8월 29일 (일)

1. Codecademy - Async JavaScript and HTTP Requests - Learn JavaScript Syntax : Async-Await

Lesson 2. Requests - 2 -

- **Introduction to Requests with ES6**
- **fetch( ) GET Requests -1-**
- **fetch( ) GET Requests -2-**
- **fetch( ) GET Requests -3-**
- **fetch( ) GET Requests -4-**
- **fetch( ) POST Requests -1-**
- **fetch( ) POST Requests -2-**
- **fetch( ) POST Requests -3-**
- **fetch( ) POST Requests -4-**
- **fetch( ) POST Requests -5-**
- **async GET Requests -1-**
- **async GET Requests -2-**
- **async GET Requests -3-**
- **async POST Requests -1-**
- **async POST Requests -2-**
- **async POST Requests -3-**
- **Review**

- **Introduction to Requests with ES6**

이전 레슨에서 우리는 비동기적 데이터를 다루는 것을 배웠다. 우리 웹페이지의 많은 상호작용은 비동기적 이벤트에 의존하고, 그래서 이러한 이벤트를 관리하는 것은 좋은 웹페이지 개발에 필수적인 것이다.

비동기적 이벤트 핸들링을 더 쉽게 하기 위해, 프로미스를 사용한다.

프로미스는 비동기적 데이터를 다루는 객체로 세 가지 상태를 갖는다. (pending, fulfilled, rejected)

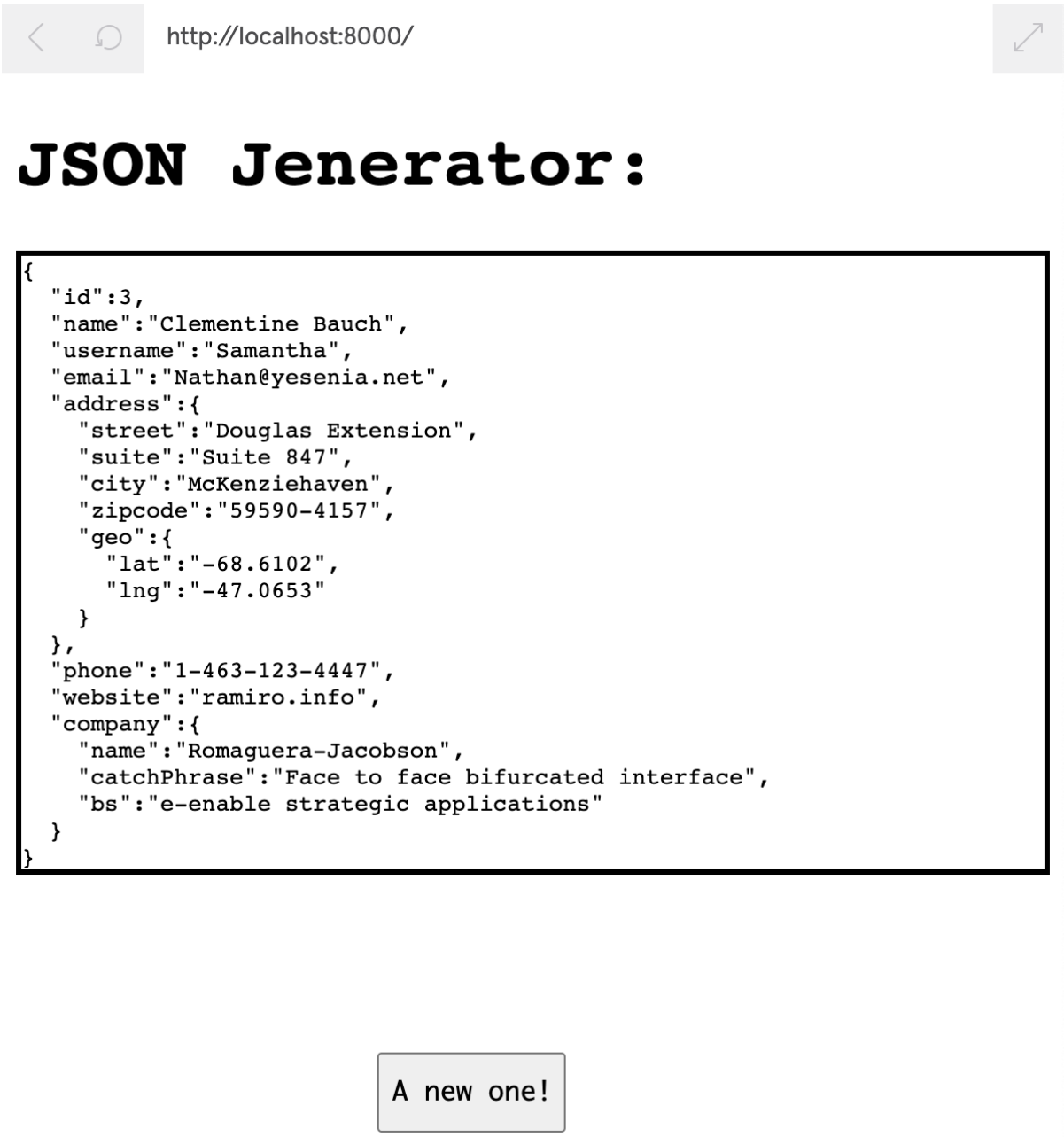
프로미스가 좋은게 일단 하나의 프로미스가 fulfilled 되거나 rejected 되면, 뒤이어 추가적인 메서드를 체이닝할 수 있게 된다.

이번 레슨에서 요청을 다루기 위한 프로미스를 사용하는 fetch( ) 를 어떻게 사용하는지 배울 것이다.

그다음 async await 을 사용하여 요청을 간소화할 것이다.

우리는 GET 요청을 위해 Datamuse API 사용하고, POST 요청을 위해 Rebrandly URL Shortener API를 사용할 것이다.

```
//main.js JSON Generator 코드



const jsonButton = document.querySelector('#generate');
const buttonContainer = document.querySelector('#buttonContainer');
const display = document.querySelector('#displayContainer');
const collection = ["Another", "More", "Next", "Continue", "Keep going", "Click me", "A new one"];

const generateJson = async () => {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/users');
    if(response.ok){
      const jsonResponse = await response.json();
      renderResponse(jsonResponse);
      changeButton();
    }
  } catch(error) {
    console.log(error);
  }
};

const formatJson = (resJson) => {
  resJson = JSON.stringify(resJson);
  let counter = 0;
  return resJson.split('')
  .map(char => {
    switch (char) {
      case ',':
        return `,\n${' '.repeat(counter * 2)}`;
      case '{':
        counter += 1;
        return `{\n${' '.repeat(counter * 2)}`;
      case '}':
        counter -= 1;
        return `\n${' '.repeat(counter * 2)}}`;
      default:
        return char;
    }
  })
  .join('');
};

const renderResponse = (jsonResponse) => {
  const jsonSelection = Math.floor(Math.random() * 10);
  display.innerHTML = `<pre>${formatJson(jsonResponse[jsonSelection])}</pre>`;
};

const changeButton = () => {
  const newText = Math.floor(Math.random() * 7);
  jsonButton.innerHTML = `${collection[newText]}!`;
};

jsonButton.addEventListener('click', generateJson);
```



- **fetch( ) GET Requests -1-**

우리가 배울 첫 번째 요청 타입은 fetch( )를 사용하는 GET 요청이다.

fetch( ) 함수는:

- API가 필요한 관련 정보를 포함하는 요청 객체를 생성한다.
- 제공된 API endpoint 에 요청 객체를 보낸다.
- 반응 객체(response object)에 완전히 resolve 한 하나의 프로미스를 리턴한다. 이것은 API가 돌려보낸 정보와 함께 프로미스의 상태를 포함한다.

- **fetch( ) GET Requests -2-**

위 다이어그램을 다시 써보는 연습

```
fetch('https://api-to-call.com/endpoint').then(response => { // 첫 번째 then
  if (response.ok) {
    return response.json();
  }
  throw new Error('Request failed!'); // 만약 if문이 false 라면 에러를 출력

}, networkError => console.log(networkError.message) // 첫 번째 then 함수안에 , 로 다른 익명함수를 작성 fetch에서 networkError가 날 시 실행
).then(jsonResponse => {.  // 두 번째 then
  return jsonResponse;
});
```

- **fetch( ) GET Requests -3-**

이전 연습에서 우리는 fetch( ), .then( ) 을 사용하여 GET 요청을 위한 코드를 써봤다. 이번 연습에서 Datamuse API에 접근하고 코드를 조작하는 코드를 사용 해볼 것이다. 그리고 브라우저에 정보를 렌더링할 것이다.

만약 요청이 성공적으로 완료되면, 인풋 필드에 입력한 비슷하게 들리는 단어의 배열을 얻게 될 것이다.

실습 예제

```
// main.js

노란색배경이 연습 때 작성한 코드
```

```
// Information to reach API      // 직접 작성한 부분
const url = 'https://api.datamuse.com/words';
const queryParams = '?sl=';

// Selects page elements
const inputField = document.querySelector('#input');
const submit = document.querySelector('#submit');
const responseField = document.querySelector('#responseField');

// AJAX function    // 직접 작성한 부분
const getSuggestions = () => {
  const wordQuery = inputField.value;
  const endpoint = `${url}${queryParams}${wordQuery}`;
  // 제공된 브라우저에서 API가 작동하기 위해 두 번째 인자에 해당 캐시코드를 넣어줬다.
  fetch(endpoint, {cache: 'no-cache'}).then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error('Request failed!');
  }, networkError => console.log(networkError.message))
};

// 직접 작성 X
// Clears previous results and display results to webpage
const displaySuggestions = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild);
  }
  getSuggestions();
};

submit.addEventListener('click', displaySuggestions);
```

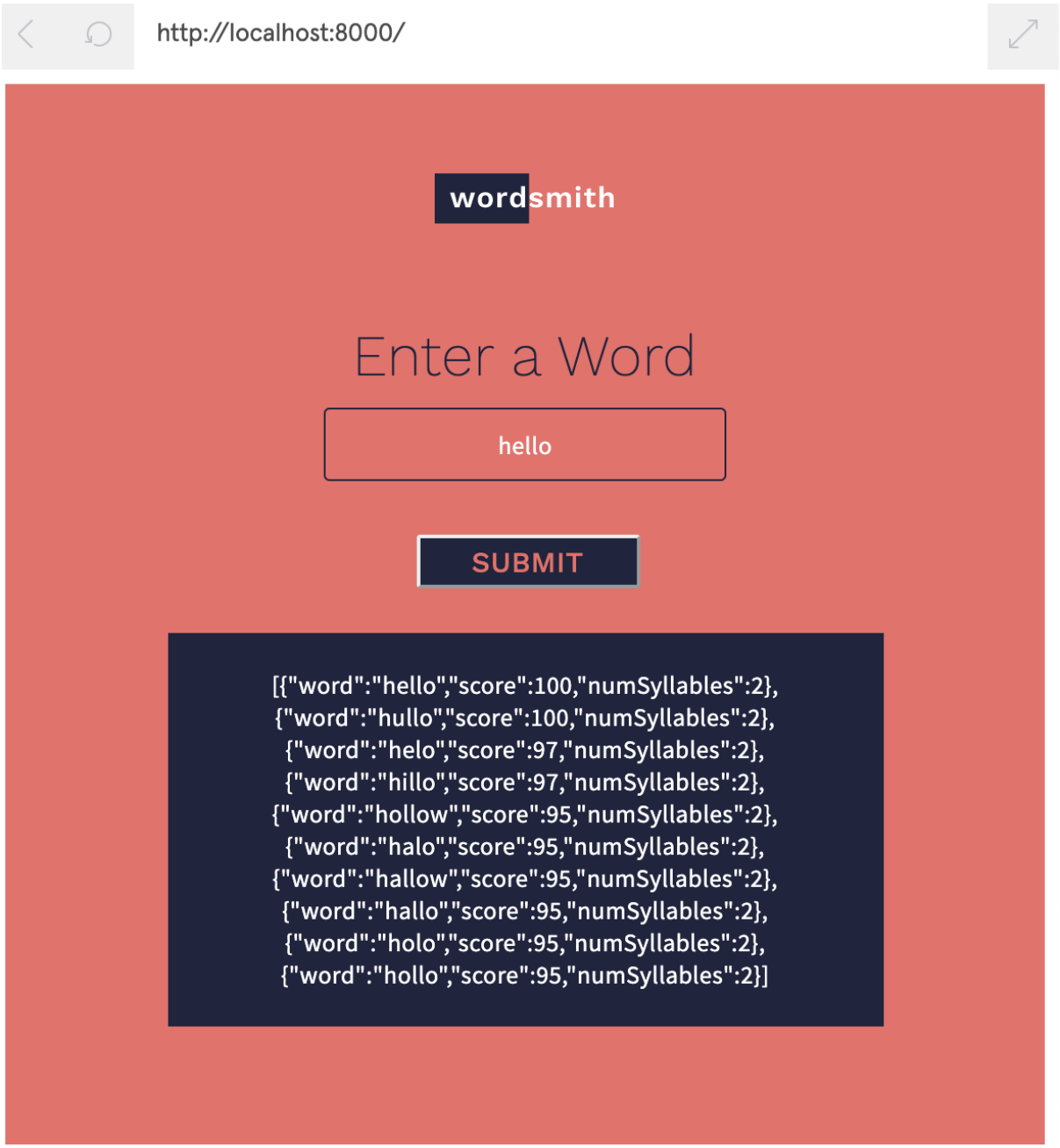
```
//helperFunction.js


// Formats response to look presentable on webpage
const renderResponse = (res) => {
  // Handles if res is falsey
  if(!res){
    console.log(res.status);
  }
  // In case res comes back as a blank array
  if(!res.length){
    responseField.innerHTML = "<p>Try again!</p><p>There were no suggestions found!</p>";
    return;
  }

  // Creates an empty array to contain the HTML strings
  let wordList = [];
  // Loops through the response and caps off at 10
  for(let i = 0; i < Math.min(res.length, 10); i++){
    // creating a list of words
    wordList.push(`<li>${res[i].word}</li>`);
  }
  // Joins the array of HTML strings into one string
  wordList = wordList.join("");

  // Manipulates responseField to render the modified response
  responseField.innerHTML = `<p>You might be interested in:</p><ol>${wordList}</ol>`;
  return
}

// Renders response before it is modified
const renderRawResponse = (res) => {
  // Takes the first 10 words from res
  let trimmedResponse = res.slice(0, 10);
  // Manipulates responseField to render the unformatted response
  responseField.innerHTML = `<text>${JSON.stringify(trimmedResponse)}</text>`;
}

// Renders the JSON that was returned when the Promise from fetch resolves.
const renderJsonResponse = (res) => {
  // Creates an empty object to store the JSON in key-value pairs
  let rawJson = {};
  for(let key in res){
```

```
    rawJson[key] = res[key];
  }
  // Converts JSON into a string and adding line breaks to make it easier to read
  rawJson = JSON.stringify(rawJson).replace(/,/g, ", \n");
  // Manipulates responseField to show the returned JSON.
  responseField.innerHTML = `<pre>${rawJson}</pre>`;
}
```

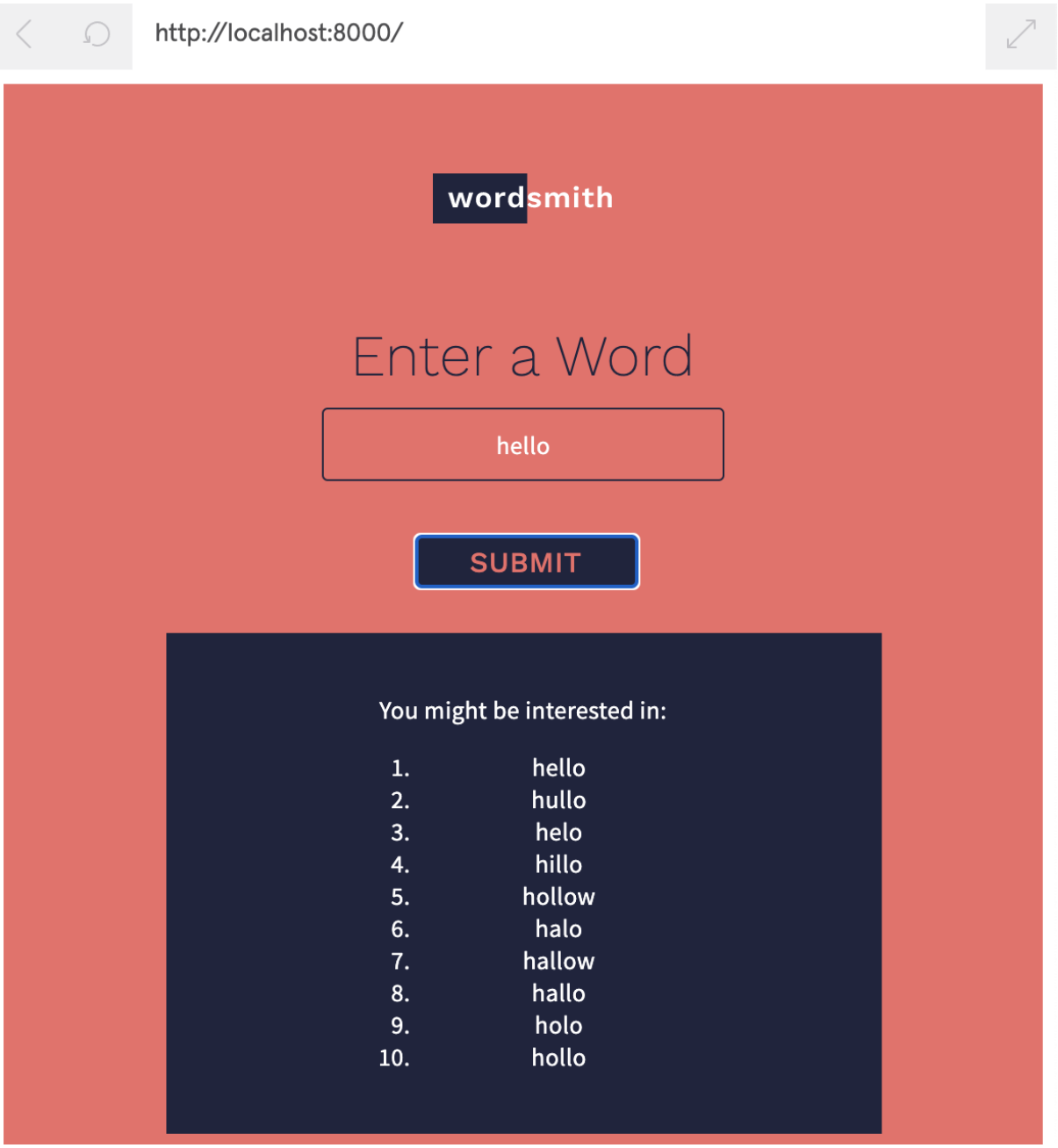
- **fetch( ) GET Requests -4-**

이전 연습에서, 우리는 fetch( ) 함수를 부르고 query URL 과 settings object 를 전달하는 query URL을 생성했다. 그다음, 우리는 then( ) 을 사용하여 체이닝을 하고 두 가지 함수(response, networkError)를 인자로써 전달했다. 즉, 만약 fetch( ) 가 resolve 되면 프로미스를 다루는 함수하나, reject 되면 실행되는 함수하나 이렇게..

이번 연습에서는 프로미스 리턴하는 정보를 가질 것이며, 그 웹사이트 조작할 것이다.



helper함수의 renderRawResponse 를 실행한 모습



helper함수의 renderResponse 함수를 실행한 것

```
// Information to reach API
const url = 'https://api.datamuse.com/words';
const queryParams = '?sl=';

// Selects page elements
const inputField = document.querySelector('#input');
const submit = document.querySelector('#submit');
const responseField = document.querySelector('#responseField');

// AJAX function
const getSuggestions = () => {
  const wordQuery = inputField.value;
  const endpoint = `${url}${queryParams}${wordQuery}`;

  fetch(endpoint, {cache: 'no-cache'}).then(response => {
    if (response.ok) {
      return response.json();
    }
```

```
    throw new Error('Request failed!');
  }, networkError => {
    console.log(networkError.message)
  }).then(jsonResponse => {   // 이번 연습에서 다룬 부분 then 을 추가하여 응답을 렌더링하는 과정을 배웠다.
    renderResponse(jsonResponse);  // renderRawResponse(jsonResponse) 와 체인지
  })
}

// Clears previous results and display results to webpage
const displaySuggestions = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild);
  }
  getSuggestions();
};

submit.addEventListener('click', displaySuggestions);
```

- **fetch( ) POST Requests -1-**

이전 연습에서는 fetch API 와 프로미스를 사용하여 GET 요청을 성공적으로 작성하고 Datamuse 로부터 단어를 얻어냈다.

지금부터, 우리는 fetch( ) 를 사용하여 POST 요청에 대해 배웟 것이다.

다음 아래 다이어그램이 POST 요청을 하는 과정이다.

- **fetch( ) POST Requests -2-**

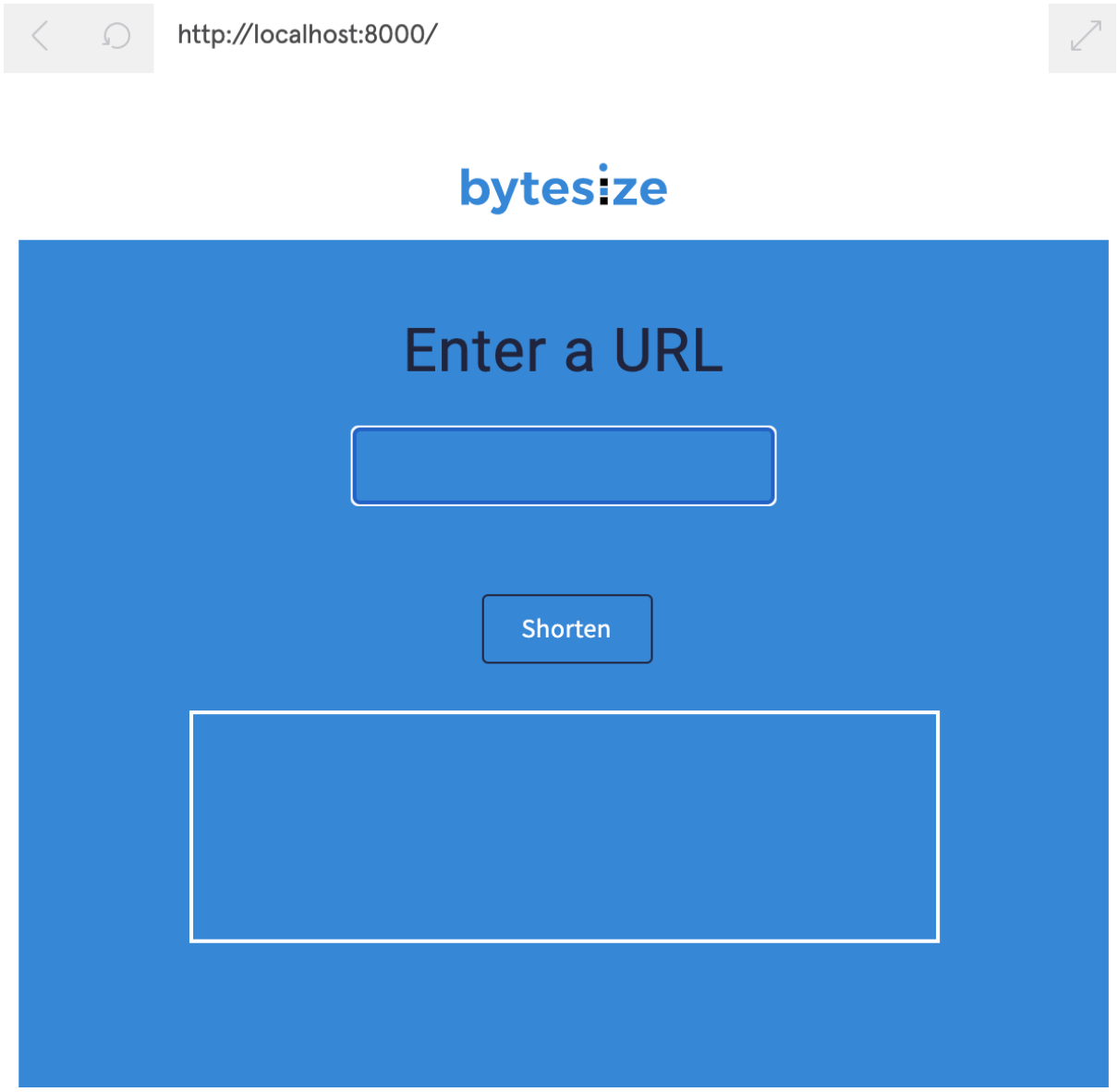
위 다이어그램을 직접 작성해보자.

```
// fetch()의 두번째 인자{} 가 settings object 이다.
// 이 내부에는 요청이 POST 요청이라는 method 와 API로 보내질 정보를 담는다.

fetch('https://api-to-call.com/endpoint', {
  method: 'POST',
  body: JSON.stringify({id: '200'})
}).then(response => {
  if (response.ok) {
    return response.json();
  }
  throw new Error('Request failed!');
}, networkError => {
    console.log(networkError.message);
}).then(jsonResponse => {
  return jsonResponse
});
```

- **fetch( ) POST Requests -3-**

이전 연습에서 POST 요청을 생성하기 위해 fetch( )와 then( ) 을 사용하여 코드를 작성했다. 이번 연습에서는 Rebrandly URL Shortener API 를 사용하여 URL 의 길이를 줄이기 위한 코드를 업데이트 해볼 것이다.

실습 예제




```
// Information to reach API
const apiKey = 'd9271a7f20ae43df8c70146075e626a4';
const url = 'https://api.rebrandly.com/v1/links';

// Some page elements
const inputField = document.querySelector('#input');
const shortenButton = document.querySelector('#shorten');
const responseField = document.querySelector('#responseField');

// AJAX functions
const shortenUrl = () => {
  const urlToShorten = inputField.value;
  const data = JSON.stringify({destination: urlToShorten});
  fetch(url, {
    method: 'POST',
    headers: {
      'Content-type': 'application/json',
      'apikey': apiKey
    },
    body: data
  });
}

// Clear page and call AJAX functions
const displayShortUrl = (event) => {
```

```
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild)
  }
  shortenUrl();
}

shortenButton.addEventListener('click', displayShortUrl);
```

```
// Manipulates responseField to render a formatted and appropriate message
const renderResponse = (res) => {
  // Displays either message depending on results
  if(res.errors){
    responseField.innerHTML = "<p>Sorry, couldn't format your URL.</p><p>Try again.</p>";
  } else {
    responseField.innerHTML = `<p>Your shortened url is: </p><p> ${res.shortUrl} </p>`;
  }
}

// Manipulates responseField to render an unformatted response
const renderRawResponse = (res) => {
  // Displays either message depending on results
  if(res.errors){
    responseField.innerHTML = "<p>Sorry, couldn't format your URL.</p><p>Try again.</p>";
  } else {
    // Adds line breaks for JSON
    let structuredRes = JSON.stringify(res).replace(/,/g, ", \n");
    structuredRes = `<pre>${structuredRes}</pre>`;
    responseField.innerHTML = `${structuredRes}`;
  }
}

// Renders the JSON that was returned when the Promise from fetch resolves.
const renderJsonResponse = (response) => {
  // Creates an empty object to store the JSON in key-value pairs
  let rawJson = {}
  for(let key in response){
    rawJson[key] = response[key]
  }
  // Converts JSON into a string and adding line breaks to make it easier to read
  rawJson = JSON.stringify(rawJson).replace(/,/g, ", \n")
  // Manipulates responseField to show the returned JSON.
  responseField.innerHTML = `<pre>${rawJson}</pre>`
}
```

- **fetch( ) POST Requests -4-**

이전 연습에서 우리는 POST 요청을 위한 endpoint 와 필요한 정보를 포함하는 객체를 위치시켰다면, 이번 연습에서는 response 를 다룰 것이다.

요청은 resolved or rejected 가 될 프로미스를 리턴한다. 만약 프로미스가 resolve 한다면, ok 상태의 정보를 사용할 수 있다. 코드를 구현해보자!

```
// Information to reach API
const apiKey = 'd9271a7f20ae43df8c70146075e626a4';
const url = 'https://api.rebrandly.com/v1/links';

// Some page elements
const inputField = document.querySelector('#input');
const shortenButton = document.querySelector('#shorten');
const responseField = document.querySelector('#responseField');

// AJAX functions
const shortenUrl = () => {
```

```
  const urlToShorten = inputField.value;
  const data = JSON.stringify({destination: urlToShorten});
  fetch(url, {
    method: 'POST',
    headers: {
      'Content-type': 'application/json',
      'apikey': apiKey
    },
    body: data
  }).then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error('Request failed!');

  }, networkError => {
    console.log(networkError.message);
  });

}

// Clear page and call AJAX functions
const displayShortUrl = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild)
  }
  shortenUrl();
}

shortenButton.addEventListener('click', displayShortUrl);
```

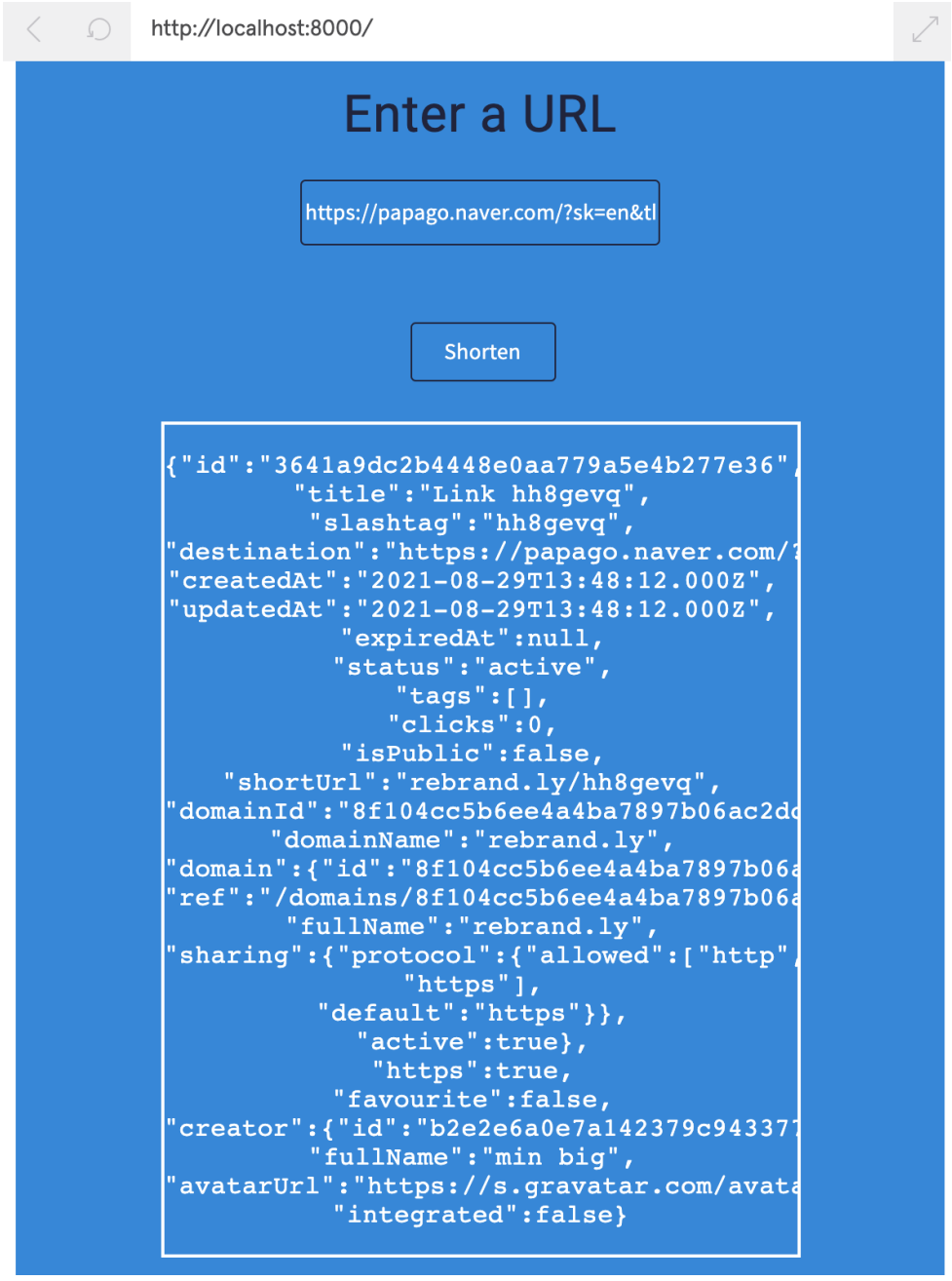
- **fetch( ) POST Requests -5-**

거의 다왔다. 마지막 웹페이지에서 사용가능한 정보를 만들어내는 then( ) 을 마침내 추가해볼 것이다.

renderRawResponse 함수를 사용

```
// Information to reach API
const apiKey = 'd9271a7f20ae43df8c70146075e626a4';
const url = 'https://api.rebrandly.com/v1/links';

// Some page elements
const inputField = document.querySelector('#input');
const shortenButton = document.querySelector('#shorten');
const responseField = document.querySelector('#responseField');

// AJAX functions
const shortenUrl = () => {
  const urlToShorten = inputField.value;
  const data = JSON.stringify({destination: urlToShorten});
  fetch(url, {
    method: 'POST',
    headers: {
      'Content-type': 'application/json',
      'apikey': apiKey
    },
    body: data
  }).then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error('Request failed!');

  }, networkError => {
    console.log(networkError.message);
  }).then(jsonResponse => {
    renderRawResponse(jsonResponse);
  });   // renderRawResponse 를 사용

}
```

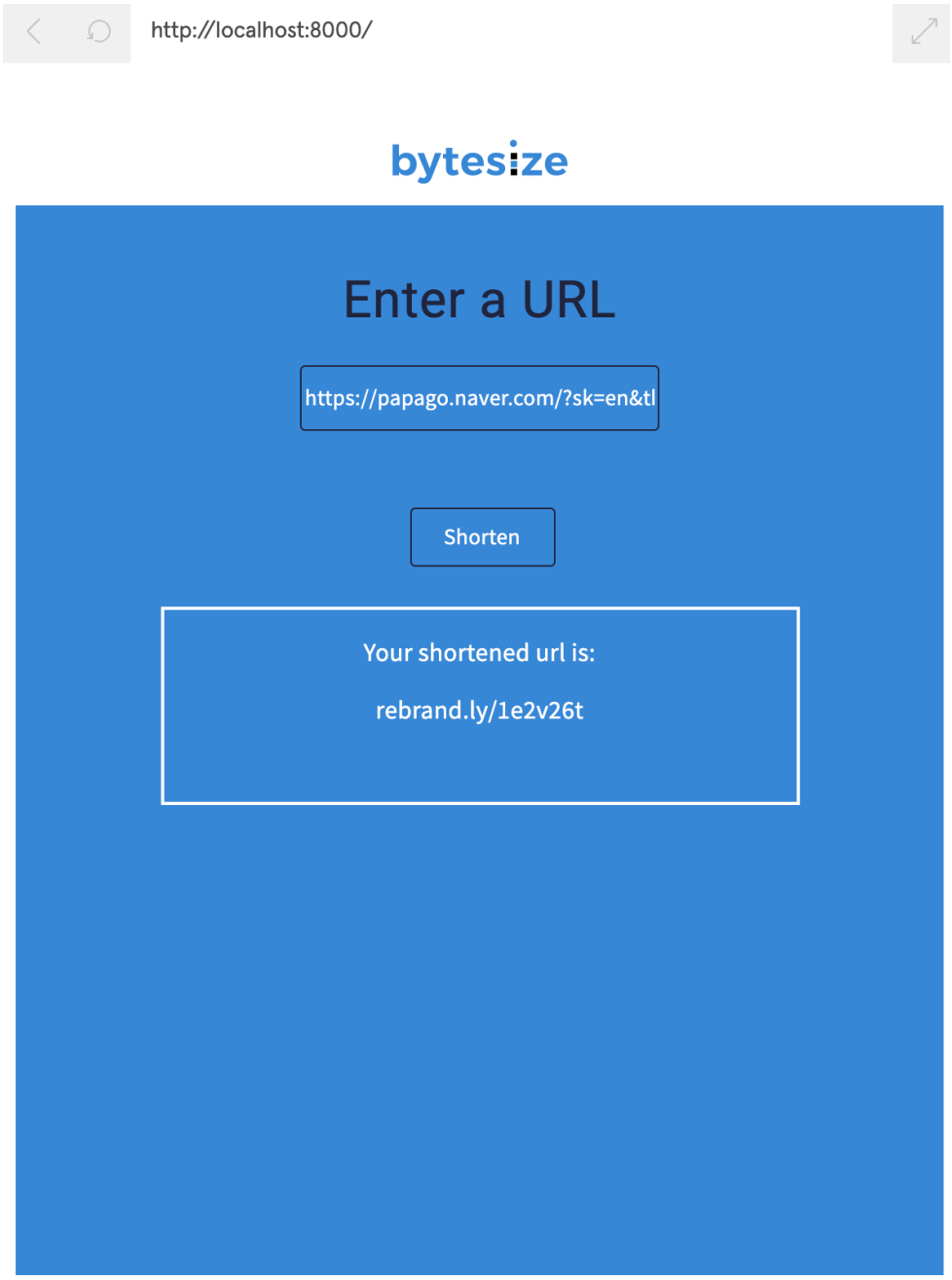
```
// Clear page and call AJAX functions
const displayShortUrl = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild)
  }
  shortenUrl();
}

shortenButton.addEventListener('click', displayShortUrl);
```



raw한 JSON결과를 렌더링해준다.

renderResponse 함수를 사용

```
// Information to reach API
const apiKey = 'd9271a7f20ae43df8c70146075e626a4';
const url = 'https://api.rebrandly.com/v1/links';

// Some page elements
const inputField = document.querySelector('#input');
const shortenButton = document.querySelector('#shorten');
const responseField = document.querySelector('#responseField');

// AJAX functions
const shortenUrl = () => {
  const urlToShorten = inputField.value;
  const data = JSON.stringify({destination: urlToShorten});
  fetch(url, {
    method: 'POST',
    headers: {
      'Content-type': 'application/json',
      'apikey': apiKey
    },
    body: data
  }).then(response => {
    if (response.ok) {
      return response.json();
    }
    throw new Error('Request failed!');

  }, networkError => {
    console.log(networkError.message);
  }).then(jsonResponse => {
    renderResponse(jsonResponse);
  });

}

// Clear page and call AJAX functions
const displayShortUrl = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild)
  }
  shortenUrl();
}

shortenButton.addEventListener('click', displayShortUrl);
```



링크가 잘 줄여서 출력되었다.

이렇게 fetch( )와 promise 를 이용하여 POST 로 API 서버에 정보를 보내주고 다시 응답을 받는 URL 줄이기 API 를 사용해봤다!!

- **async GET Requests -1-**

지금까지 우리가 해온 것의 순서를 정리해보자.

- GET, POST 요청을 만들기 위해 fetch( )를 사용한다.
- 돌아온 response의 상태를 확인한다.

- 일어날 수 있는 에러를 캐치한다.
- 성공적인 response 를 가지고 웹 사이트에 렌더링한다.

환상적이다. 이것은 인터넷이 작동하는 기초방식이다.

다음 연습에서는 async await 를 사용하여 체이닝 프로미스를 더욱더 심플하게 만드는 법을 배울 것이다. 지금까지 작성한 읽기 어렵고 이해하기 어려운 코드를 더 간단하게 리팩토링 해보자.

다음 아래는 async 를 이용한 GET 요청의 기본예시이다.

- **async GET Requests -2-**

위 예시를 직접 작성해보자.

```
const getData = async () => {   // async Function getData() { } 도 가능하다.
  try {
    const response = await fetch('https://api-to-call.com/endpoint');
    if(response.ok) {
      const jsonResponse = await response.json();
      return jsonResponse;
    }
    throw new Error('Request failed!');
  } catch(error) {
    console.log(error);
  }
};
```

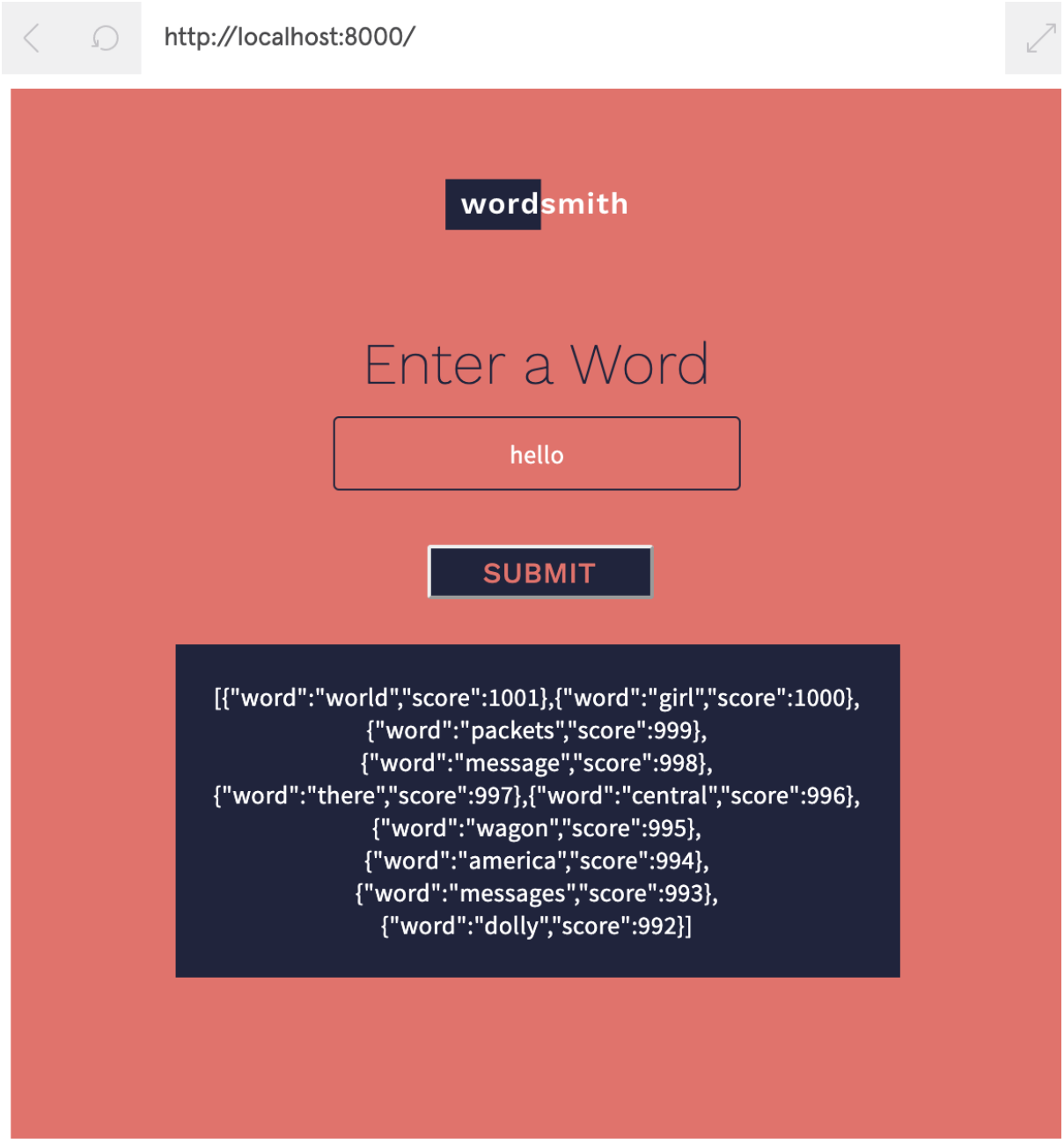
- **async GET Requests -3-**

이번엔 Datamuse API 를 사용하기 위해 코드를 업데이트해볼 것이다.

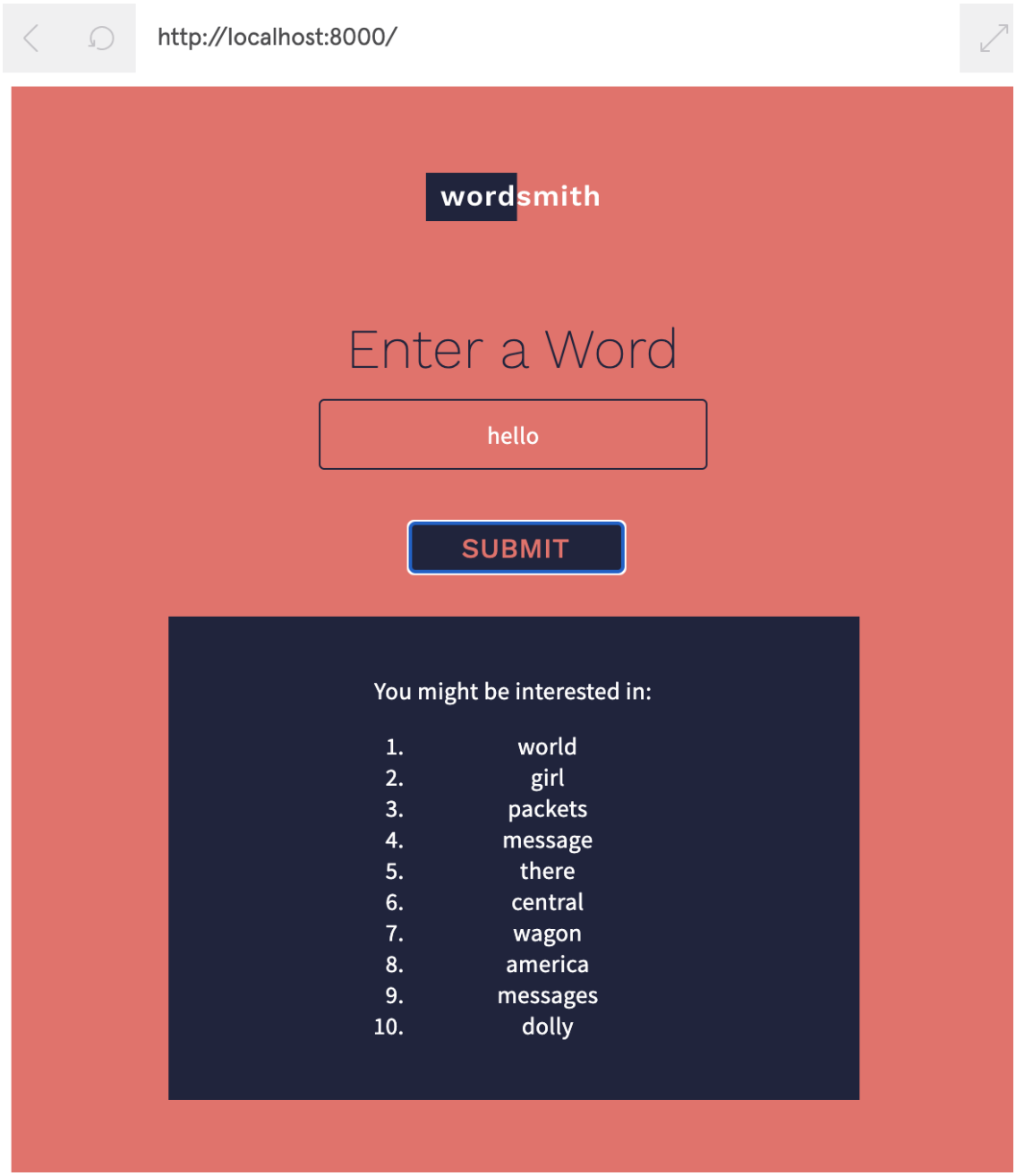
```
// Information to reach API
const url = 'https://api.datamuse.com/words?';
const queryParams = 'rel_jja=';

// Selecting page elements
const inputField = document.querySelector('#input');
const submit = document.querySelector('#submit');
const responseField = document.querySelector('#responseField');

// AJAX function async await 이용
// Code goes here
const getSuggestions = async () => {
   const wordQuery = inputField.value;
   const endpoint = `${url}${queryParams}${wordQuery}`;

   try{
      const response = await fetch(endpoint, {cache: 'no-cache'});
      if (response.ok) {
        const jsonResponse = await response.json();
        renderRawResponse(jsonResponse);
      }
   } catch(error) {
     console.log(error);
   }
};


// Clear previous results and display results to webpage
const displaySuggestions = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild)
  }
  getSuggestions();
}

submit.addEventListener('click', displaySuggestions);
```

```
/ Formats response to look presentable on webpage
const renderResponse = (res) => {
  // Handles if res is falsey
  if(!res){
    console.log(res.status);
  }
  // In case res comes back as a blank array
  if(!res.length){
    responseField.innerHTML = "<p>Try again!</p><p>There were no suggestions found!</p>";
    return;
  }

  // Creates an empty array to contain the HTML strings
  let wordList = [];
  // Loops through the response and caps off at 10
  for(let i = 0; i < Math.min(res.length, 10); i++){
    // creating a list of words
    wordList.push(`<li>${res[i].word}</li>`);
  }
  // Joins the array of HTML strings into one string
  wordList = wordList.join("");

  // Manipulates responseField to render the modified response
  responseField.innerHTML = `<p>You might be interested in:</p><ol>${wordList}</ol>`;
```

```
  return
}

// Renders response before it is modified
const renderRawResponse = (res) => {
  // Takes the first 10 words from res
  let trimmedResponse = res.slice(0, 10);
  // Manipulates responseField to render the unformatted response
  responseField.innerHTML = `<text>${JSON.stringify(trimmedResponse)}</text>`;
}

// Renders the JSON that was returned when the Promise from fetch resolves.
const renderJsonResponse = (res) => {
  // Creates an empty object to store the JSON in key-value pairs
  let rawJson = {};
  for(let key in response){
    rawJson[key] = response[key];
  }
  // Converts JSON into a string and adding line breaks to make it easier to read
  rawJson = JSON.stringify(rawJson).replace(/,/g, ", \n");
  // Manipulates responseField to show the returned JSON.
  responseField.innerHTML = `<pre>${rawJson}</pre>`;
}
```



renderRawResponse 함수 사용



renderResponse 함수 사용

- **async POST Requests -1-**

이제 async 를 이용한 POST 요청을 배워보자.

- **async POST Requests -2-**

직접 작성해보자.

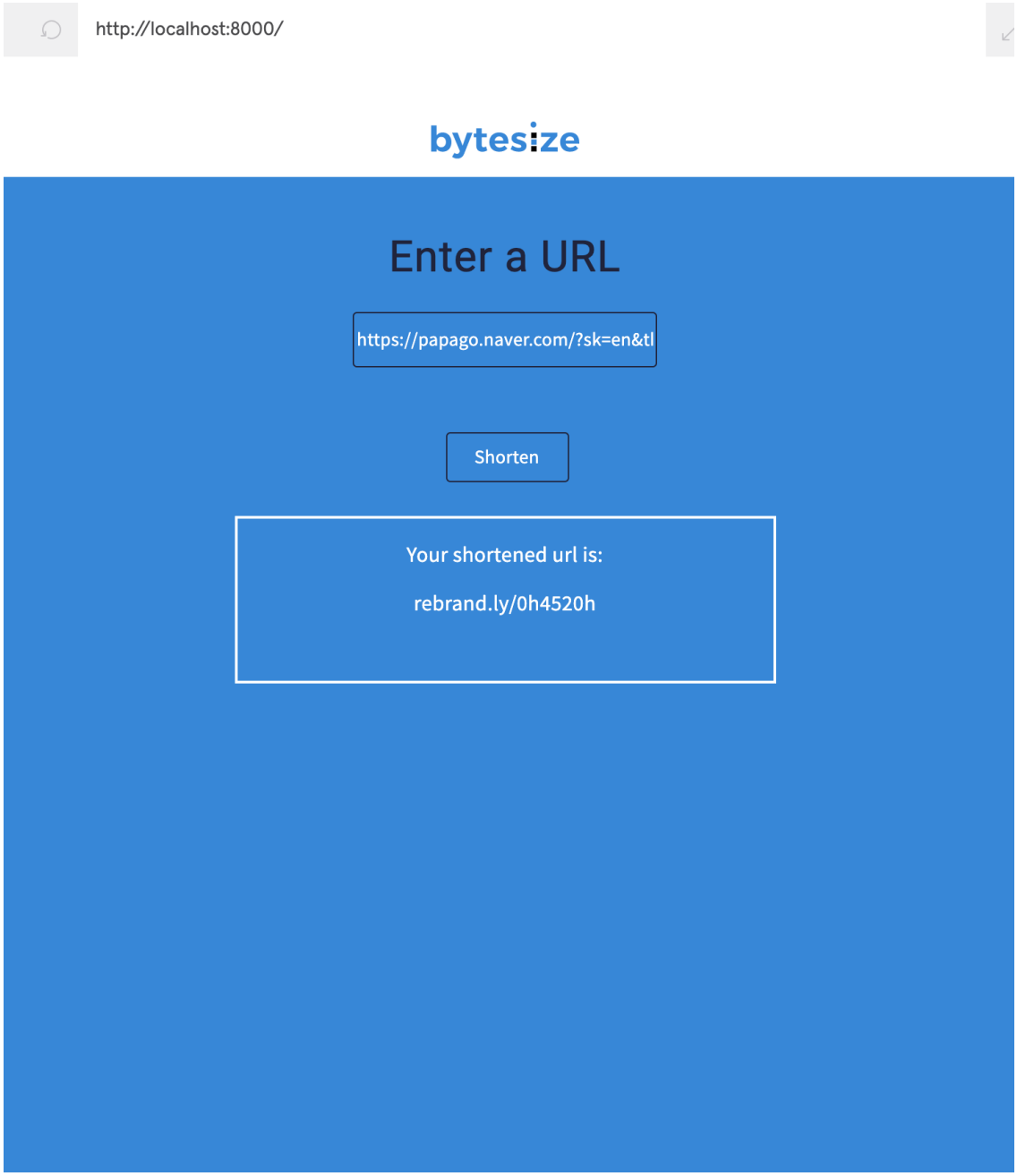
```
const getData = async () => {
  try{
    const response = await fetch('https://api-to-call.com/endpoint',{
      method: 'POST',
      body: JSON.stringify({id: 200})
    });
    if (response.ok) {
      const jsonResponse = await response.json();
      return jsonResponse;
    }
    throw new Error('Request failed!');
  } catch(error) {
    console.log(error);
  }
};
```

- **async POST Requests -3-**

다음은 코드를 업데이트하여 Rebrandly API 를 직접 사용해보자.

```
// information to reach API
const apiKey = 'd9271a7f20ae43df8c70146075e626a4';
const url = 'https://api.rebrandly.com/v1/links';

// Some page elements
const inputField = document.querySelector('#input');
const shortenButton = document.querySelector('#shorten');
const responseField = document.querySelector('#responseField');

// AJAX functions
// Code goes here
const shortenUrl = async () => {
  const urlToShorten = inputField.value;
  const data = JSON.stringify({destination: urlToShorten});

  try {
    const response = await fetch(url, {
      method: 'POST',
      body: data,
      headers: {
        'Content-type': 'application/json',
        'apikey': apiKey
      }
    });
    if (response.ok) {
      const jsonResponse = await response.json();
      renderResponse(jsonResponse);
    }
    throw new Error('Request failed!');
  } catch(error) {
    console.log(error);
  }
};


// Clear page and call AJAX functions
const displayShortUrl = (event) => {
  event.preventDefault();
  while(responseField.firstChild){
    responseField.removeChild(responseField.firstChild);
  }
  shortenUrl();
}

shortenButton.addEventListener('click', displayShortUrl);
```

```
// Manipulates responseField to render a formatted and appropriate message
const renderResponse = (res) => {
  // Displays either message depending on results
  if(res.errors){
    responseField.innerHTML = "<p>Sorry, couldn't format your URL.</p><p>Try again.</p>";
  } else {
    responseField.innerHTML = `<p>Your shortened url is: </p><p> ${res.shortUrl} </p>`;
  }
}

// Manipulates responseField to render an unformatted response
const renderRawResponse = (res) => {
  // Displays either message depending on results
  if(res.errors){
    responseField.innerHTML = "<p>Sorry, couldn't format your URL.</p><p>Try again.</p>";
  } else {
    // Adds line breaks for JSON
    let structuredRes = JSON.stringify(res).replace(/,/g, ", \n");
    structuredRes = `<pre>${structuredRes}</pre>`;
    responseField.innerHTML = `${structuredRes}`;
  }
}
```

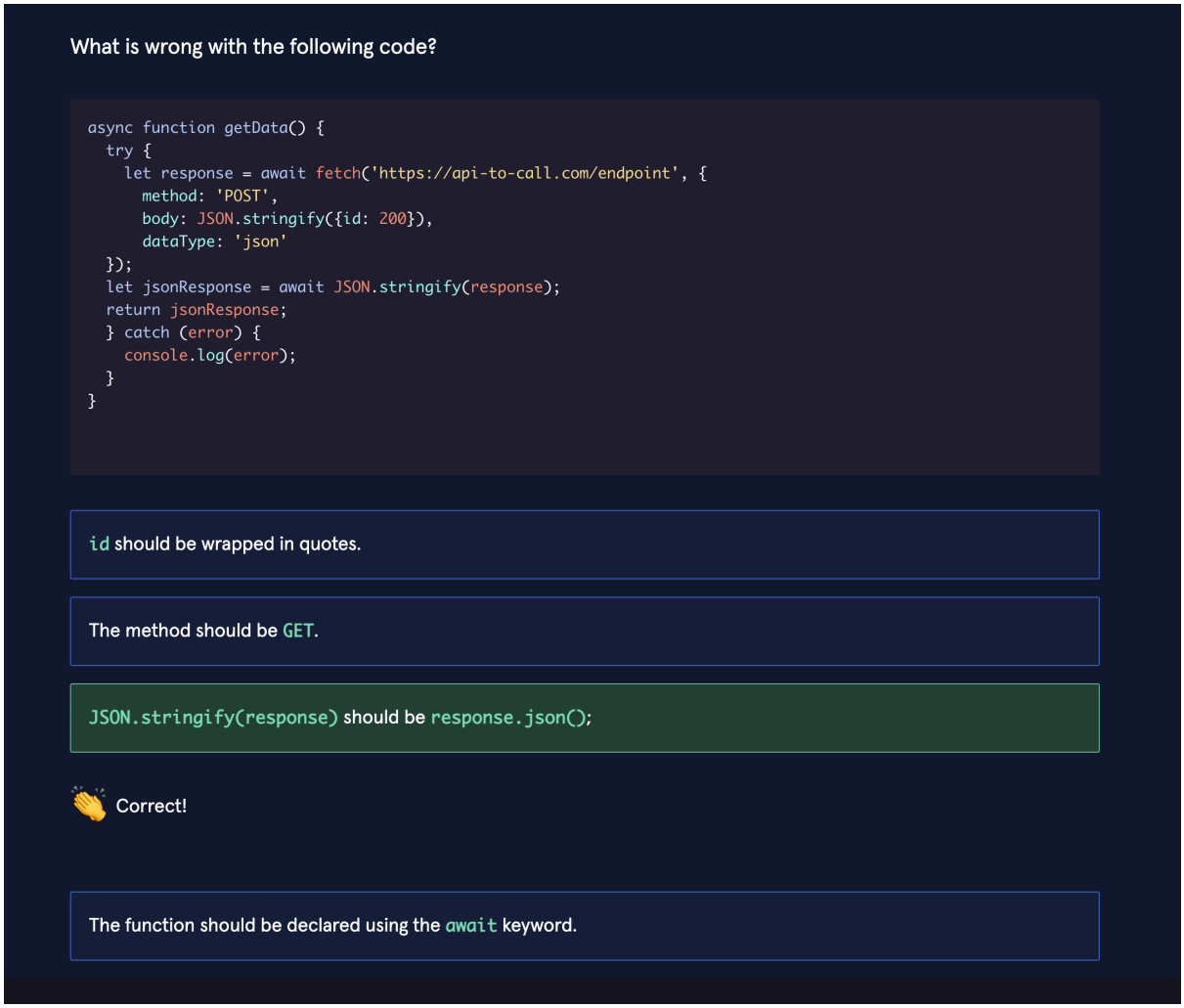


- **Review**

- GET 과 POST 요청은 다양한 방식으로 생성될 수 있다.
- API 로부터 비동기적으로 데이터를 요청하기 위해 AJAX 를 사용해라. fetch( ) 와 async await 는 ES6(promise), ES8에서 각각 개발된 새로운 기능이다.
- 프로미스는 하나의 요청으로부터 마침내 리턴될 데이터를 대표하는 자바스트립트의 객체의 새로운 타입이다.
- fetch( ) 는 요청을 생성할 때 사용되는 web API 이다. 이는 프로미스를 리턴한다.

- fetch( )에 의해 리턴된 프로미스를 다루기 위해 then( ) 메서드를 체이닝할 수 있다.
- .json( ) 메서드는 리턴된 하나의 프로미스를 하나의 JSON 객체로 변환해준다.
- async 는 프로미스를 리턴하는 함수를 생성하기 위해 사용되는 키워드이다.
- await 는 프로미스가 resolve 하는 동안 메세지 큐를 통해 계속해서 움직이라고 프로그램에게 말해주기 위해 사용된다.
- await 은 async 로 선언된 함수안에서만 사용된다.

퀴즈



response 는 프로미스를 리턴하기 때문에 .json( ) 메서드를 사용해야한다. JSON.stringify( ) 는 파라미터로 js객체만 받는다.

What is wrong with the following code?

```
async function getData() {
  try {
    let response = await fetch('https://api-to-call.com/endpoint', 'POST', JSON.stringify({id: 200}),
'json');
    let jsonResponse = await response.json();
    return jsonResponse;
  } catch (error) {
    console.log(error);
  }
}
```

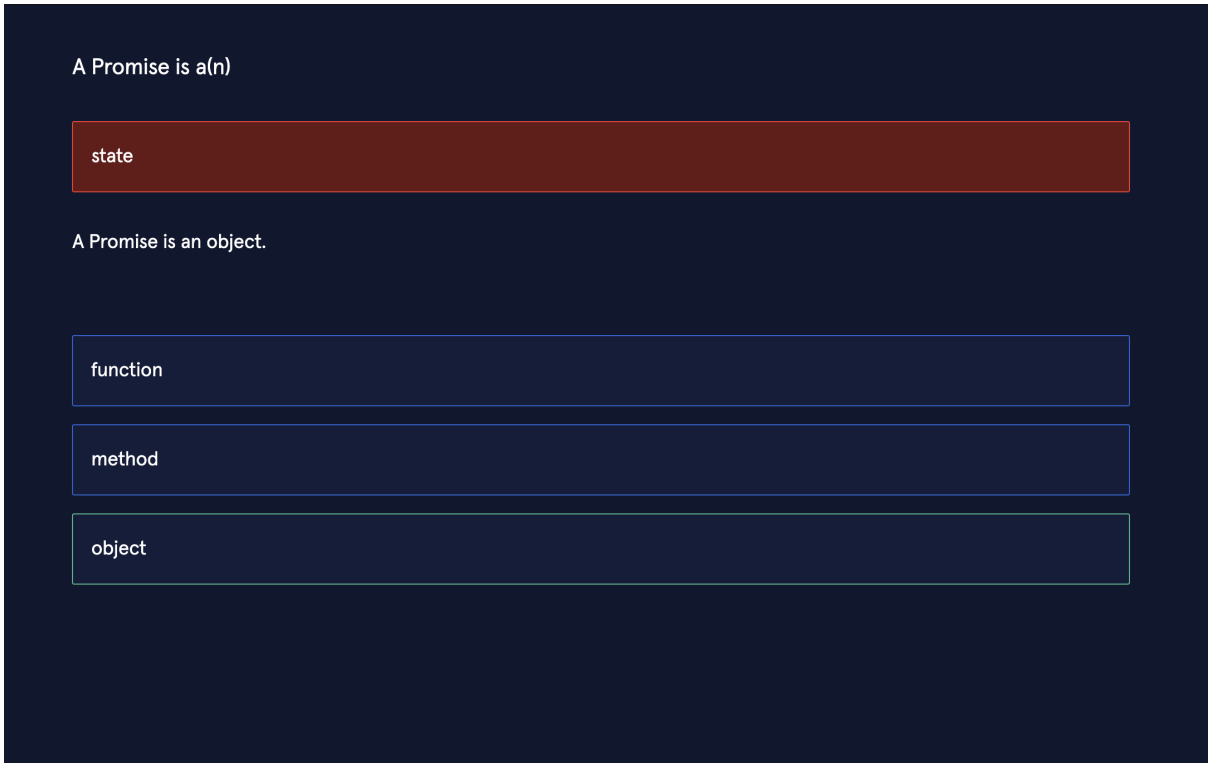
The settings need to be inside of an object and part of property/value pairs.

The method should be `GET`.

Both `GET` and `POST` requests are acceptable.

The function should be declared using the `await` keyword.

`id` should be wrapped in quotes.



Article 1. Fetching Data from the Server

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data

Article 2. Using Fetch

https://developer.mozilla.org/ko/docs/Web/API/Fetch_API/Using_Fetch

Article 3. Third-Party APIs

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Third_party_APIs