원쌤의 Vue.js 퀵스타트

11. axios를 이용한 HTTP 통신

1. axios란?

❖axios

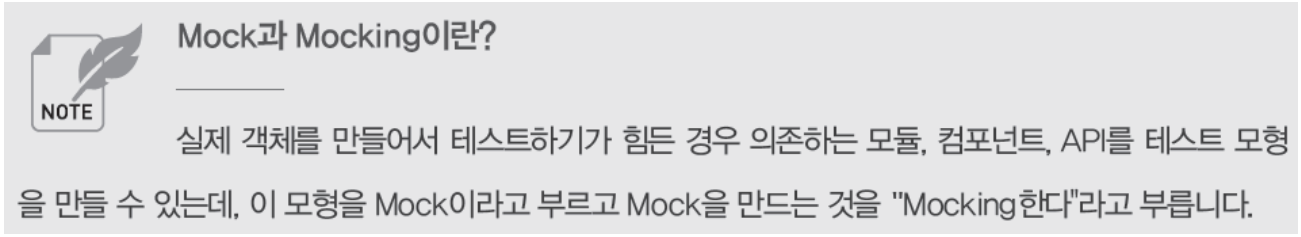
- HTTP 기반 통신을 지원하는 가장 많이 사용되는 자바스크립트 라이브러리
  - fetch와 같은 브라우저 내장 함수도 있지만 axios가 가장 강력하고 많이 사용됨

구분	axios	fetch
모듈 설치	설치해야 함 (npm install —save axios)	설치할 필요 없음 (브라우저 내장 API)
Promise API	사용	사용
브라우저 호환성	뛰어남	IE 지원하지 않음 (IE에서 사용하려면 Polyfill 라이브러리를 사용해야 함)
timeout 기능	지원 (timeout 시간 내에 응답이 오지 않으면 중단시킬 수 있음)	지원하지 않음
JSON 자동 변환	지원 (Content-type 정보를 이용해 자동으로 객체로 변환함)	지원하지 않음 (수신한 JSON 데이터를 객체로 변환하는 Promise 체인을 추가해야 함)

2. 테스트용 백엔드 API 소개

❖HTTP 통신을 테스트하려면 다음 중 하나가 준비되어야 함

- 백엔드 API
- 백엔드 API 모킹 도구 : json-server, mocky.io, mockoon
  - 이중 json-server는 json 파일 만으로 빠르게 백엔드 API 모형을 생성할 수 있음



- 필자가 만들어둔 RESTful API 서버
  - 접근 경로
    - https://todosvc.bmaster.kro.kr
  - 직접 다운로드 해서 로컬에서 실행할 수 있음
    - https://github.com/stepanowon/todosvc

2. 테스트용 백엔드 API 소개

❖ todosvc가 제공하는 API 목록

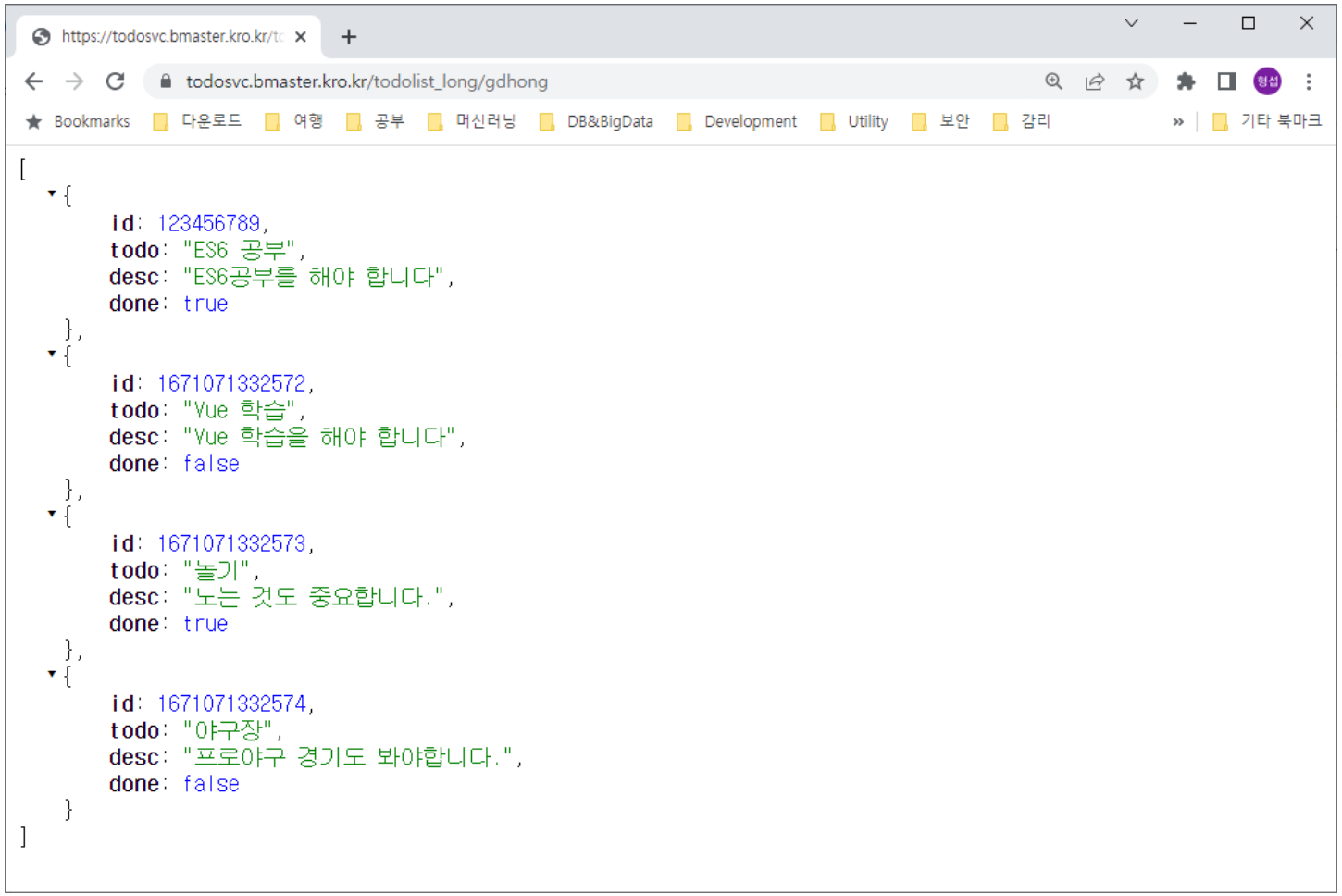
API	설명
GET /todolist/:owner	특정 사용자(owner)의 todolis(할 일 목록)을 조회합니다. 서비스가 처음 시작하면 gdhong 사용자의 할 일 4건이 포함되어 있습니다.
GET /todolist/:owner/:id	특정 사용자(owner)의 todo(할 일) 한 건을 조회합니다. id 파라미터는 todo의 고유 키입니다.
POST /todolist/:owner	사용자의 todolist에 새로운 todo 한 건을 추가합니다.
PUT /todolist/:owner/:id	사용자의 id에 해당하는 todo 한 건을 변경합니다.
PUT /todolist/:owner/:id/done	todo 한 건의 완료 여부인 true/false 값을 토글합니다.
DELETE /todolist/:owner/:id	todo 한 건을 삭제합니다
GET /todolist/:owner/create	새로운 사용자(owner)를 위한 샘플 todo 데이터 3건을 생성합니다.

- /todolist_long 으로 시작하는 경로 : 1초의 의도적 지연 시간 발생

2. 테스트용 백엔드 API 소개

❖ 백엔드 API 요청 결과

- gdhong 사용자의 할일 목록은 기본 제공됨



3. 프로젝트 생성과 크로스 오리진 오류 발생

❖ 프로젝트 생성

```
npm init vue axios-test-app
cd axios-test-app
npm install
npm install axios
```

- src/components 디렉터리 내부의 것들을 모두 삭제

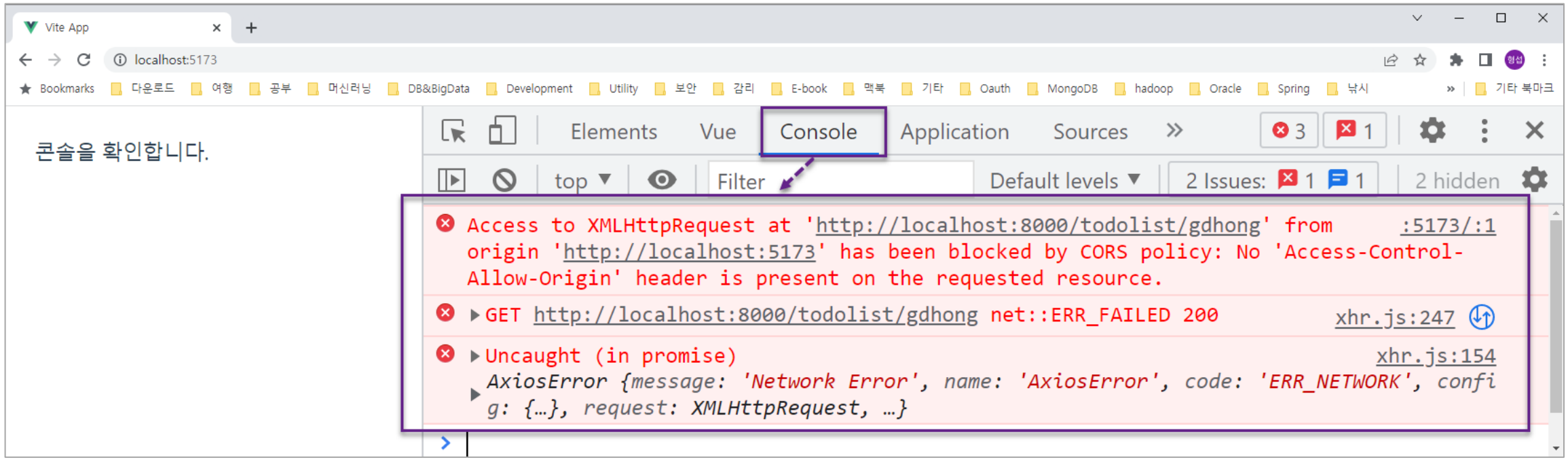
❖예제 11-01 : src/App.vue 새롭게 작성

```vue
<template>
  <div>
    <h2>콘솔을 확인합니다.</h2>
  </div>
</template>

<script setup>
import axios from 'axios'

const requestAPI = () => {
    const url = "http://localhost:8000/todolist/gdhong";

    axios.get(url).then((response) => {
        console.log("# 응답객체 : ", response);
    });
};

requestAPI();
</script>
```

3. 프로젝트 생성과 크로스 오리진 오류 발생

❖ 실행 결과 오류 발생

- 크로스 오리진 오류



4. 크로스 오리진 문제란?

❖크로스 오리진 문제

- 개념 : "브라우저는 자신의 오리진과 다른 오리진의 API 서버와 통신할 때 문제가 발생한다"
  - 잠재적인 위험을 가진 문서의 로딩을 제한해 공격의 가능성을 줄일 수 있음
- 오류 발생 이유
  - SOP(Same Origin Policy)

❖ Browser Origin

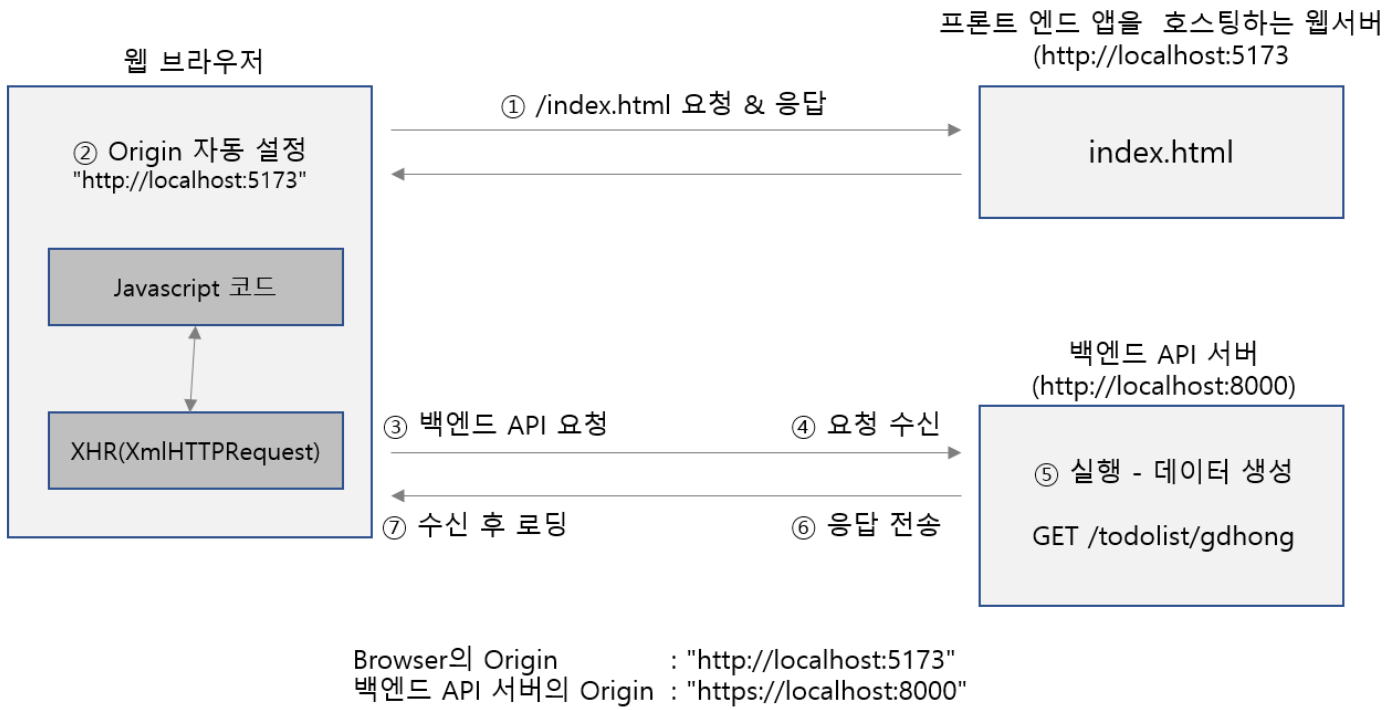
- origin : "현재 보여지고 있는 HTML 문서의 원천지는 이곳입니다"

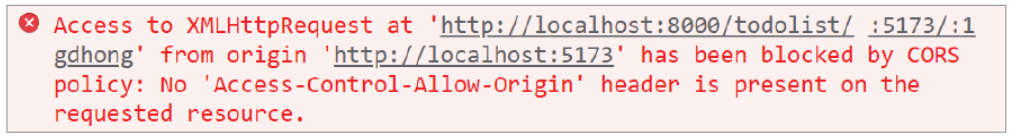

```
> location.origin
< 'http://localhost:5173'
```

4. 크로스 오리진 문제란?

❖SOP : Same Origin Policy

- 브라우저 내부 보안 정책
- "브라우저의 오리진과 동일한 오리진을 가진 서버일 때만 통신을 가능하게 한다"
- 크로스 오리진일 때는 통신 문제 발생





Browser의 Origin : "http://localhost:5173"
백엔드 API 서버의 Origin : "https://localhost:8000"

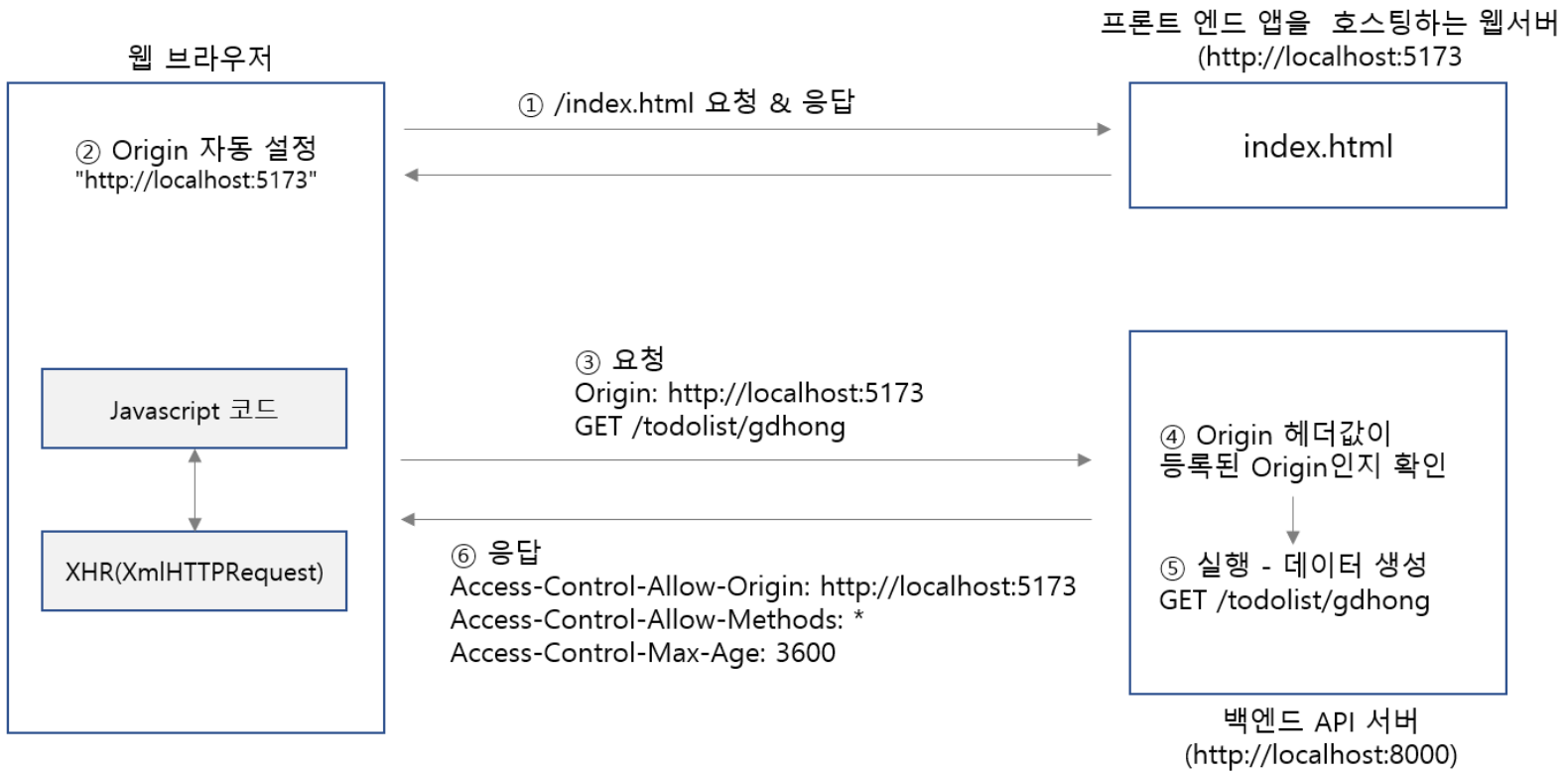
5. 크로스 오리진 문제 해결 방법

- CORS(Cross Origin Resource Sharing)
- 프런트 엔드 애플리케이션을 호스팅하는 웹서버에 프록시 설치

5.1 CORS

❖이 책에서는 개념만 학습

- 백엔드 API에서 제공할 수 있는 기능이므로 이 책의 내용 범주를 벗어남
- 크로스 오리진의 브라우저가 백엔드 API 서버로 요청했을 때, 서버에서 Access-Control-Allow-Origin HTTP 헤더로 브라우저의 오리진을 응답하여 브라우저가 통신 및 데이터 로딩을 할 수 있도록 허용하는 방법



5.1 CORS

❖ CORS 요청 흐름

- 브라우저는 프런트 서버에서 HTML 문서를 받아와 자신의 오리진을 설정합니다.
- 자바스크립트 코드로 백엔드 API 서버로 요청합니다. 이때 자신의 오리진을 Origin HTTP 헤더에 추가합니다.
- 백엔드 API 서버는 전송된 Origin 헤더를 읽어내어 등록된 리스트에 일치하는 것이 있는지 확인합니다(이 단계는 선택적입니다).
- 백엔드 API 서버는 Access-Control-Allow-Origin 응답 헤더를 추가하고 * 또는 브라우저의 오리진을 값으로 지정하여 응답합니다.
- 브라우저는 자신의 오리진과 백엔드 API 서버로부터 전송받은 Access-Control-Allow-Origin 헤더가 일치하면 허가된 것으로 간주하고 데이터를 로딩합니다.

❖CORS에 대한 더 자세한 내용은 다음 문서 확인

- https://developer.mozilla.org/ko/docs/Web/HTTP/Access_control_CORS

5.2 프록시를 이용한 우회

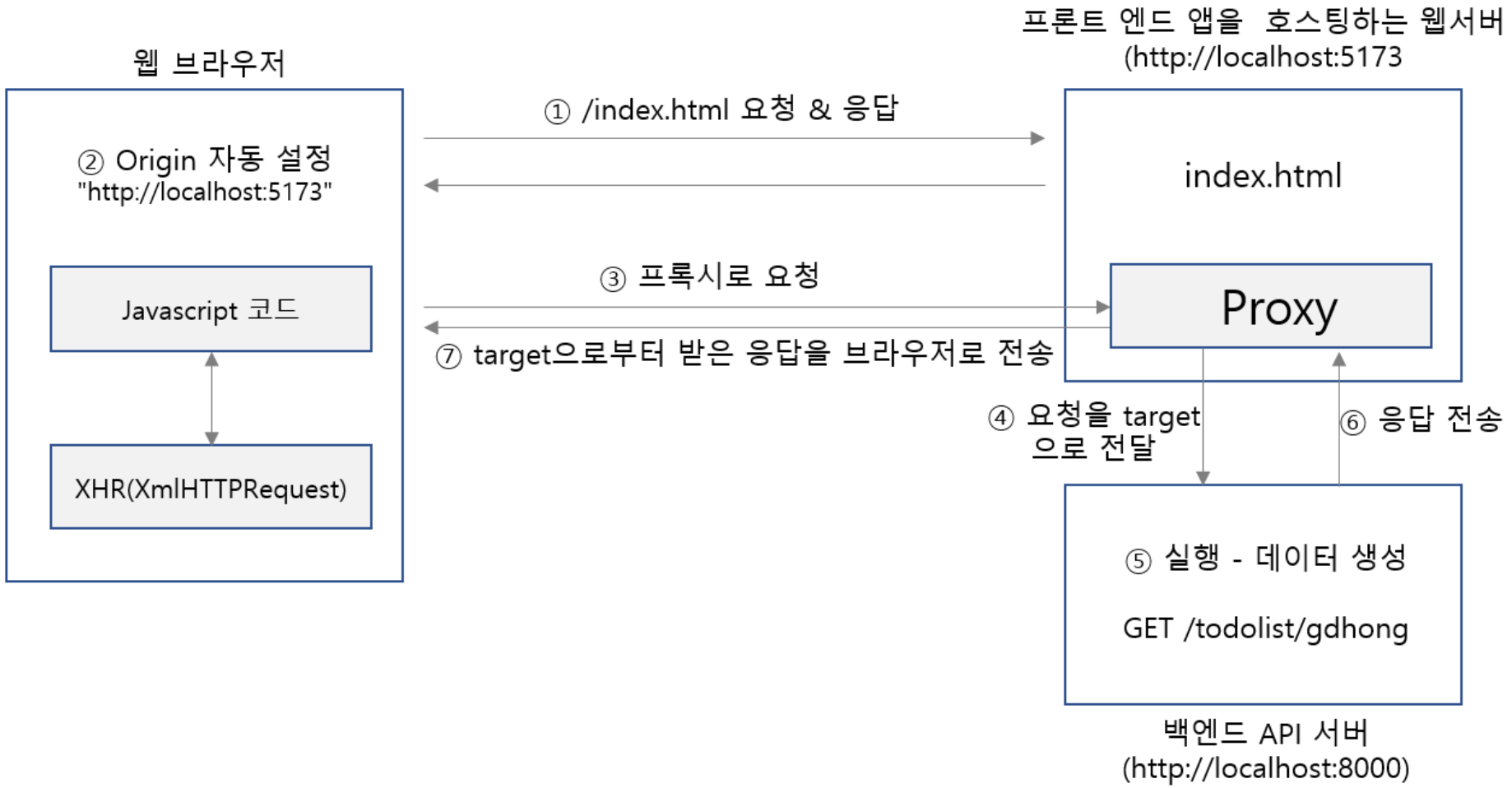
❖ 프록시를 이용한 우회 방법
- 프런트 엔드 애플리케이션을 호스팅하는 서버에 프록시를 설치
- 브라우저가 프런트 서버의 프록시를 거쳐서 백엔드API와 통신
  - 브라우저 측에서는 동일 오리진과 통신하는 것임

❖ 프런트 앱을 백엔드 API 서버에 호스팅하면 되지 않나?
- 운영환경에서는 프론트 엔드, 백엔드가 분리된 경우가 많음
- 적어도 개발중에는 개발용 웹서버가 백엔드 API와 분리됨

❖ 따라서 프록시를 운영서버, 개발용 웹서버에 설정하는 방법을 알고 있어야 함

5.2 프록시를 이용한 우회

❖ 프록시를 사용한 경우의 개요도



5.2 프록시를 이용한 우회

❖ Vite 기반 프로젝트의 개발용 웹서버에 프록시 설정하기

- vite.config.js

```
1  import { fileURLToPath, URL } from 'node:url'
2
3  import { defineConfig } from 'vite'
4  import vue from '@vitejs/plugin-vue'
5
6  // https://vitejs.dev/config/
7  export default defineConfig({
8    plugins: [vue()],
9    resolve: {
10     alias: {
11       '@': fileURLToPath(new URL('./src', import.meta.url))
12     }
13   },
14   server: {
15     proxy: {
16       "/api": {
17         target: "http://localhost:8000",
18         changeOrigin: true,
19         rewrite: (path) => path.replace(/^\/api/, ""),
20       },
21     },
22   },
23 })
```

- 최초 요청 경로: /api/todolist/gdhong
- 타깃: http://localhost:8000
- 최종 전달 경로: http://localhost:8000/todolist/gdhong

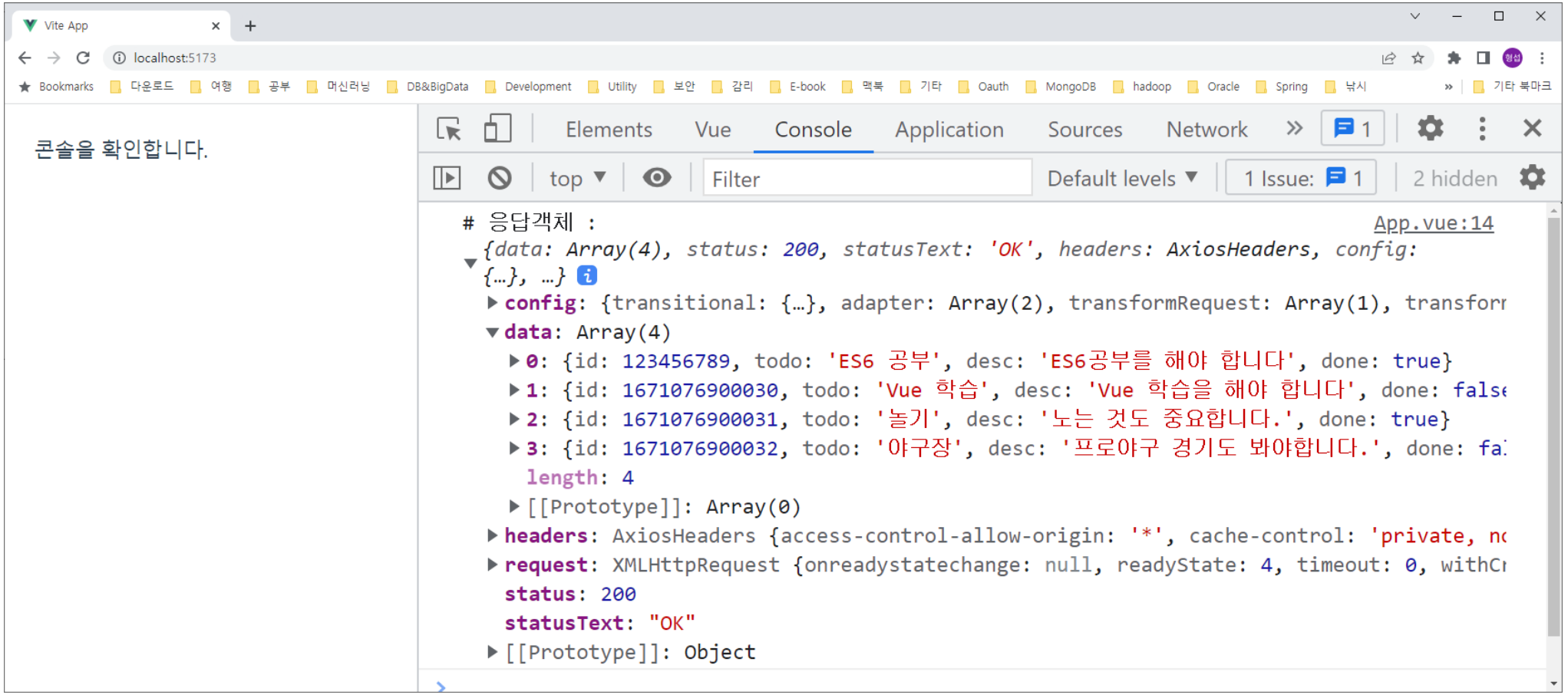
5.2 프록시를 이용한 우회

❖ 예제 11-03 : src/App.vue 변경 - url을 프록시 경로로 변경

```
1   <template>
2     <div>
3       <h2>콘솔을 확인합니다.</h2>
4     </div>
5   </template>
6
7   <script setup>
8   import axios from 'axios'
9
10  const requestAPI = () => {
11      //const url = "http://localhost:8000/todolist/gdhong";
12      const url = "/api/todolist/gdhong";
13      axios.get(url).then((response) => {
14          console.log("# 응답객체 : ", response);
15      });
16  };
17
18  requestAPI();
19  </script>
```

5.2 프록시를 이용한 우회

❖ 프록시 지정 후 실행 결과



5.2 프록시를 이용한 우회

❖ 운영 환경의 프론트 서버에 프록시를 설정하는 방법?

- 웹서버의 종류에 따라 제각각임
- "웹서버 기술명 + http proxy" 키워드로 검색해봄

  - Node.js + express 기반의 웹서버

    https://github.com/chimurai/http-proxy-middleware

  - Python Django

    https://github.com/mjumbewu/django-proxy

  - JSP/Servlet

    https://github.com/mitre/HTTP-Proxy-Servlet

6. axios 라이브러리 사용법

- Promise와 async~await
- axios 라이브러리 사용 방법
- 에러 처리

6.1 Promise와 async~await

❖ Promise

- 2장 ES6에서 이미 학습한 비동기 처리를 수행하는 패턴 지원
- 하지만 비동기 작업을 순차적으로 실행되도록 하려면 then()이 반복되어야 함
- 예제 11-04 : src/App.vue 변경

```
<template>
  <div>
    <h2>콘솔을 확인합니다.</h2>
  </div>
</template>

<script setup>
import axios from 'axios'

const listUrl = "/api/todolist_long/gdhong";
const todoUrlPrefix = "/api/todolist_long/gdhong/";

//4건의 목록을 조회한 후 첫번째, 두번째 할일을 순차적으로 조회합니다.
const requestAPI = () => {
  let todoList = [];
  axios
    .get(listUrl)
    .then((response) => {
      todoList = response.data;
      console.log("# TodoList : ", todoList);
      return todoList[0].id;
    })
    .then((id) => {
      return axios.get(todoUrlPrefix + id);
    })
    .then((response) => {
      console.log("## 첫번째 Todo : ", response.data);
      return todoList[1].id;
    })
    .then((id) => {
      axios.get(todoUrlPrefix + id).then((response) => {
        console.log("## 두번째 Todo : ", response.data);
      });
    });
};

requestAPI();
</script>
```

6.1 Promise와 async~await

❖ 예제 11-04 실행 흐름

- 순차적으로 결과 출력
- 하지만 뭔가 복잡하고 관리가 어려워보임

6.1 Promise와 async~await

❖ async/await 문법

- 비동기로 호출할 함수의 앞부분에 async 키워드 추가
  - async를 붙이는 것은 함수가 Promise를 리턴하거나 비동기 실행을 한다는 의미
- 함수 내부에서는 Promise를 리턴하는 함수 호출 구문 앞에 await 부여
  - await 구문을 붙인 호출 구문은 Promise가 처리될 때까지 대기하고 결과는 처리가 완료된 후에 리턴

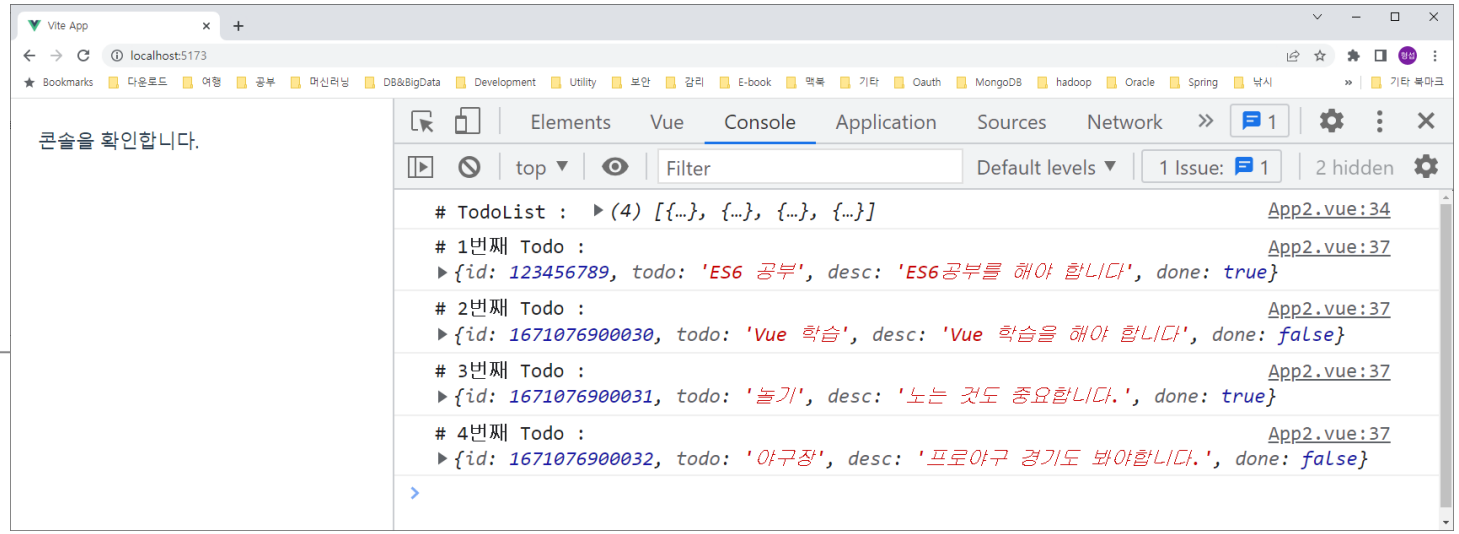
❖ 예제 11-05 : src/App2.vue 추가

```
<template>
    <div>
        <h2>콘솔을 확인합니다.</h2>
    </div>
</template>

<script setup>
import axios from 'axios'

const listUrl = "/api/todolist_long/gdhong";
const todoUrlPrefix = "/api/todolist_long/gdhong/";

//4건의 목록을 조회한 후 첫번째, 두번째 할일을 순차적으로 조회합니다.
const requestAPI = async () => {
    let todoList;

    let response = await axios.get(listUrl);
    todoList = response.data;
    console.log("# TodoList : ", todoList);

    response = await axios.get(todoUrlPrefix + todoList[0].id);
    console.log("## 첫번째 Todo : ", response.data);

    response = await axios.get(todoUrlPrefix + todoList[1].id);
    console.log("## 두번째 Todo : ", response.data);
};

requestAPI();
</script>
```

6.1 Promise와 async~await

❖전체 데이터를 순회하면서 순차적으로 비동기 처리할 때

```
//전체 목록을 조회한 후 한 건씩 순차적으로 순회하며 조회하기
const requestAPI = async () => {
  let todoList: Array<TodoType>;

  let response = await axios.get(listUrl);
  todoList = response.data;
  console.log("# TodoList : ", todoList);
  for (let i = 0; i < todoList.length; i++) {
    response = await axios.get(todoUrlPrefix + todoList[i].id);
    console.log(`# ${i + 1}번째 Todo : `, response.data);
  }
};
```



6.1 Promise와 async~await

❖ 예외 처리

- Promise
  - catch() 함수
- async~await
  - try ~ catch 문을 이용하여 catch(e) { } 블록으로 처리

```
// promise의 catch 블록
axios.get( ..... )
.then((response)=> { ..... })
.catch((error) => {
if (e instanceof Error) console.log(e.message);
      else console.log(e);
})
```

```
//async/await 예외 처리
//11장 예제 코드 axios-test-app61 폴더의 App21.vue를 참조하세요
const requestAPI = async () => {
  let todoList;

  try {
    let response = await axios.get(listUrl);
    todoList = response.data;
    console.log("# TodoList : ", todoList);
    for (let i = 0; i < todoList.length; i++) {
      response = await axios.get(todoUrlPrefix + todoList[i].id);
      console.log(`# ${i + 1}번째 Todo : `, response.data);
    }
  } catch (e) {
    if (e instanceof Error) console.log(e.message);
    else console.log(e);
  }
};
```

6.2 axios 라이브러리 사용 방법

❖ axios.get()

- GET 요청 처리를 해주는 메서드
- 사용 방법과 예시

[사용방법]

```
// url : 요청하는 백엔드 API 의 URL을 지정합니다
// config : 요청시에 지정할 설정값들입니다.
// 요청 후에는 Promise를 리턴하며 처리가 완료된 후에는 response 객체를 응답받습니다.
axios.get( url, config)
```

[사용 예시 : Promise]

```
const requestAPI = () => {
  const url = "/api/todolist/gdhong";
  axios.get(url)
  .then((response) => {
    console.log("# 응답객체 : ", response);
  });
};
requestAPI();
```

[사용 예시 : async/await]

```
const requestAPI = async () => {
  const url = "/api/todolist/gdhong";
  const response = await axios.get(url);
  console.log("# 응답객체 : ", response);
};
requestAPI();
```

6.2 axios 라이브러리 사용 방법

- 예제 11-06 : src/App3.vue

```vue
<template>
  <div>
    <h2>콘솔을 확인합니다.</h2>
  </div>
</template>

<script setup>
import axios from "axios";

const requestAPI = async () => {
  const url = "/api/todolist/gdhong";
  const response = await axios.get(url);
  console.log("# 응답객체 : ", response);
};

requestAPI();
</script>
```

```
# 응답객체 :                                                                    App3.vue:13
▼{data: Array(4), status: 200, statusText: 'OK', headers: AxiosHeaders, config: {…}, …}
  ▼config:
    ▶adapter: (2) ['xhr', 'http']
     data: undefined
    ▶env: {FormData: ƒ, Blob: ƒ}
    ▶headers: AxiosHeaders {Accept: 'application/json, text/plain, */*', Content-Type: null}
     maxBodyLength: -1
     maxContentLength: -1
     method: "get"
     timeout: 0
    ▶transformRequest: [ƒ]
    ▶transformResponse: [ƒ]
    ▶transitional: {silentJSONParsing: true, forcedJSONParsing: true, clarifyTimeoutError: false}
     url: "/api/todolist/gdhong"
    ▶validateStatus: ƒ validateStatus(status)
     xsrfCookieName: "XSRF-TOKEN"
     xsrfHeaderName: "X-XSRF-TOKEN"
    ▶[[Prototype]]: Object
  ▶data: (4) [{…}, {…}, {…}, {…}]
  ▶headers: AxiosHeaders {access-control-allow-origin: '*', cache-control: 'private, no-cache, no-store, must-revalidate
  ▶request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false, upload: XMLHttp
   status: 200
   statusText: "OK"
  ▶[[Prototype]]: Object
```

6.2 axios 라이브러리 사용 방법

▪ 응답 객체

- data: 수신된 응답 데이터입니다.
- config: 요청시에 사용된 config 옵션입니다.
- headers: 백엔드 API 서버가 응답할 때 사용된 응답 HTTP 헤더입니다.
- request: 서버와의 통신에 사용된 XMLHttpRequest 객체의 정보입니다.
- status: 서버가 응답한 HTTP 상태 코드입니다.
- statusText: 서버의 HTTP 상태를 나타내는 문자열 정보입니다.

```
axios.get( url, {
   timeout : 2000,
   headers : { Authorization : "Bearer xxxxxxxxx" }
} )
```

- config 에 대한 상세 정보 : https://axios-http.com/kr/docs/req_config

6.2 axios 라이브러리 사용 방법

❖ axios.post()

- POST 요청을 처리하는 함수
- 주로 백엔드 API로 데이터를 전달해 데이터를 추가할 때 사용함

```
// url, config는 axios.get()과 동일합니다.
// data는 POST 요청의 HTTP Content Body로 전송할 데이터입니다.
axios.post( url, data, config )
```

- 예제 11-07 : src/App4.vue 추가

```
7   <script setup>
8   import axios from "axios";
9
10  const requestAPI = async () => {
11    const url = "/api/todolist_long/gdhong";
12    let data = { todo: "윗몸일으키기 3세트", desc: "너무 빠르지 않게..." };
13    const resp1 = await axios.post(url, data);
14    console.log(resp1.data);
15  };
16  requestAPI();
17  </script>
```

```
▼ {status: 'success', message: '추가 성공', item: {…}}
  ▶ item: {id: 1671080104568, todo: '윗몸일으키기 3세트', desc: '너무 빠르지 않게...'}
    message: "추가 성공"
    status: "success"
  ▶ [[Prototype]]: Object
```

6.2 axios 라이브러리 사용 방법

❖ 기타 axios 함수
- axios.put()
- axios.delete()

❖ axios 기본 설정 변경

```
// todolist_long 은 1초의 의도적 지연시간을 일으키는 엔드포인트임
axios.defaults.baseURL = '/api/todolist_long';

//인증 토큰은 백엔드 API 요청시 항상 전달하므로 기본값으로 설정할 수 있음
axios.defaults.headers.common['Authorization'] = JWT;

//timeout에 설정된 시간내에 응답이 오지 않으면 연결을 중단(abort)시킴
axios.defaults.timeout = 2000;
```

6.3 에러 처리

❖ 이전 절에서 에러 처리 방법을 이미 학습. 바로 예제로

- 예) 타임아웃 오류
- 예제 11-08 : src/App5.vue, App6.vue

```
<script setup>
import axios from "axios";

const requestAPI = async () => {
  const url = "/api/todolist_long/gdhong";
  try {
    const response = await axios.get(url, { timeout: 900 });
    console.log("# 응답객체 : ", response);
  } catch (e) {
    console.log("## 다음 오류가 발생했습니다.");
    if (e instanceof Error) console.log(e.message);
    else console.log(e);
  }
};

requestAPI();
</script>
```

```
<script setup>
import axios from "axios";

const requestAPI = async () => {
  const url = "/api/todolist_long/gdhong";
  axios
    .get(url, { timeout: 900 })
    .then((response) => {
      console.log("# 응답객체 : ", response);
    })
    .catch((e) => {
      if (e instanceof Error) console.log(e.message);
      else console.log(e);
    });
};

requestAPI();
</script>
```

6.3 에러 처리

❖ 예외 처리하지 않았을 때와 했을 때

```
▶Uncaught (in promise)                                                xhr.js:167
▶AxiosError {message: 'timeout of 900ms exceeded', name: 'AxiosError', code: 'ECONNABOR
TED', config: {…}, request: XMLHttpRequest, …}
```

그림 11-14 예외 처리를 하지 않았을 때의 오류 발생

```
## 다음 오류가 발생했습니다.                                            App5.vue:16
timeout of 900ms exceeded                                             App5.vue:17
```

그림 11-15 예외 처리를 수행한 경우

7. 마무리

Vue와 같은 프런트엔드 애플리케이션은 영속성을 가진 데이터를 처리하기 위해 반드시 외부 API와 통신해야 합니다. 따라서 axios와 같은 HTTP 통신 라이브러리를 반드시 다룰 줄 알아야 합니다. 여러 가지 라이브러리리들이 있지만 axios가 가장 강력하고 Vue의 창시자인 에반 유도 사용을 추천하고 있습니다.

또한 크로스 오리진 문제의 해결 방법도 숙지하고 해결할 수 있어야 합니다. 특히 개발 서버이든 Vue애플리케이션을 호스팅하는 웹서버이든 HTTP 프록시를 설정하는 방법을 익히고 적용할 수 있어야 합니다. Vite 기반의 Vue 프로젝트에서는 vite.config.js 파일을 통해 손쉽게 프록시를 설정할 수 있습니다.