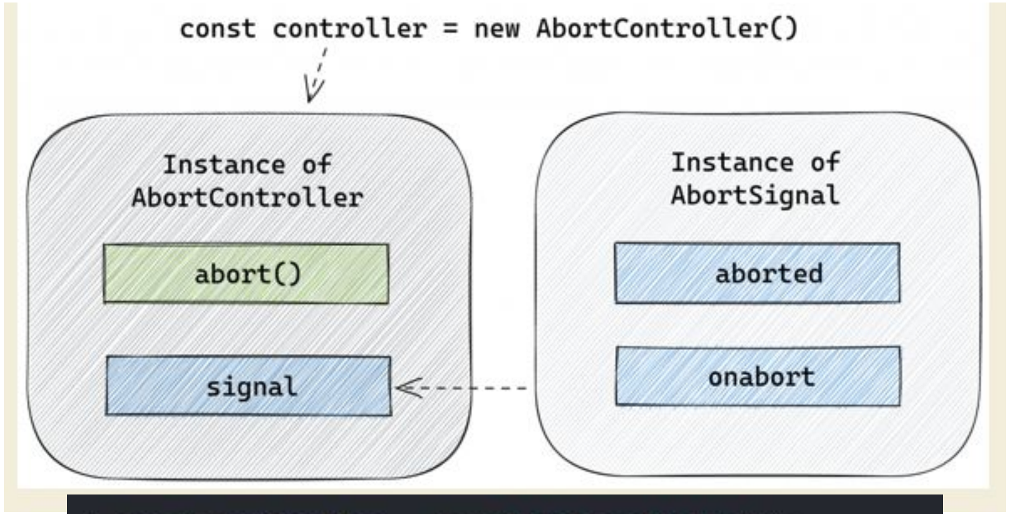
Canceling Pending Async Requests

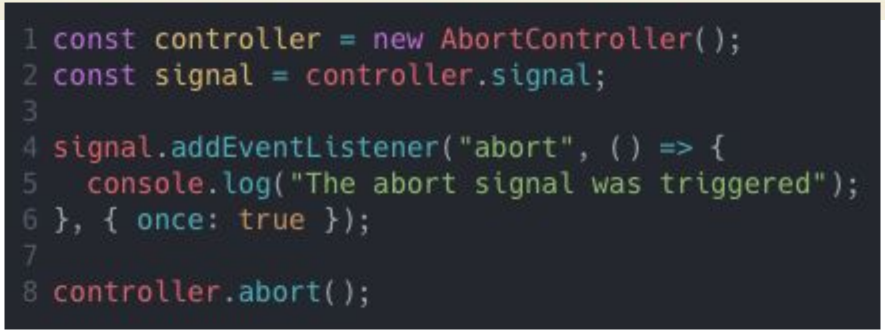
Steven Liou, Alican Sungur

Scenarios for Canceling Async Actions

- After a set amount of time
    - Promise.race()
- When a condition is met
- When an user manually cancels a request
- Cancel multiple async requests or event listeners at once

AbortController Interface


const controller = new AbortController()
Instance of AbortController
abort()
signal
Instance of AbortSignal
aborted
onabort

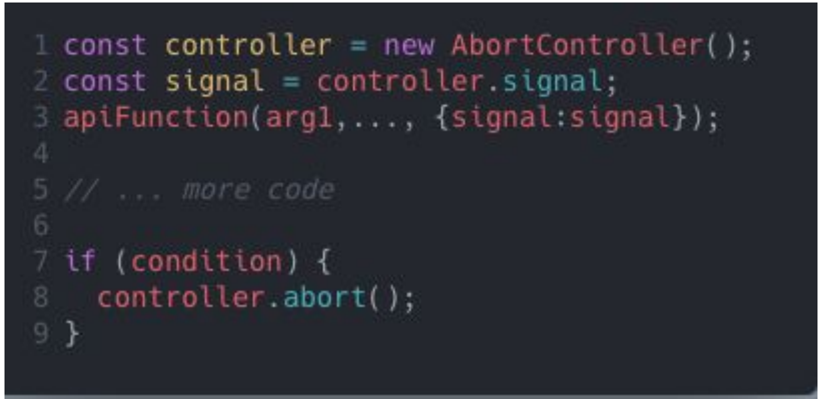


```javascript
const controller = new AbortController();
const signal = controller.signal;

signal.addEventListener("abort", () => {
  console.log("The abort signal was triggered");
}, { once: true });

controller.abort();
```

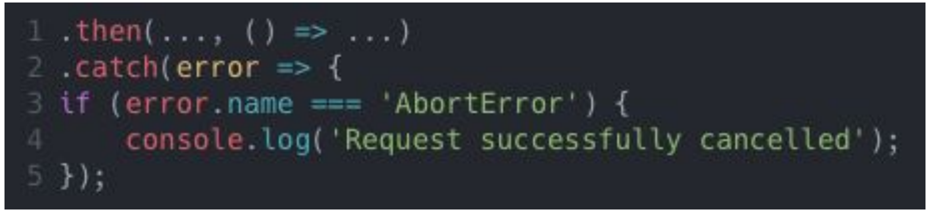
Reference: https://simonplend.com/how-to-cancel-an-http-request-in-node-js/

Using AbortController

1. Instantiate an AbortController object
2. Set up API calls (attaching event listeners or making async requests)
3. Pass the AbortController object's signal object to supported APIs
4. Call AbortController.abort() to cancel attached event listeners or async requests
    a. signal.aborted is true if cancellation succeeds
    b. If there is a pending promise, it is rejected with a typical DOMException error



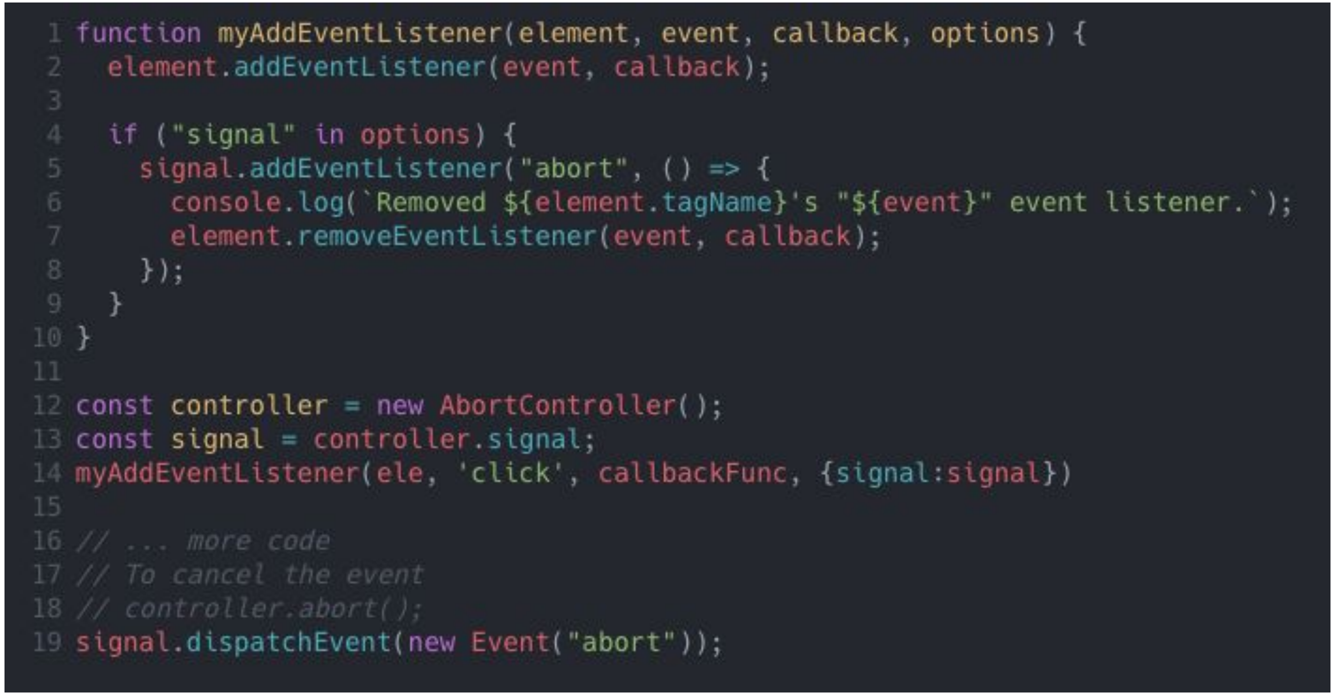
```
const controller = new AbortController();
const signal = controller.signal;
apiFunction(arg1,..., {signal:signal});

// ... more code

if (condition) {
  controller.abort();
}
```



```
.then(..., () => ...)
.catch(error => {
if (error.name === 'AbortError') {
    console.log('Request successfully cancelled');
});
```

How AbortController Interface Works?

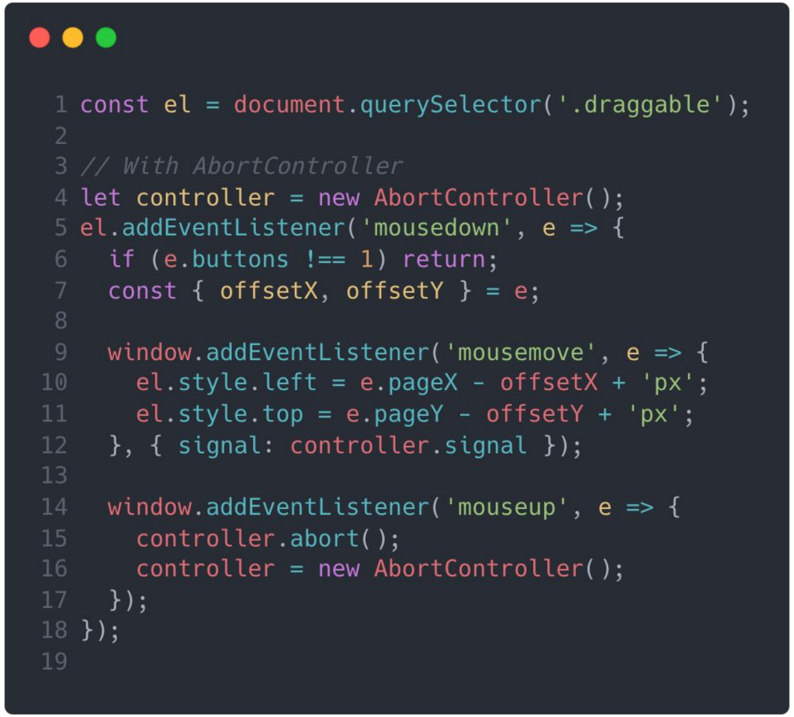
- An event listener is attached on the signal object to listen for the abort event
- AbortController.abort() triggers the abort event on signal object



```javascript
function myAddEventListener(element, event, callback, options) {
  element.addEventListener(event, callback);

  if ("signal" in options) {
    signal.addEventListener("abort", () => {
      console.log(`Removed ${element.tagName}'s "${event}" event listener.`);
      element.removeEventListener(event, callback);
    });
  }
}

const controller = new AbortController();
const signal = controller.signal;
myAddEventListener(ele, 'click', callbackFunc, {signal:signal})

// ... more code
// To cancel the event
// controller.abort();
signal.dispatchEvent(new Event("abort"));
```

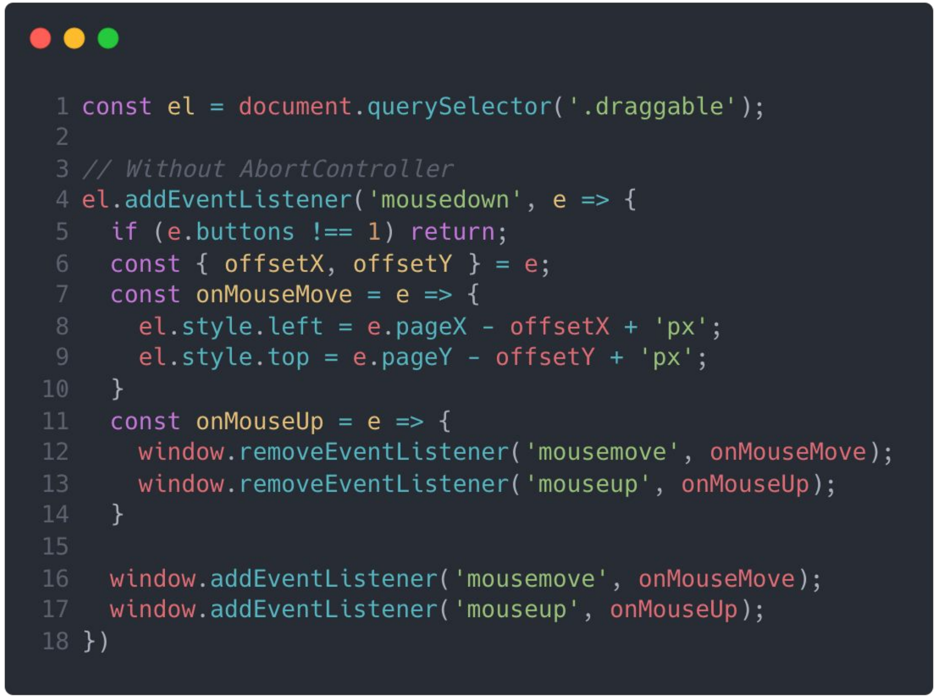
Demo 1: Remove UI event using AbortController

Using AbortController



```javascript
const el = document.querySelector('.draggable');

// With AbortController
let controller = new AbortController();
el.addEventListener('mousedown', e => {
  if (e.buttons !== 1) return;
  const { offsetX, offsetY } = e;

  window.addEventListener('mousemove', e => {
    el.style.left = e.pageX - offsetX + 'px';
    el.style.top = e.pageY - offsetY + 'px';
  }, { signal: controller.signal });

  window.addEventListener('mouseup', e => {
    controller.abort();
    controller = new AbortController();
  });
});

```

Using `removeEventListener()`



```javascript
const el = document.querySelector('.draggable');

// Without AbortController
el.addEventListener('mousedown', e => {
  if (e.buttons !== 1) return;
  const { offsetX, offsetY } = e;
  const onMouseMove = e => {
    el.style.left = e.pageX - offsetX + 'px';
    el.style.top = e.pageY - offsetY + 'px';
  }
  const onMouseUp = e => {
    window.removeEventListener('mousemove', onMouseMove);
    window.removeEventListener('mouseup', onMouseUp);
  }

  window.addEventListener('mousemove', onMouseMove);
  window.addEventListener('mouseup', onMouseUp);
})
```

Based on: *css-tricks.com/using-abortcontroller-as-an-alternative-for-removing-event-listeners/*

Fetch API

Fetch API

- Javascript interface for accessing and manipulating parts of HTTP Processes
- Supported by all modern web browsers (RIP)
- It’s `fetch()` method allows us to fetch resources asynchronously over the network
- It allows us to write code looks much simpler compared to other ways like XHR.

https://developer.mozilla.org/en-US/docs/Web/API/fetch

Fetch API

chucknorris.io is a free JSON API for hand curated Chuck Norris facts. Read more

```
npm install node-fetch -g

npm list -g | grep node-fetch
```

`XMLHttpRequest` vs `fetch()`

Comparison over examples

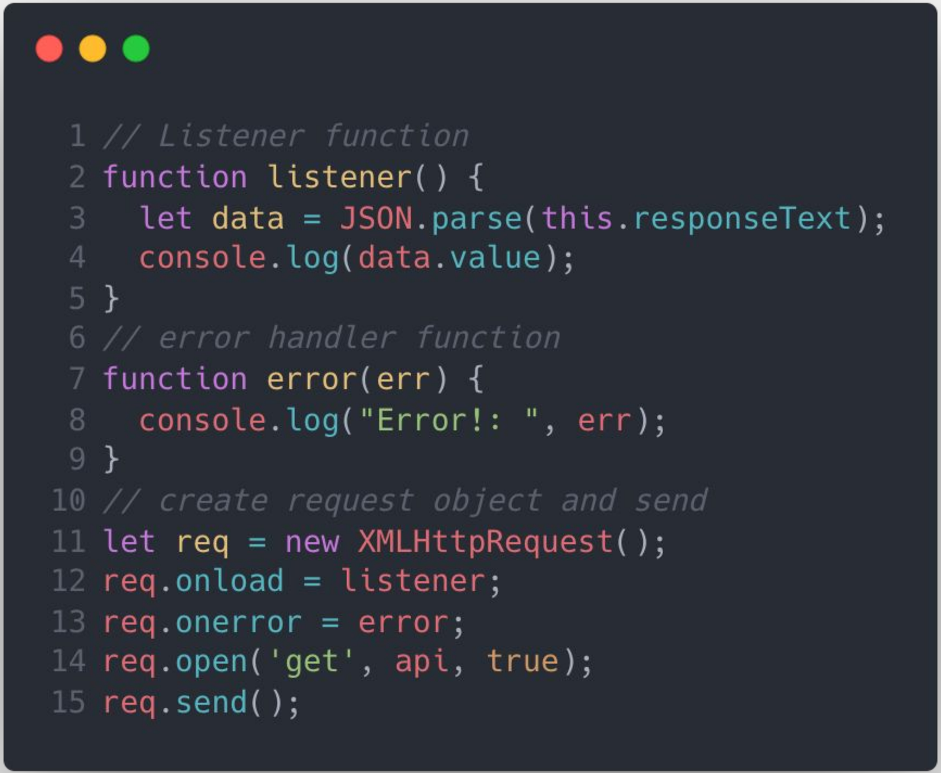
`xhr2` for XHR

`node-fetch` for Fetch API

https://api.chucknorris.io/

Fetch API

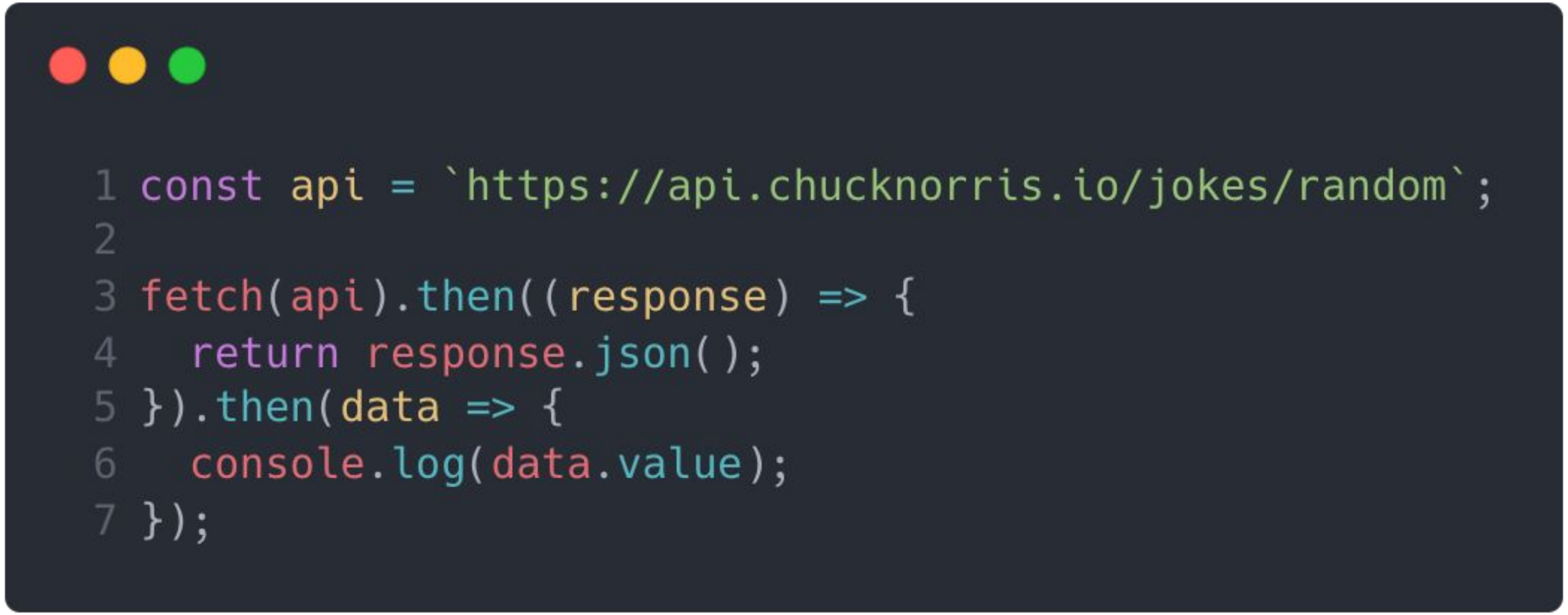
XHR example



```javascript
// Listener function
function listener() {
  let data = JSON.parse(this.responseText);
  console.log(data.value);
}
// error handler function
function error(err) {
  console.log("Error!: ", err);
}
// create request object and send
let req = new XMLHttpRequest();
req.onload = listener;
req.onerror = error;
req.open('get', api, true);
req.send();
```

Fetch API

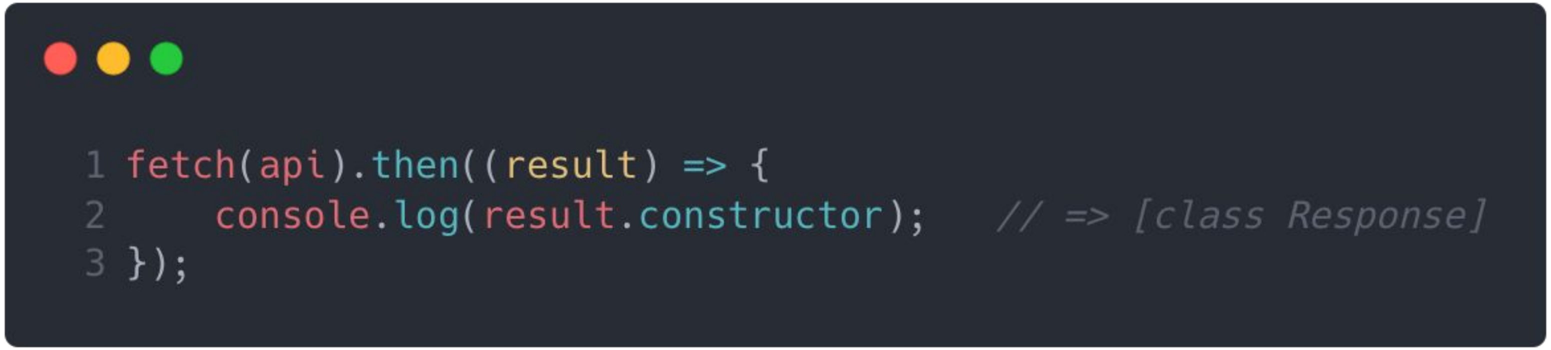
Using `fetch()`



```javascript
const api = `https://api.chucknorris.io/jokes/random`;

fetch(api).then((response) => {
  return response.json();
}).then(data => {
  console.log(data.value);
});
```

Fetch API

Return values



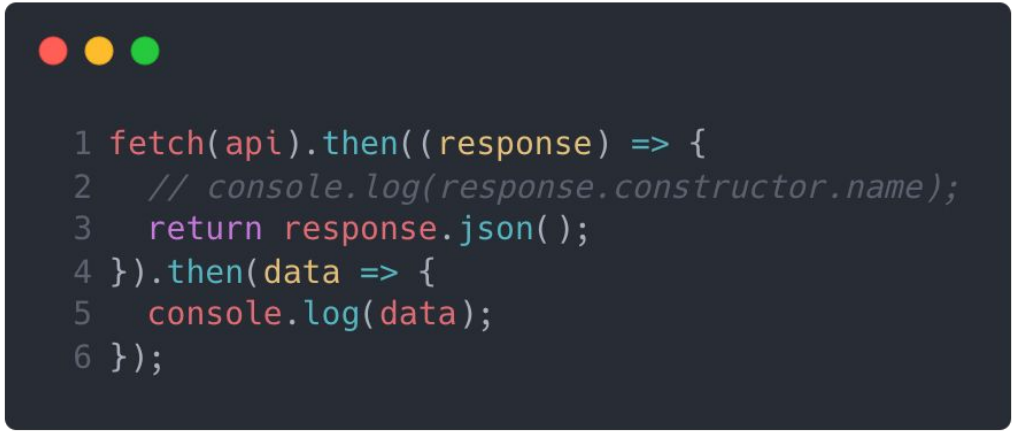
```
fetch(api).then((result) => {
    console.log(result.constructor);    // => [class Response]
});
```

- `fetch()` returns a pending promise
- Promise resolves into a `Response` object when the server responds, similar to onload event for XHR api, Otherwise, it throws an error
- Must check for response status code not in the 200 range, using the response.ok

https://developer.mozilla.org/en-US/docs/Web/API/Response

Fetch API

Parsing response body as JSON



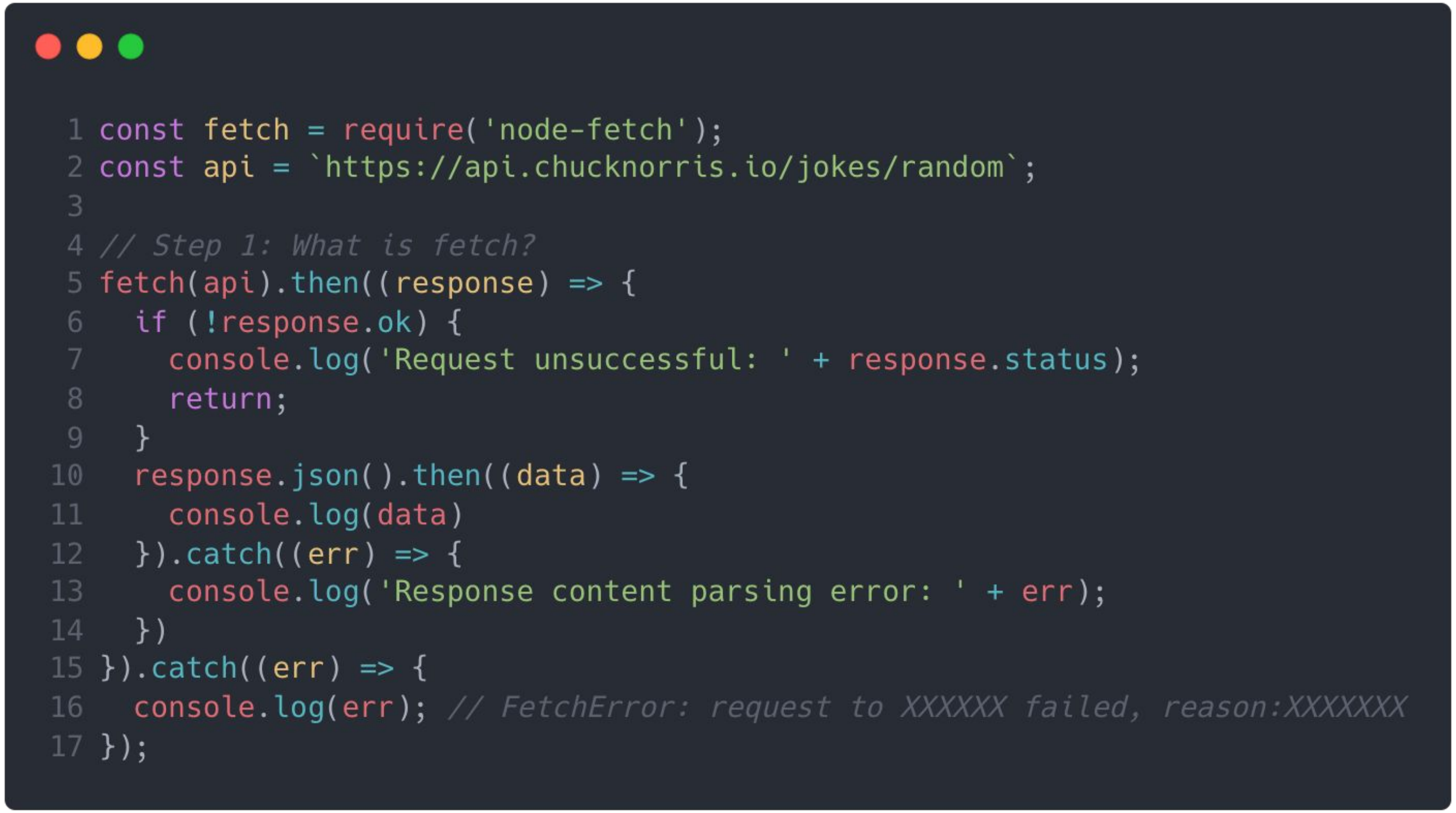
```
fetch(api).then((response) => {
  // console.log(response.constructor.name);
  return response.json();
}).then(data => {
  console.log(data);
});
```

- `json()` returns a pending promise.
- It resolves into response body in JSON format

https://developer.mozilla.org/en-US/docs/Web/API/Response

Fetch API

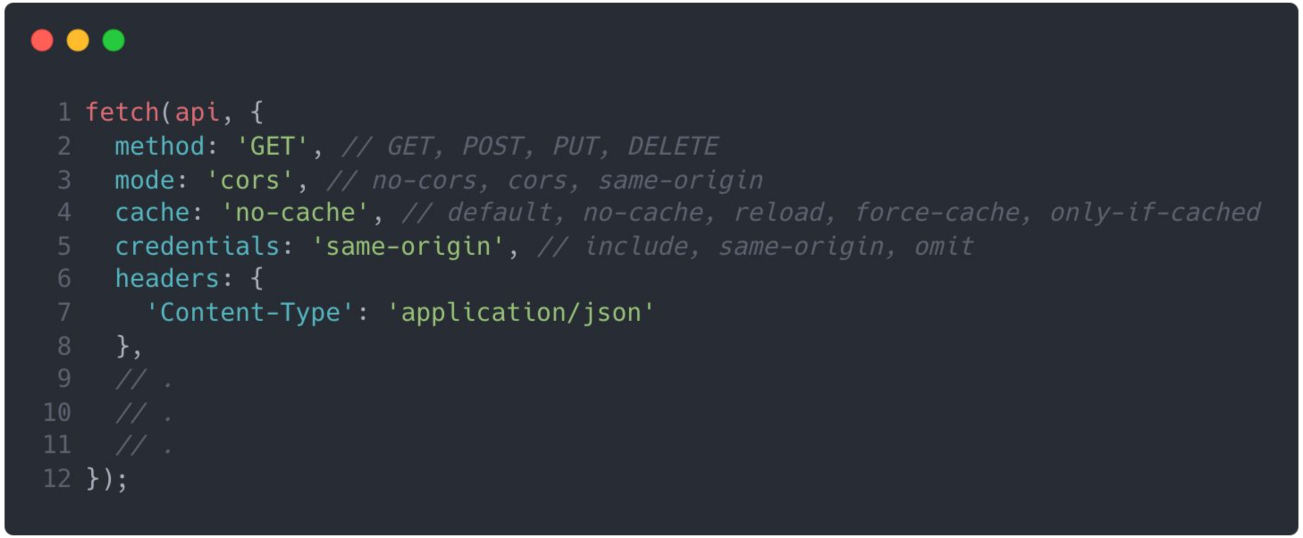
Handling errors and the response object



```javascript
const fetch = require('node-fetch');
const api = `https://api.chucknorris.io/jokes/random`;

// Step 1: What is fetch?
fetch(api).then((response) => {
  if (!response.ok) {
    console.log('Request unsuccessful: ' + response.status);
    return;
  }
  response.json().then((data) => {
    console.log(data)
  }).catch((err) => {
    console.log('Response content parsing error: ' + err);
  })
}).catch((err) => {
  console.log(err); // FetchError: request to XXXXXX failed, reason:XXXXXXX
});
```

Fetch API

Options

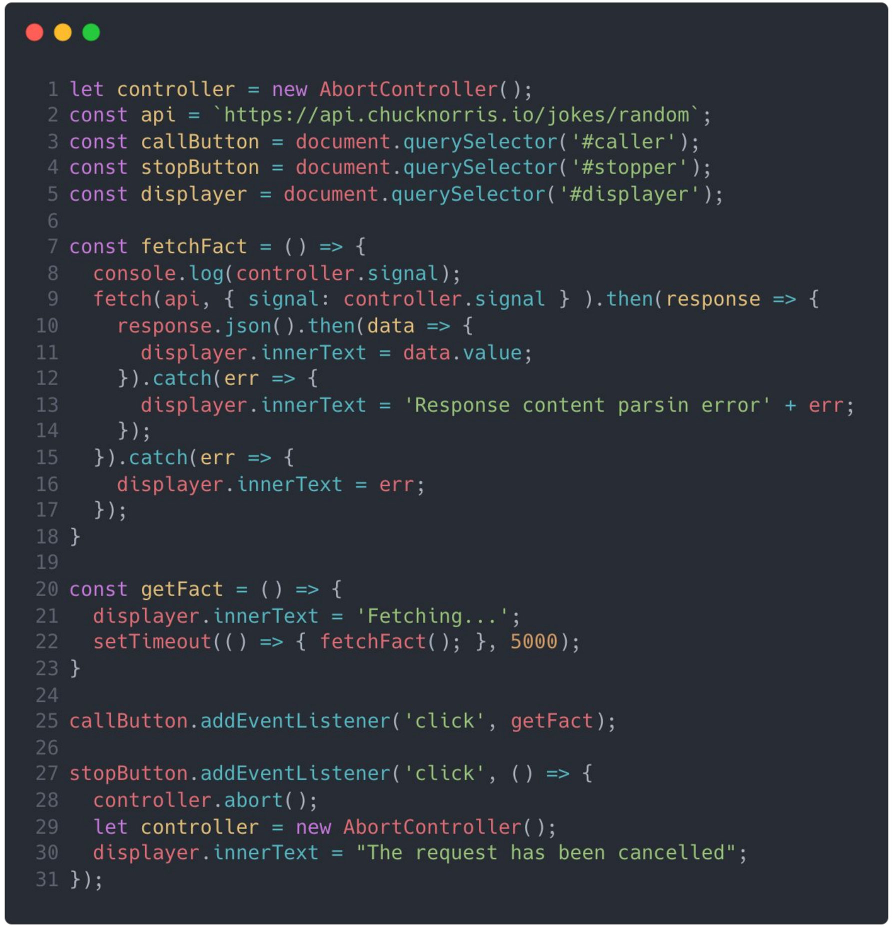


```
fetch(api, {
  method: 'GET', // GET, POST, PUT, DELETE
  mode: 'cors', // no-cors, cors, same-origin
  cache: 'no-cache', // default, no-cache, reload, force-cache, only-if-cached
  credentials: 'same-origin', // include, same-origin, omit
  headers: {
    'Content-Type': 'application/json'
  },
  // .
  // .
  // .
});
```

- `mode` Allows you to resolve only certain requests. For example, cors will allow same origin and cross origin requests.
- `cache` Cache options

Code examples: github.com/asungur/async_js/tree/main/fetch

Demo 2: Reject fetch() using AbortController



```javascript
let controller = new AbortController();
const api = `https://api.chucknorris.io/jokes/random`;
const callButton = document.querySelector('#caller');
const stopButton = document.querySelector('#stopper');
const displayer = document.querySelector('#displayer');

const fetchFact = () => {
  console.log(controller.signal);
  fetch(api, { signal: controller.signal } ).then(response => {
    response.json().then(data => {
      displayer.innerText = data.value;
    }).catch(err => {
      displayer.innerText = 'Response content parsin error' + err;
    });
  }).catch(err => {
    displayer.innerText = err;
  });
}

const getFact = () => {
  displayer.innerText = 'Fetching...';
  setTimeout(() => { fetchFact(); }, 5000);
}

callButton.addEventListener('click', getFact);

stopButton.addEventListener('click', () => {
  controller.abort();
  let controller = new AbortController();
  displayer.innerText = "The request has been cancelled";
});
```

- Add `'click'` event listeners to action and cancel buttons
- Upon call action, display “fetching…” text
- Display response (or error) message.
- Wait 5 seconds (to make this process cancellable)
- Pass signal objects to all async functions to be able to cancel those with `controller.abort()`

Code examples: *github.com/asungur/async_js/tree/main/fetch*