GO FETCH

code class, 23 June 2017

The fetch API is an interface to **request and handle (remote) resources**.

Fetch API

```
fetch( '/endpoint/` )
  .then( response => … )
  .catch( err => … )
```

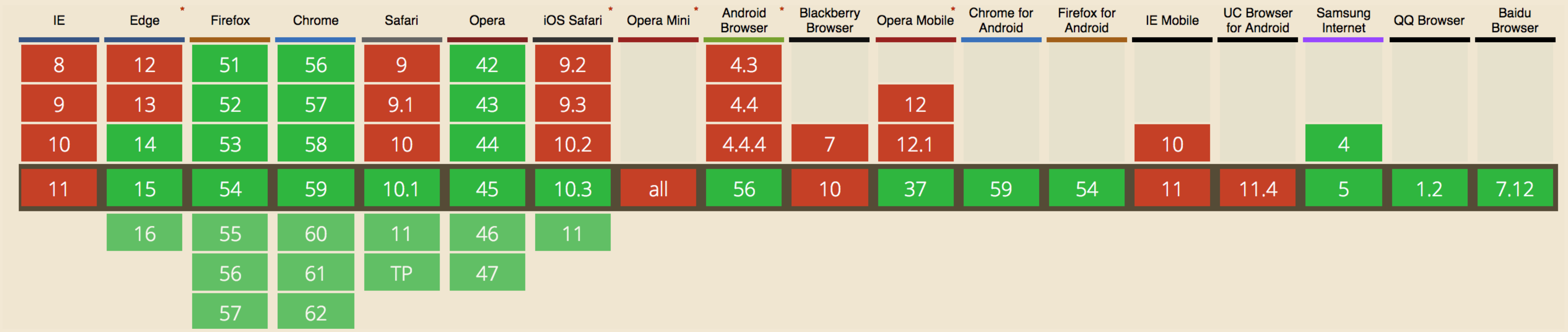
succeeds XMLHttpRequest

```
const xhr = new XMLHttpRequest();

xhr.open('GET', '/endpoint/');
xhr.onload = () => /* use `xhr.response` */;
xhr.onerror = (err) => …;
xhr.send();
```

developer.mozilla.org/en/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

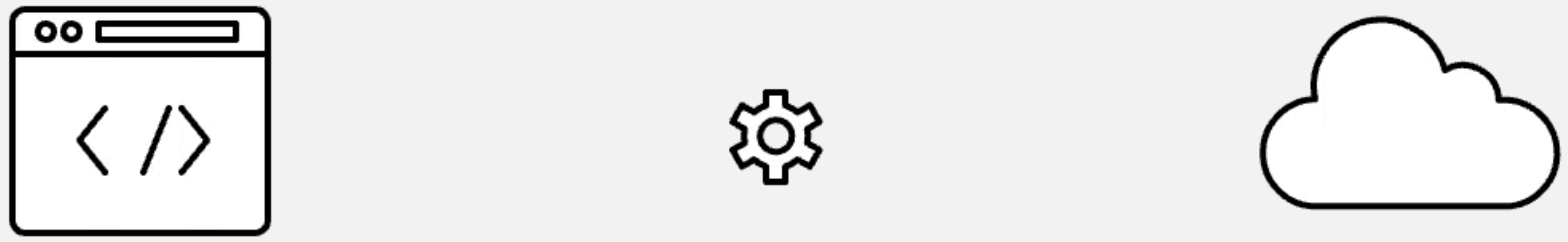
Fetch API support



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE Mobile	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser
8	12	51	56	9	42	9.2		4.3									
9	13	52	57	9.1	43	9.3		4.4		12							
10	14	53	58	10	44	10.2		4.4.4	7	12.1			10		4		
11	15	54	59	10.1	45	10.3	all	56	10	37	59	54	11	11.4	5	1.2	7.12
	16	55	60	11	46	11											
		56	61	TP	47												
		57	62														

caniuse.com/#feat=fetch

Fetch used in Service Worker



Fetch polyfills

- **WHATWG Fetch** replaces subset of Fetch spec based on XMLHttpRequest.

- **Node Fetch** is based on Node’s built-in http module instead of XMLHttpRequest.

- **Isomorphic fetch** exports `node-fetch` for server-side, `whatwg-fetch` for client-side.

EXERCISES

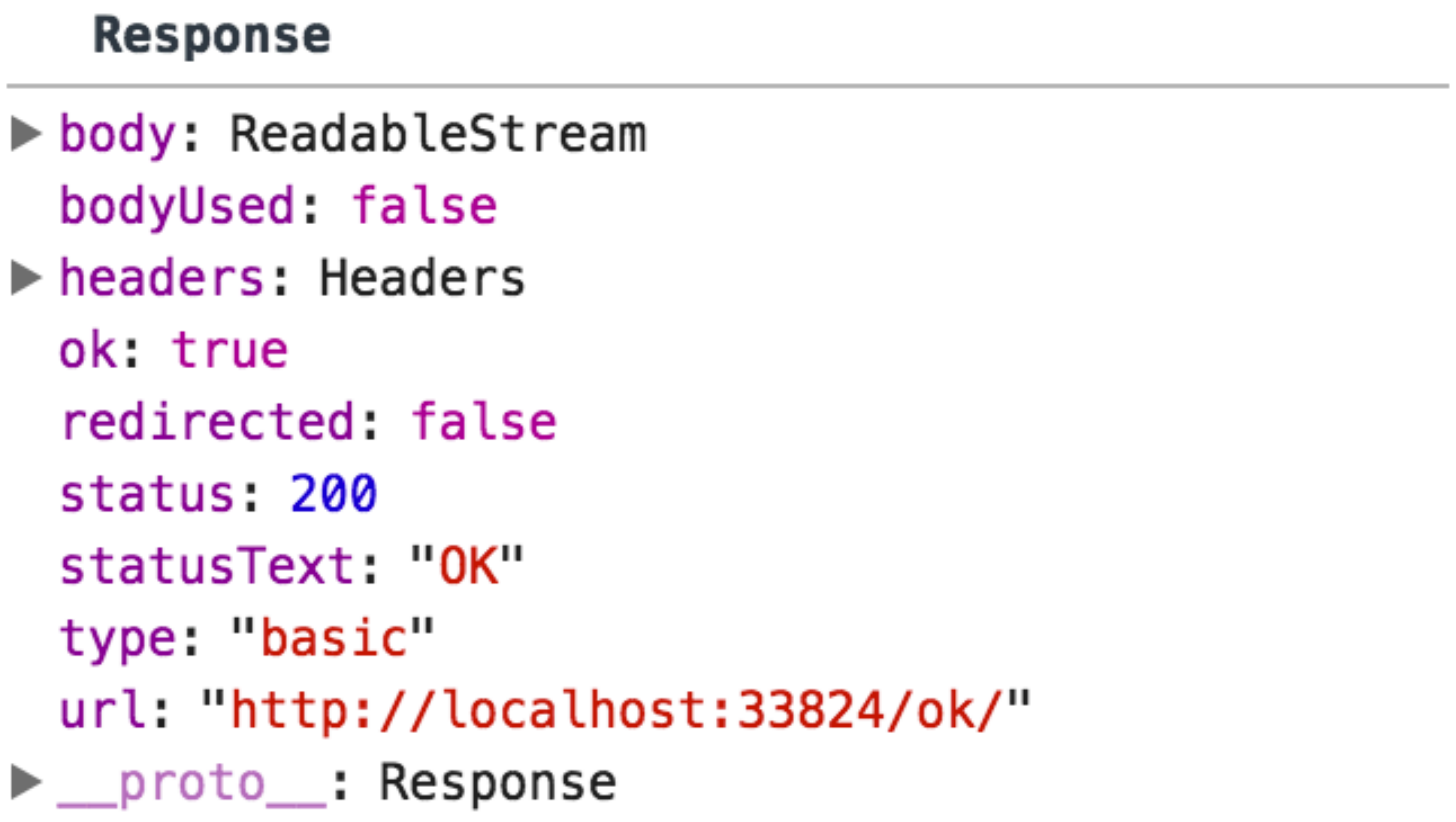
A fetch request **resolves with a Response** object.

Exercise 1: Output response metadata

```
function fetchAndOutput() {
 fetch('/ok/')
   .then(function(response) {
     output(
      '@todo: output status code and content type'
     );
   });
}
```

github.com/voorhoede/code-class-fetch/blob/master/src/exercise-1.html

Exercise 1: Output response metadata



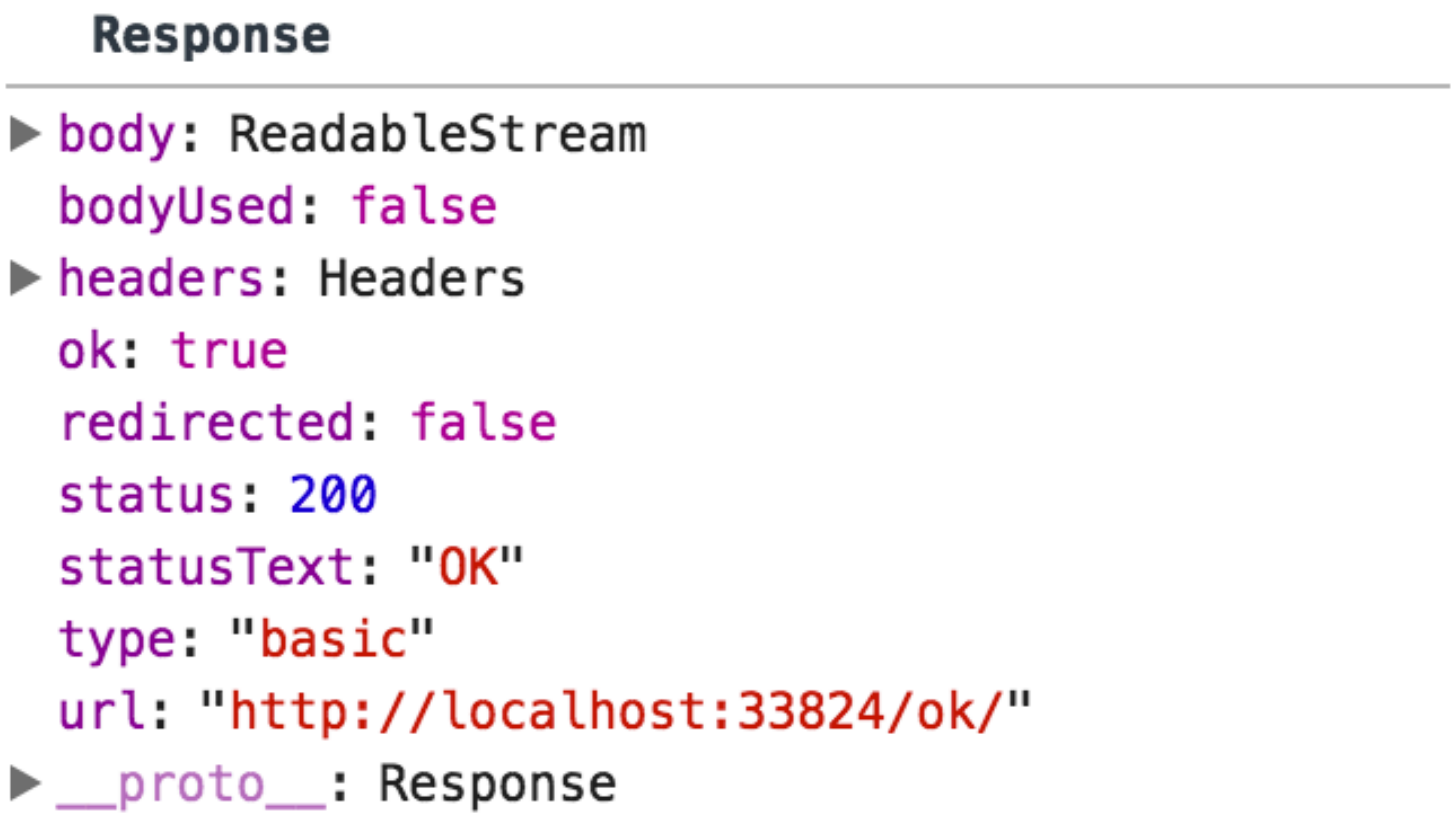
```
Response
▶ body: ReadableStream
  bodyUsed: false
▶ headers: Headers
  ok: true
  redirected: false
  status: 200
  statusText: "OK"
  type: "basic"
  url: "http://localhost:33824/ok/"
▶ __proto__: Response
```

Exercise 1: Output response metadata

```
function fetchAndOutput() {
 fetch('/ok/')
   .then(function(response) {
     output(
       response.status + ' ' +
       response.headers.get('Content-Type')
     );
   });
}
```

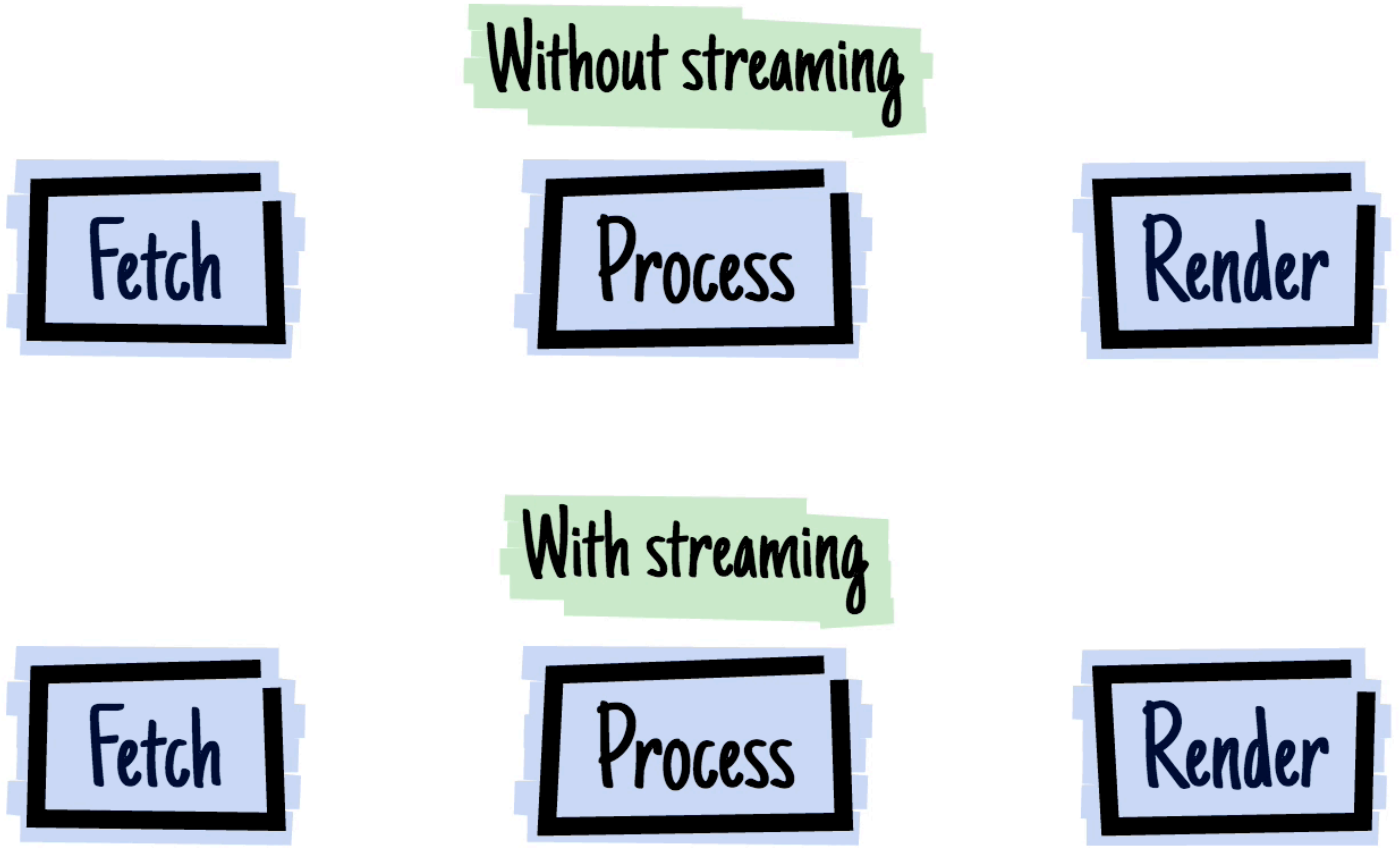
The fetch response body is a **readable stream**.

Body is a readable stream



```
Response
▶ body: ReadableStream
  bodyUsed: false
▶ headers: Headers
  ok: true
  redirected: false
  status: 200
  statusText: "OK"
  type: "basic"
  url: "http://localhost:33824/ok/"
▶ __proto__: Response
```

Body is a readable stream



jakearchibald.com/2016/streams-ftw/

Reading the response body

- **response.text().then(text => …)** reads stream to completion and returns promise that resolves with text.

- **response.json().then(json => …)** reads stream to completion and returns promise that resolves with parsed JSON.

- response.blob(), response.formData(), and many ways to handle streams without waiting for completion.

Exercise 2: Output response body

```
function fetchAndOutput (endpoint) {
 fetch(endpoint)
   .then(function(response) {
     var type = response.headers.get('Content-Type');
     // @todo: Convert body to text or json
     //           depending on content type
    })
    .then(output);
}
```

github.com/voorhoede/code-class-fetch/blob/master/src/exercise-2.html

Exercise 2: Output response body

```
.then(function(response) {
  var type = response.headers.get('Content-Type');
  if (type.startsWith('text/html')) {
    return response.text();
  };
  if (type.startsWith('application/json')) {
    return response.json();
  };
})
```

Fetch accepts

2nd param to configure options

like method, headers and body.

Fetch API parameters on MDN

Exercise 3: Post with Fetch

```
function postForm (form) {
  var content = { a: form.inputA.value, b: form.inputB.value };

  // @todo: use fetch to post content

  fetch('/store/', ... )
    .then(function(response) { return response.json(); })
    .then(output);
}
```

github.com/voorhoede/code-class-fetch/blob/master/src/exercise-3.html

Exercise 3: Post with Fetch

```
function postForm (form) {
 var content = { a: form.inputA.value, b: form.inputB.value };
 fetch('/store/', {
   method: 'post',
   headers: { 'Content-type': 'application/json' },
   body: JSON.stringify(content)
 })
 .then(function(response) { return response.json(); })
 .then(output);
}
```

Any response resolves a fetch request, so **defining success and handling errors is up to you**.

Exercise 4: Handle Fetch errors

```
function fetchAndOutput (endpoint) {
  fetch(endpoint)
  .then(function(response) {
   // @todo: reject "unsuccessful requests"
   //              with error using `statusText`.
   return response.text();
  })
  .then(outputSuccess)
}
```

github.com/voorhoede/code-class-fetch/blob/master/src/exercise-4.html

Exercise 4: Handle Fetch errors

```
.then(function(response) {
  if (response.status >= 200 && response.status < 300) {
    return response.text();
  }
  return Promise.reject(
    new Error(response.statusText));
})
.then(outputSuccess)
.catch(outputFailure);
```

Credentials of a fetch request object can be configured to **`omit`**, **`same-origin`** or **`include`**.

Request credentials on MDN

Exercise 5: Use credentials

```
function fetchAndOutput() {
  // @todo: Include cookies in fetch request
  //          to authenticate.
  fetch('/protected/', { /* @todo: configure */ })
    .then(handleResponse)
    .then(outputSuccess)
    .catch(outputFailure);
}
```

github.com/voorhoede/code-class-fetch/blob/master/src/exercise-5.html

Exercise 5: Use credentials

```
function fetchAndOutput() {

  fetch('/protected/', { credentials: 'include' })
   .then(handleResponse)
    .then(outputSuccess)
    .catch(outputFailure);
}
```

BONUS

Bonus: async form with Fetch

```
const form = document.querySelector('form');

fetch(new Request(form.action, {
  method: form.method,
  headers: {
    'X-Requested-With': 'XMLHttpRequest',
     Accept': 'application/json'
  },
  body: new FormData(form)
})
.then(…)
```

gist.github.com/jbmoelker/4b763e8e0d97021642f3

DE VOORHOEDE

front-end developers