

axios	fetch
써드파티 라이브러리로 설치가 필요	현대 브라우저에 빌트인이라 설치 필요 없음
XSRF 보호를 해준다.	별도 보호 없음
data 속성을 사용	body 속성을 사용
data는 object를 포함한다	body는 문자열화 되어있다
status가 200이고 statusText가 ‘OK’이면 성공이다	응답객체가 ok 속성을 포함하면 성공이다
자동으로 JSON데이터 형식으로 변환된다	.json()메서드를 사용해야 한다.
요청을 취소할 수 있고 타임아웃을 걸 수 있다.	해당 기능 존재 하지않음
HTTP 요청을 가로챌수 있음	기본적으로 제공하지 않음
download진행에 대해 기본적인 지원을 함	지원하지 않음
좀더 많은 브라우저에 지원됨	Chrome 42+, Firefox 39+, Edge 14+, and Safari 10.1+이상에 지원

설치

```
npm install axios
```

모듈 불러오기

```
import axios from 'axios';
```

axios 문법

```
axios({
  url: 'https://test/api/cafe/list/today', // 통신할 웹문서
  method: 'get', // 통신할 방식
  data: { // 인자로 보낼 데이터
    foo: 'diary'
  }
});
```

axios 요청(request) 파라미터 옵션

- **method** : 요청방식. (get이 디폴트)
- **url** : 서버 주소
- baseURL : url을 상대경로로 쓸대 url 맨 앞에 붙는 주소.
  - 예를들어, url이 /post 이고 baseURL이 https://some-domain.com/api/ 이면,https://some-domain.com/api/post로 요청 가게 된다.
- **headers** : 요청 헤더
- **data** : 요청 방식이 'PUT', 'POST', 'PATCH' 해당하는 경우 body에 보내는 데이터
- **params** : URL 파라미터 (?key=value 로 요청하는 url get방식을 객체로 표현한 것)
- timeout : 요청 timeout이 발동 되기 전 milliseconds의 시간을 요청. timeout 보다 요청이 길어진다면, 요청은 취소되게 된다.
- **responseType** : 서버가 응답해주는 데이터의 타입 지정 (arraybuffer, documetn, json, text, stream, blob)
- responseEncoding : 디코딩 응답에 사용하기 위한 인코딩 (utf-8)
- transformRequest : 서버에 전달되기 전에 요청 데이터를 바꿀 수 있다.
  - 요청 방식 'PUT', 'POST', 'PATCH', 'DELETE' 에 해당하는 경우에 이용 가능
  - 배열의 마지막 함수는 string 또는 Buffer, 또는 ArrayBuffer를 반환합
  - header 객체를 수정 가능
- transformResponse : 응답 데이터가 만들어지기 전에 변환 가능
- **withCredentials** : cross-site access-control 요청을 허용 유무. 이를 true로 하면 cross-origin으로 쿠키값을 전달 할 수 있다.
- auth : HTTP의 기본 인증에 사용. auth를 통해서 HTTP의 기본 인증이 구성이 가능
- maxContentLength: http 응답 내용의 max 사이즈를 지정하도록 하는 옵션

- validateStatus : HTTP응답 상태 코드에 대해 promise의 반환 값이 resolve 또는 reject 할지 지정하도록 하는 옵션
- maxRedirects : node.js에서 사용되는 리다이렉트 최대치를 지정
- httpAgent / httpsAgent : node.js에서 http나 https를 요청을 할때 사용자 정의 agent를 정의하는 옵션
- proxy : proxy서버의 hostname과 port를 정의하는 옵션
- cancelToken : 요청을 취소하는데 사용되어 지는 취소토큰을 명시

```
/* axios 파라미터 문법 예시 */

axios({
    method: "get", // 통신 방식
    url: "www.naver.com", // 서버
    headers: {'X-Requested-With': 'XMLHttpRequest'} // 요청 헤더 설정
    params: { api_key: "1234", langualge: "en" }, // ?파라미터를 전달
    responseType: 'json', // default

    maxContentLength: 2000, // http 응답 내용의 max 사이즈
    validateStatus: function (status) {
      return status >= 200 && status < 300; // default
    }, // HTTP응답 상태 코드에 대해 promise의 반환 값이 resolve 또는 reject 할지 지정
    proxy: {
      host: '127.0.0.1',
      port: 9000,
      auth: {
        username: 'mikeymike',
        password: 'rapunz3l'
      }
    }, // proxy서버의 hostname과 port를 정의
    maxRedirects: 5, // node.js에서 사용되는 리다이렉트 최대치를 지정
    httpsAgent: new https.Agent({ keepAlive: true }), // node.js에서 https를 요청을 할때
 사용자 정의 agent를 정의
})
.then(function (response) {
    // response Action
});
```

axios 응답(response) 데이터

- axios를 통해 요청을 서버에게 보내면, 서버에서 처리를하고 다시 데이터를 클라이언트에 응답 하게 된다.이를 .then으로 함수인자로(response)로 받아 객체에 담진 데이터가 바로 응답 데이터이다.ajax를 통해 **서버로부터 받는 응답의 정보**는 다음과 같다.

```
axios({
    method: "get", // 통신 방식
    url: "www.naver.com", // 서버
```

```
})
.then(function(response) {
  console.log(response.data)
  console.log(response.status)
  console.log(response.statusText)
  console.log(response.headers)
  console.log(response.config)
})
```

```
response.data: {}, // 서버가 제공한 응답(데이터)

response.status: 200, // `status`는 서버 응답의 HTTP 상태 코드

response.statusText: 'OK',  // `statusText`는 서버 응답으로 부터의 HTTP 상태 메시지

response.headers: {},  // `headers` 서버가 응답 한 헤더는 모든 헤더 이름이 소문자로 제공

response.config: {}, // `config`는 요청에 대해 `axios`에 설정된 구성(config)

response.request: {}
// `request`는 응답을 생성한 요청
// 브라우저: XMLHttpRequest 인스턴스
// Node.js: ClientRequest 인스턴스(리디렉션)
```

Axios 단축 메소드

- **GET** : axios.get(url[, config])
- **POST** : axios.post(url, data[, config])
- **PUT** : axios.put(url, data[, config])
- **DELETE** : axios.delete(url[, config])

```
axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])
```

axios GET

1. 단순 데이터(페이지 요청, 지정된 요청) 요청을 수행할 경우
2. 파라미터 데이터를 포함시키는 경우 (**사용자 번호에 따른 조회**)

```
const axios = require('axios'); // node.js쓸때 모듈 불러오기


// user에게 할당된 id 값과 함께 요청을 합니다.
axios.get('/user?ID=12345')
  .then(function (response) {
    // 성공했을 때
    console.log(response);
  })
  .catch(function (error) {
    // 에러가 났을 때
    console.log(error);
  })
  .finally(function () {
    // 항상 실행되는 함수
  });


// 위와는 같지만, 옵션을 주고자 할 때는 이렇게 요청을 합니다.
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .finally(function () {
    // always executed
  });


// async/await 를 쓰고 싶다면 async 함수/메소드를 만듭니다.
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

axios POST

- 데이터를 Message Body에 넣어서 전달

```
axios.post("url", {
    firstName: 'Fred',
    lastName: 'Flintstone'
    })
    .then(function (response) {
        // response
    }).catch(function (error) {
        // 오류발생시 실행
    })
```

axios Delete

- Body가 없는 delete

```
axios.delete('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
```

- Body가 있는 delete

```
axios.delete('/user?ID=12345',{
    data: {
      post_id: 1,
      comment_id: 13,
      username: "foo"
    }
  })
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
```

axios PUT

```
axios.put("url", {
        username: "",
        password: ""
```

```
    })
    .then(function (response) {
         // response
    }).catch(function (error) {
        // 오류발생시 실행
    })
```

axios Instance 만들기

- custom 속성을 지닌 axios 만의 instance를 만들 수 있다.

```
//axios.create([config])

const instance = axios.create({
  baseURL: 'https://some-domain.com/api/',
  timeout: 1000,
  headers: {'X-Custom-Header': 'foobar'}
});
```

axios로 formdata 보내기

```
const addCustomer = () => {
  const url = `/api/customers`;

  const formData = new FormData();
  formData.append("image", file);
  formData.append("name", userName);
  formData.append("birthday", birthday);
  formData.append("gender", gender);
  formData.append("job", job);

  const config = {
    headers: {
      "content-type": "multipart/form-data",
    },
  };

  return axios.post(url, formData, config);
};
```

원격 이미지 다운 받기 (blob)

```
const imgurl = 'https://play-lh.googleusercontent.com/hYdIazwJBlPhmN74Yz3m_jU9nA6t02U7
ZARfKunt6dauUAB6O3nLHp0v5ypisNt9OJk';

axios({
   url: imgurl,
   method: 'GET',
```

```
   responseType: 'blob' // blob 데이터로 이미지 리소스를 받아오게 지정
})
.then((response) => {
   const url = URL.createObjectURL(new Blob([response.data])); // blob 데이터를 객체 url로
변환

   // 이미지 자동 다운 로직
   const link = document.createElement('a');
   link.href = url
   link.setAttribute('download', `sample.png`)
   document.body.appendChild(link)
   link.click()
 })
```

axios 에러 핸들링 하기

```
axios
.get('/user/12345', {
  validateStatus: function (status) {
   return status < 500; // 만일 응답코드가 500일경우 reject()를 반환한다.
  }
})
.catch(function (error) {
    console.log(error.toJSON());
})
```

Hook - useReducer

useReducer 훅이란?

- useState를 대체할 수 있는 함수이다.
- `useReducer`는 `useState`처럼 `State`를 관리하고 업데이트 할 수 있는 Hook이다.
- 좀 더 복잡한 상태 관리가 필요한 경우 reducer를 사용할 수 있다.

useState와 useReducer 비교

- **useState**
  - 관리해야 할 `State`가 1개일 경우
  - 그 `State`가 단순한 숫자, 문자열 또는 `Boolean` 값일 경우
- **useReducer**
  - 관리해야 할 `State`가 1개 이상, 복수일 경우
  - 혹은 현재는 단일 `State` 값만 관리하지만, 추후 유동적일 가능성이 있는 경우

- 스케일이 큰 프로젝트의 경우
- `State`의 구조가 복잡해질 것으로 보이는 경우

useReducer 의 구성요소

- 1️⃣ useReducer 함수
- 2️⃣ action
- 3️⃣ dispatch 함수
- 4️⃣ reducer 함수

useReducer 사용

1. useReducer 함수 불러오기

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

2. useReducer 함수 사용

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

- state
  - 컴포넌트에서 사용할 State
- dispatch 함수
  - 첫번째 인자인 reducer 함수를 실행시킨다.
  - 컴포넌트 내에서 state의 업데이트를 일으키키 위해 사용하는 함수
- reducer 함수
  - 컴포넌트 외부에서 state를 업데이트 하는 함수
  - 현재state, action 객체를 인자로 받아, 기존의 state를 대체하여 새로운 state를 반환하는 함수
- initialState
  - 초기 state
- init
  - 초기 함수 (초기 state를 조금 지연해서 생성하기 위해 사용)

3. 예제

```
const [number, dispatch] = useReducer(reducer, 0);
```

action

- **업데이트를 위한 정보를 가지고 있는 것**
- `dispatch`의 인자가 되며, reducer 함수의 두번째 인자인 `action`에 할당된다.
- 따로 정해진 형태는 없으나, 주로 type라는 값을 지닌 객체 형태로 사용된다고 한다.
- 예제

```
dispatch({ type: "decrement" })
```

dispatch 함수

- reducer 함수를 실행 시킨다.
- action 객체를 인자로 받으며 action 객체는 어떤 행동인지를 나타내는 type 속성과 해당 행동과 관련된 데이터(payload)를 담고 있다.
- action을 이용하여 컴포넌트 내에서 state의 업데이트를 일으킨다.
- ex1) action type만 정의하여 사용

```
<button onClick={() => dispatch({ type: "INCREMENT" })}>증가</button>
```

- ex2) action type과, 데이터를 정의하여 사용

```
<button onClick={() => dispatch({ type: "INCREMENT", payload: 1 })}>증가</button>
```

reducer 함수

- dispatch 함수에 의해 실행되며, 컴포넌트 외부에서 state를 업데이트 하는 로직을 담당한다.
- 함수의 인자로는 state와 action을 받게 된다.
- state와 action을 활용하여 새로운 state를 반환 한다.
- ex1) action type만 정의하여 사용

```
functionreducer(state, action) {
switch (action.type) {
case "INCREMENT":
return { count: state.count + 1 };
case "DECREMENT":
return { count: state.count - 1 };
default:
thrownew Error("unsupported action type: ", action.type);
  }
}
```

- ex2) action type과, 데이터를 정의하여 사용

```
functionreducer(state, action) {
switch (action.type) {
case "INCREMENT":
return { count: state.count + action.payload };
case "DECREMENT":
return { count: state.count - action.payload };
default:
thrownew Error("unsupported action type: ", action.type);
  }
}
```

JavaScript

3항연산자

&& 연산자는 본인(조건)이 참이면 오른쪽으로 넘어가 다른 녀석을 return 하고,

본인이 거짓이면 본인을 그대로 return 한다.

반대로 **|| 연산자는 본인이 참이면 본인을 그대로 return** 하고,

본인이 거짓이면 오른쪽으로 넘어가 다른 녀석을 return 한다.