Axios 라이브러리

- Axios는 브라우저, Node.js를 위한 Promise API를 활용하는 HTTP 비동기 통신 라이브러리 아다.
- 쉽게 말해서 백엔드랑 프론트엔드랑 통신을 쉽게하기 위해 Ajax와 더불어 사용한다.
- 이미 자바스크립트에는 fetch api가 있지만, **프레임워크에서 ajax를 구현할땐 axios를 쓰는 편**이라고 보면 된다.

Latest ✓	Latest ✓	Latest ✓	Latest ✓	Latest ✓	11 ✓

axios 브라우저 호환성

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

위와 같은 표를 보았을 때 axios는 별도의 설치가 필요하다는 단점이 있지만 그것을 커버할 만한 fetch 보다 많은 기능 지원과 문법이 조금이나마 간소화 된다는 장점이 있다는 것을 볼 수 있다.

따라서, **간단하게 사용할때는 fetch**를 쓰고, 이외의 **확장성을 염두해봤을 땐 axios**를 쓰면 좋다고 보면 된다.

추가적으로 fetch에 대해 자세히 공부하고 싶다면, 다음 포스팅을 참고 바란다. fetch 문법과 axios 문법을 비교해보며 공부하는 방법도 좋다.

Axios 사용법

Axios 설치하기

서버에서 axios 설치

SHELL

```
> npm install axios
```

클라이언트(html)에서 axios 설치

- jsDeliver / unpkg CDN 둘중 택

HTML

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

HTML

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

Axios 문법 구성

JAVASCRIPT

```
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

> **Tip**
>
> **붉은색 표기한 파라미터**가 주로 많이 쓰이고 나머지는 참고 정도로 보면 된다.

- **method** : 요청방식. (get이 디폴트)
- **url** : 서버 주소
- baseURL : url을 상대경로로 쓸때 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

JAVASCRIPT

```
/* 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.

JAVASCRIPT

```
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)
})
```

JAVASCRIPT

```
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

axios를 편리하게 사용하기 위해 모든 요청 메소드는 aliases가 제공된다.

위 처럼 객체 옵션을 이것저것 주면 가독성이 떨어지고 너저분하니, 함수형으로 재구성하여 나눠논 것으로 이해하면 된다.

axios의 Request method에는 대표적으로 다음과 같은 것들이 있다.

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

JAVASCRIPT

```
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]])
```

제이쿼리를 배워보신 분들은 $.ajax()를 분리해 사용하는 $.get(), $.post()처럼 매우 흡사하다고 생각되어 질 것이다.

axios GET

get 메서드에는 2가지 상황이 크게 존재한다.

단순 데이터(페이지 요청, 지정된 요청) 요청을 수행할 경우

파라미터 데이터를 포함시키는 경우 (**사용자 번호에 따른 조회**)

JAVASCRIPT

```
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
  });
```

axios POST

post 메서드에는 일반적으로 **데이터를 Message Body에 포함시켜 보낸다.**
위에서 봤던 get 메서드에서 params를 사용한 경우와 비슷하게 수행된다.

JAVASCRIPT

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

axios Delete

delete 메서드에는 일반적으로 body가 비어있다.
REST 기반 API 프로그램에서 데이터베이스에 저장되어 있는 내용을 삭제하는 목적으로 사용한다.

JAVASCRIPT

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

하지만 query나 params가 많아져서 헤더에 많은 정보를 담을 수 없을 때는
다음과 같이 두 번째 인자에 data를 추가해줄 수 있다.

JAVASCRIPT

```
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

REST 기반 API 프로그램에서 데이터베이스에 저장되어 있는 내용을 갱신(수정)하는 목적으로 사용된다..
PUT메서드는 서버에 있는 데이터베이스의 내용을 변경하는 것을 주 목적으로 하고 있다.
put 메서드는 서버 내부적으로 get -> post 과정을 거치기 때문에 post 메서드와 비슷한 형태이다.

JAVASCRIPT

```
axios.put("url", {
        username: "",
        password: ""
    })
    .then(function (response) {
         // response
    }).catch(function (error) {
        // 오류발생시 실행
    })
```

다양한 Axios 응용 메소드

axios 동시 요청

JAVASCRIPT

```
function getUserAccount() {
  return axios.get('/user/12345');
}

function getUserPermissions() {
  return axios.get('/user/12345/permissions');
}

axios.all([getUserAccount(), getUserPermissions()])
  .then(axios.spread(function (acct, perms) {
    // 두개의 요청이 성공했을 때
  }));
```

axios Instance 만들기

custom 속성을 지닌 axios 만의 instance를 만들 수 있다.

JAVASCRIPT

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

axios로 formdata 보내기

JAVASCRIPT

```
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)

JAVASCRIPT

```
const imgurl = 'https://play-lh.googleusercontent.com/hYdlazwJBlPhmN74Yz3m_jU9nA6t02U7ZARfKunt6dauUAB603nLHp0v5ypisNt90Jk';
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

JAVASCRIPT

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

Axios 전역 설정 (Config Defaults)

모든 요청에 적용되는 설정의 default 값을 전역으로 명시할 수 있다.

주로 **서버에서 서버로 axios를 사용할때** 요청 헤더를 명시하는데 자주 쓰인다.

JAVASCRIPT

```
axios.defaults.baseURL = 'https://api.example.com';
axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';
```

JAVASCRIPT

```
// Instance를 만들 때 설정의 default 값을 설정할 수 있다.
const instance = axios.create({
  baseURL: 'https://api.example.com'
});
```

```
// Instance를 만든 후  defalut 값을 수정할 수 있다.
instance.defaults.headers.common['Authorization'] = AUTH_TOKEN;
instance.defaults.timeout = 2500;
```

Axios 실전 통신 예제

서버와 통신하는 방법

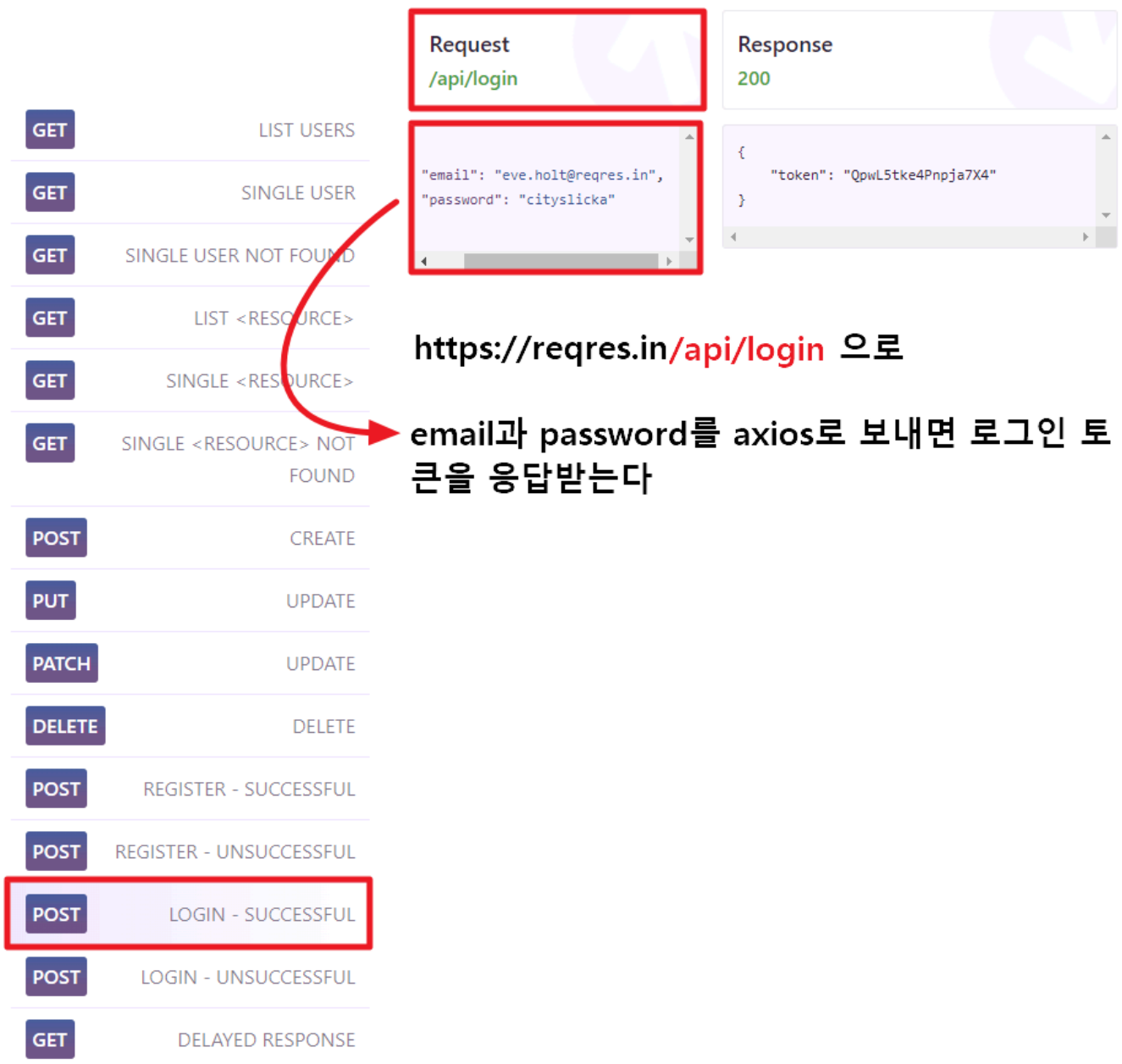
실제 API를 가지고 서버 통신을 테스트 해보고 싶으면 따로 서버가 있어야 하지만, 백엔드 개발자가 없어도 온라인에서 무료로 지원하는 Fake(가짜) 서버를 이용해 서버 통신 테스트 해볼 수 있다.

이번 포스팅에 사용할 것은 **reqres** 이라는 FakeServer를 이용해 axios 요청을 보내볼 예정이다.

Reqres - A hosted REST-API ready to respo…

reqres.in

Native JavaScript If you've already got your own application entities, ie. "products", you can send them in the endpoint URL, like so: var …



로그인 요청 해보기

아래 로그인 폼 창에 아래의 이메일과 비밀번호를 똑같이 입력해야 로그인 인증이 된다.

- email : eve.holt@reqres.in
- password : cityslicka

HTML

```
<head>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
</head>
<body>
  <div>
     <input type="email" placeholder="email을 입력해주세용" id="email" value="" />
  </div>
  <div>
     <input type="password" placeholder="비밀번호를 입력해주세용" id="pw" value="" />
  </div>
  <input type="button" onclick="onLoggin()" value="로그인" />
  <script>
     let onLoggin = async function () {
        const email = document.getElementById('email');
        const password = document.getElementById('pw');
        try {
           // axios으로 로그인 요청 (post)
           let res = await axios({
              method: 'POST',
              url: 'https://reqres.in/api/login',
              data: {
                 email: email.value,
                 password: password.value,
              },
           });
           console.log(res);
           document.write(JSON.stringify(res));
        } catch (err) {
           console.log(err);
           throw new Error(err);
        }
```

요청 결과

index.html:58

```
{data: {…}, status: 200, statusText: '', headers: {…}, config: {…}, …}
  config: {url: 'https://reqres.in/api/login', method: 'post', data: '{"email":"eve.holt@reqres.in","password":"cityslicka"}',…
  data: {token: 'QpwL5tke4Pnpja7X4'}
  headers: {content-length: '29', content-type: 'application/json; charset=utf-8'}
  request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false, upload: XMLHttpRequest…
  status: 200
  statusText: ""
  [[Prototype]]: Object
```

- 만일 Email를 잘못 입력 했을 경우 아래와 같이 400(Bad Request) 에러가 뜨게된다.

```
POST https://reqres.in/api/login 400                                   isAxiosError.js:10
Error: Request failed with status code 400                             index.html:60
    at e.exports (isAxiosError.js:10)
    at e.exports (isAxiosError.js:10)
    at XMLHttpRequest.y (isAxiosError.js:10)
Uncaught (in promise) Error: Error: Request failed with status code 400  index.html:61
    at onLoggin (index.html:61)
```