* 예습
1. 6/6

44. REST API

TABLE OF CONTENTS

REST(Representational State Transfer)는 HTTP/1.0과 1.1의 스펙 작성에 참여하였고 아파치 HTTP 서버 프로젝트의 공동 설립자인 로이 필딩(Roy Fielding)의 2000년 논문에서 처음 소개되었다. 발표 당시의 웹이 HTTP를 제대로 사용하지 못하고 있는 상황을 보고 HTTP의 장점을 최대한 활용할 수 있는 아키텍쳐로서 REST를 소개하였고 이는 HTTP 프로토콜을 의도에 맞게 디자인하도록 유도하고 있다. REST의 기본 원칙을 성실히 지킨 서비스 디자인을 “RESTful”이라고 표현한다.

즉, REST는 HTTP를 기반으로 클라이언트가 서버의 리소스에 접근하는 방식을 규정한 아키텍쳐이고, REST API는 REST를 기반으로 서비스 API를 구현한 것을 의미한다.

#1. REST API의 구성

REST API는 자원(Resource), 행위(Verb), 표현(Representations)의 3가지 요소로 구성된다. REST는 자체 표현 구조(Self-descriptiveness)로 구성되어 REST API만으로 요청을 이해할 수 있다.

구성 요소	내용	표현 방법
자원(Resource)	자원	HTTP URI
행위(Verb)	자원에 대한 행위	HTTP 요청 메소드
표현(Representations)	자원에 대한 행위의 구체적 내용	HTTP 페이로드

#2. REST API 설계 방침

REST에서 가장 중요한 기본적인 규칙은 두 가지이다. URI는 리소스를 표현하는 데에 집중하고 행위에 대한 정의는 HTTP 요청 메소드를 통해 하는 것이 REST한 API를 설계하는 중심 규칙이다.

1. URI**는 리소스를 표현해야 한다**.

리소스를 식별할 수 있는 이름은 동사보다는 명사를 사용한다. URI는 리소스를 표현하는데 중점을 두어야 한다. 리소스 이름에 get 같은 행위에 대한 표현이 들어가서는 안된다.

CODE

```
# bad
GET /getTodos/1
GET /todos/show/1

# good
GET /todos/1
```

2. **리소스에 대한 행위는** HTTP **요청 메소드로 표현한다**.

리소스를 취득하는 경우에는 GET, 리소소를 삭제하는 경우에는 DELETE 메소드를 사용하여 리소스에 대한 행위를 명확히 표현한다. 리소스에 대한 행위는 GET, POST, PUT, PATCH, DELETE와 같은

HTTP 요청 메소드(“43.3.2. HTTP 요청 전송”의 XMLHttpRequest.prototype.open 참고)를 통해 표현하며 URI에 표현하지 않는다.

CODE

```
# bad
GET /todos/delete/1

# good
DELETE /todos/1
```

3. JSON Server를 사용한 REST API 실습

3.1. JSON Server 설치

JSON Server는 json 파일을 사용하여 REST API Mock server를 구축할 수 있는 툴이다. 사용법은 매우 간단하다. 먼저 npm을 사용하여 JSON Server를 설치하도록 하자.

> npm
> npm(node package manager)은 자바스크립트 패키지 매니저이다. Node.js에서 사용할 수 있는 모듈들을 패키지화하여 모아둔 저장소 역할과 패키지 설치 및 관리를 위한 CLI(Command line interface)를 제공한다. 자신이 작성한 패키지를 공개할 수도 있고 필요한 패키지를 검색하여 재사용할 수도 있다. npm에 대한 보다 자세한 내용은 아래를 참고하기 바란다.
> - 모듈화와 npm(node package manager)

터미널에서 아래 명령어를 사용하여 JSON Server를 설치한다.

BASH

```
$ mkdir json-server-exam && cd json-server-exam
$ npm init -y
$ npm install json-server --save-dev
```

3.2. db.json 파일 생성

프로젝트 루트 폴더(/json-server-exam)에 아래와 같이 db.json 파일을 생성한다. db.json 파일은 리소스를 제공하는 데이터베이스 역할을 한다.

JSON

```
{
  "todos": [
    {
      "id": 1,
      "content": "HTML",
      "completed": true
    },
    {
      "id": 2,
      "content": "CSS",
      "completed": false
    },
    {
      "id": 3,
      "content": "Javascript",
      "completed": true
    }
  ]
}
```

3.3. JSON Server 실행

터미널에서 아래와 같이 명령어를 입력하여 JSON Server를 실행한다. JSON Server가 데이터베이스 역할을 하는 db.json 파일의 변경을 감지하도록 하려면 watch 옵션을 추가하고, 기본 포트는 3000을 변경하려면 port 옵션을 추가한다.

BASH

```
## 기본 포트(3000) 사용 / watch 옵션 적용
$ json-server --watch db.json
## 포트 변경  / watch 옵션 적용
$ json-server --watch db.json --port 5000
```

위와 같이 매번 명령어를 입력하는 것이 번거로우니 package.json 파일의 scripts를 아래와 수정하여 JSON Server를 실행하여 보자. 불필요한 항목은 삭제하였다.

JSON

```
{
  "name": "json-server-exam",
  "version": "1.0.0",
  "scripts": {
    "start": "json-server --watch db.json"
  },
  "devDependencies": {
    "json-server": "^0.16.1"
  }
}
```

터미널에서 아래와 같이 명령어를 입력하여 JSON Server를 실행한다.

BASH

```
$ npm start

> json-server-exam@1.0.0 start /Users/leeungmo/Desktop/json-server-exam
> json-server --watch db.json

  \{^_^}/ hi!

  Loading db.json
  Oops, db.json doesn't seem to exist
  Creating db.json with some default data

  Done
```

```
  Resources
  http://localhost:3000/posts
  http://localhost:3000/comments
  http://localhost:3000/profile

  Home
  http://localhost:3000

  Type s + enter at any time to create a snapshot of the database
  Watching ...
```

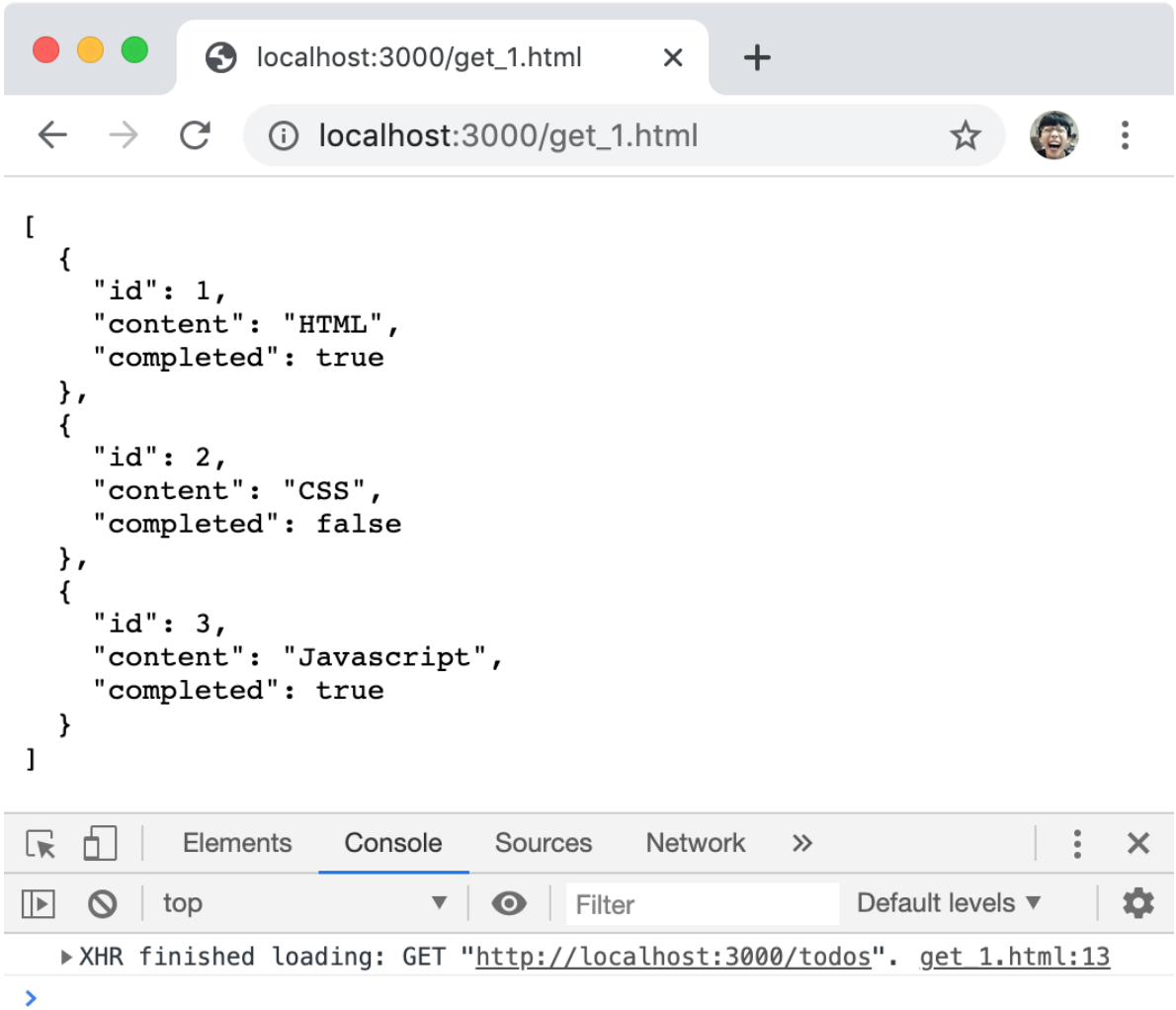
3.4. GET 요청

todos 리소스에서 모든 todo를 취득(index)한다. JSON Server의 루트 폴더에 public 폴더를 생성하고 JSON Server를 중단한 후, 재실행한다. 그리고 아래 get_1.html을 추가하고 브라우저에서 `http://localhost:3000/get_1.html` 으로 접속한다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <pre></pre>
  <script>
    // XMLHttpRequest 객체 생성
    const xhr = new XMLHttpRequest();

    // HTTP 요청 초기화
    // todos 리소스에서 모든 todo를 취득(index)
    xhr.open('GET', '/todos');

    // HTTP 요청 전송
    xhr.send();

    // load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
    xhr.onload = () => {
      // status는 response 상태 코드를 반환 : 200 => 정상 응답
      if (xhr.status === 200) {
```

```
        document.querySelector('pre').textContent = xhr.response;
      } else {
        console.error('Error', xhr.status, xhr.statusText);
      }
    };
  </script>
</body>
</html>
```



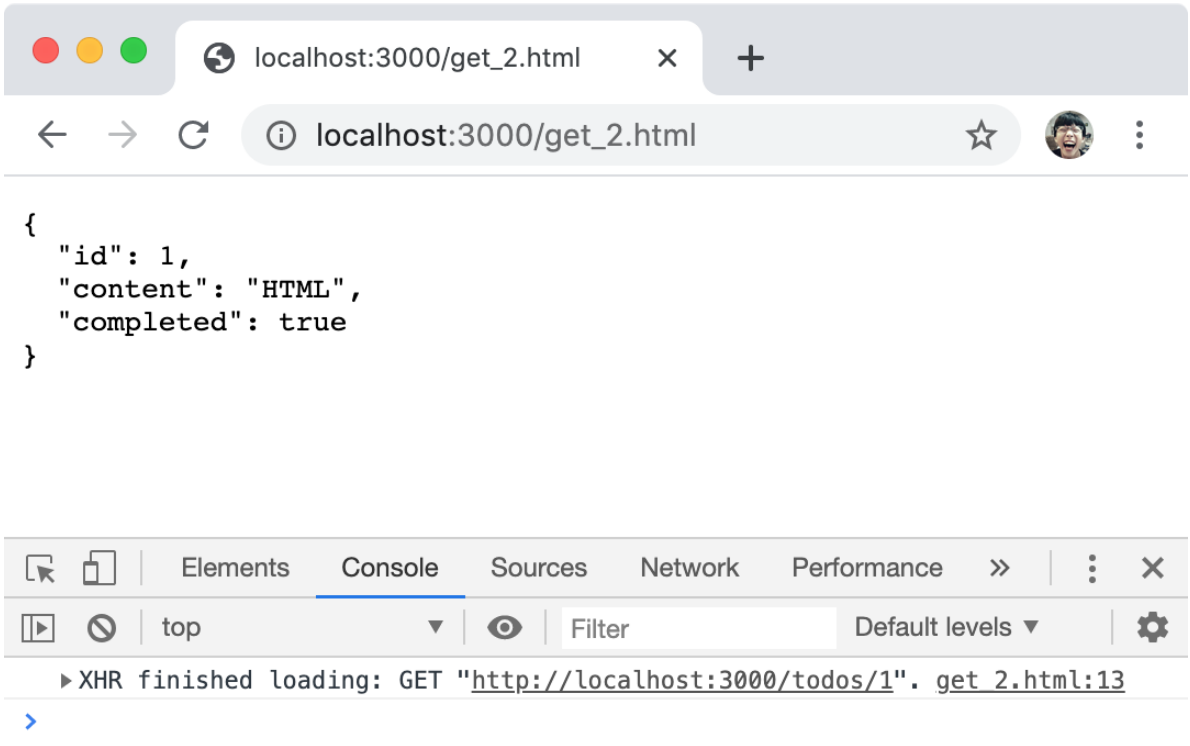
GET 요청(index)

todos 리소스에서 id를 사용하여 특정 todo를 취득(retrieve)한다. 아래 get_2.html을 추가하고 브라우저에서 `http://localhost:3000/get_2.html` 으로 접속한다.

HTML

```
<!DOCTYPE html>
<html>
```

```
<body>
  <pre></pre>
  <script>
    // XMLHttpRequest 객체 생성
    const xhr = new XMLHttpRequest();

    // HTTP 요청 초기화
    // todos 리소스에서 id를 사용하여 특정 todo를 취득(retrieve)
    xhr.open('GET', '/todos/1');

    // HTTP 요청 전송
    xhr.send();

    // load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
    xhr.onload = () => {
      // status는 response 상태 코드를 반환 : 200 => 정상 응답
      if (xhr.status === 200) {
        document.querySelector('pre').textContent = xhr.response;
      } else {
        console.error('Error', xhr.status, xhr.statusText);
      }
    };
  </script>
</body>
</html>
```



GET 요청(retrieve)

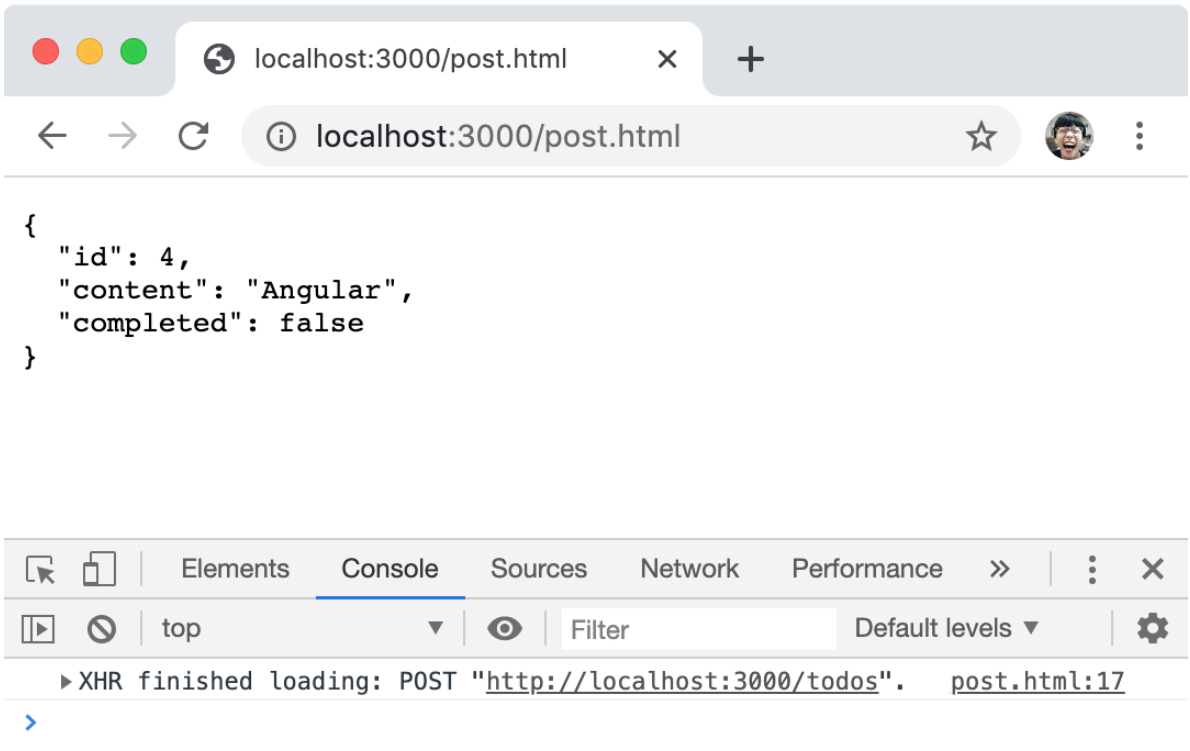
3.5. POST 요청

todos 리소스에 새로운 todo를 생성한다. POST 요청 시에는 setRequestHeader 메소드를 사용하여 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정하도록 한다. 아래 post.html을 추가하고 브라우저에서 `http://localhost:3000/post.html` 으로 접속한다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <pre></pre>
  <script>
    // XMLHttpRequest 객체 생성
    const xhr = new XMLHttpRequest();

    // HTTP 요청 초기화
    // todos 리소스에 새로운 todo를 생성
    xhr.open('POST', '/todos');

    // 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정
```

```
    xhr.setRequestHeader('content-type', 'application/json');

    // HTTP 요청 전송
    // 새로운 todo를 생성하기 위해 페이로드가 필요하다.
    xhr.send(JSON.stringify({ id: 4, content: 'Angular', completed: false }));

    // load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
    xhr.onload = () => {
      // status는 response 상태 코드를 반환 : 200 => 정상 응답
      // 201 Created ⇒ 새로운 리소스가 생성되었습니다.
      if (xhr.status === 200 || xhr.status === 201) {
        document.querySelector('pre').textContent = xhr.response;
      } else {
        console.error('Error', xhr.status, xhr.statusText);
      }
    };
  </script>
</body>
</html>
```



POST 요청

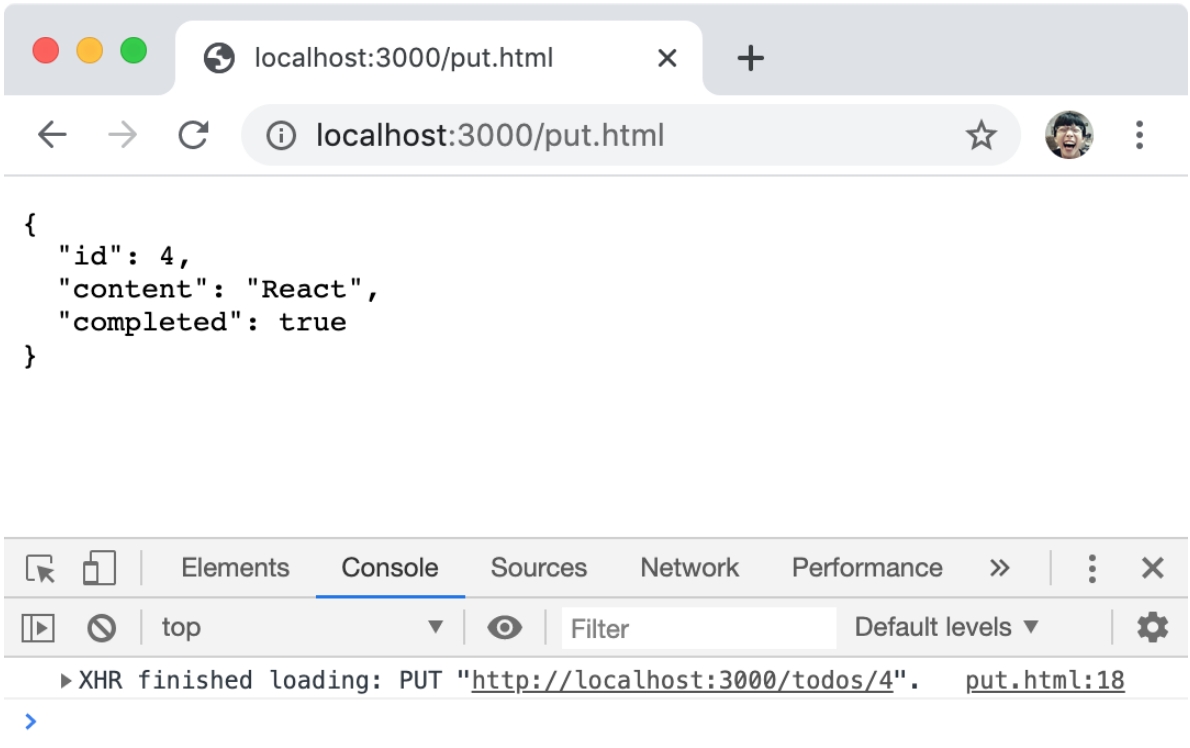
3.6. PUT 요청

PUT은 특정 리소스의 전체를 교체할 때 사용한다. todos 리소스에서 id를 사용하여 todo를 특정하여 id를 제외한 리소스 전체를 교체한다. PUT요청 시에는 setRequestHeader 메소드를 사용하여 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정하도록 한다. 아래 put.html을 추가하고 브라우저에서 `http://localhost:3000/put.html` 으로 접속한다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <pre></pre>
  <script>
    // XMLHttpRequest 객체 생성
    const xhr = new XMLHttpRequest();

    // HTTP 요청 초기화
    // todos 리소스에서 id를 사용하여 todo를 특정하여 id를 제외한 리소스 전체를 교체
    xhr.open('PUT', '/todos/4');

    // 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정
    xhr.setRequestHeader('content-type', 'application/json');

    // HTTP 요청 전송
    // 리소스 전체를 교체하기 위해 페이로드가 필요하다.
    xhr.send(JSON.stringify({ id: 4, content: 'React', completed: true }));

    // load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
    xhr.onload = () => {
      // status는 response 상태 코드를 반환 : 200 => 정상 응답
      // 201 Created ⇒ 새로운 리소스가 생성되었습니다.
      if (xhr.status === 200 || xhr.status === 201) {
        document.querySelector('pre').textContent = xhr.response;
      } else {
        console.error('Error', xhr.status, xhr.statusText);
      }
    };
  </script>
```

```
</body>
</html>
```



PUT 요청

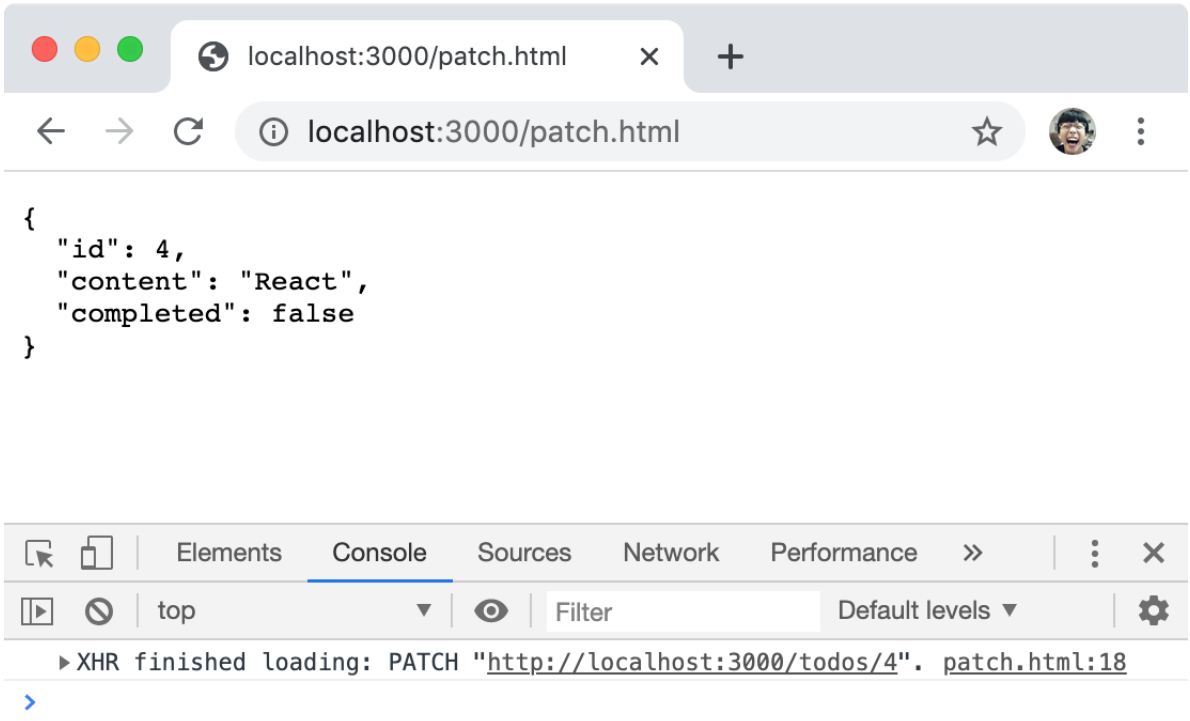
3.7. PATCH 요청

PATCH는 특정 리소스의 일부를 수정할 때 사용한다. todos 리소스의 id를 사용하여 todo를 특정하여 completed만을 수정한다. PATCH요청 시에는 setRequestHeader 메소드를 사용하여 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정하도록 한다. 아래 patch.html을 추가하고 브라우저에서 `http://localhost:3000/patch.html` 으로 접속한다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <pre></pre>
  <script>
    // XMLHttpRequest 객체 생성
    const xhr = new XMLHttpRequest();
```

```
    // HTTP 요청 초기화
    // todos 리소스의 id를 사용하여 todo를 특정하여 completed만을 수정
    xhr.open('PATCH', '/todos/4');

    // 요청 몸체에 담아 서버로 전송할 페이로드의 MIME-type을 지정
    xhr.setRequestHeader('content-type', 'application/json');

    // HTTP 요청 전송
    // 리소스를 수정하기 위해 페이로드가 필요하다.
    xhr.send(JSON.stringify({ completed: false }));

    // load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
    xhr.onload = () => {
      // status는 response 상태 코드를 반환 : 200 => 정상 응답
      // 201 Created ⇒ 새로운 리소스가 생성되었습니다.
      if (xhr.status === 200 || xhr.status === 201) {
        document.querySelector('pre').textContent = xhr.response;
      } else {
        console.error('Error', xhr.status, xhr.statusText);
      }
    };
  </script>
</body>
</html>
```



PATCH 요청

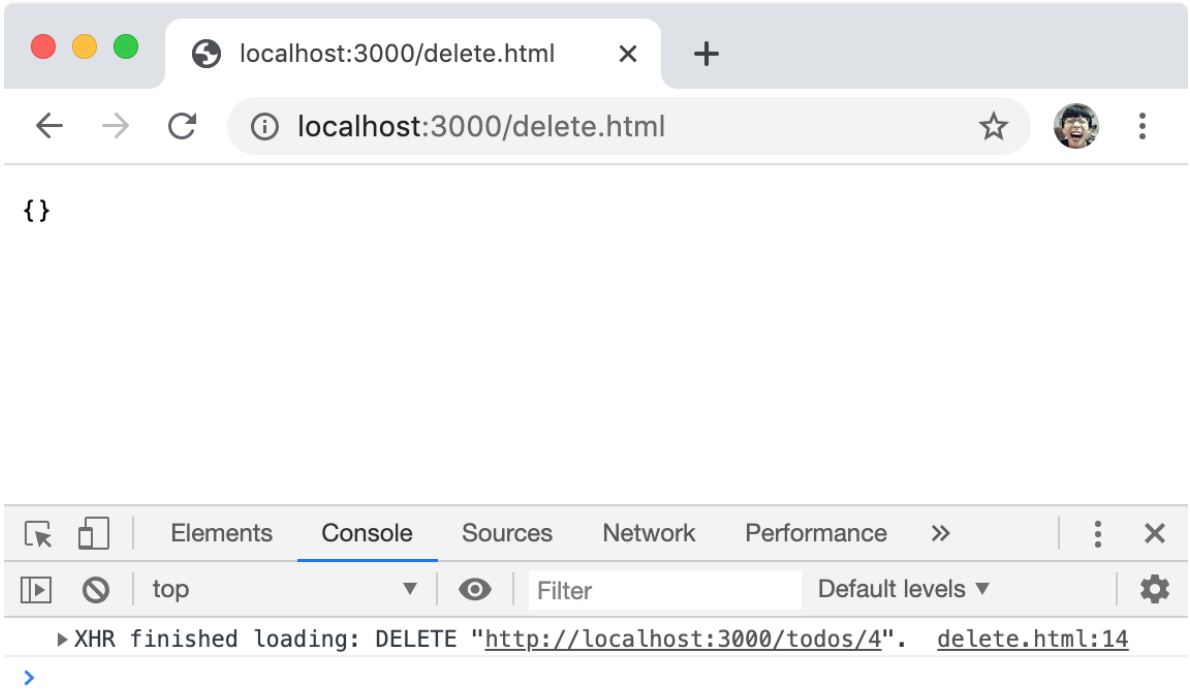
3.8. DELETE 요청

todos 리소스에서 id를 사용하여 todo를 삭제한다. 아래 delete.html을 추가하고 브라우저에서 `http://localhost:3000/delete.html` 으로 접속한다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <pre></pre>
  <script>
    // XMLHttpRequest 객체 생성
    const xhr = new XMLHttpRequest();

    // HTTP 요청 초기화
    // todos 리소스에서 id를 사용하여 todo를 삭제한다.
    xhr.open('DELETE', '/todos/4');

    // HTTP 요청 전송
    xhr.send();
```

```
    // load 이벤트는 요청이 성공적으로 완료된 경우 발생한다.
    xhr.onload = () => {
      // status는 response 상태 코드를 반환 : 200 => 정상 응답
      if (xhr.status === 200) {
        document.querySelector('pre').textContent = xhr.response;
      } else {
        console.error('Error', xhr.status, xhr.statusText);
      }
    };
  </script>
</body>
</html>
```



DELETE 요청