Web Development

COMP 431 / COMP 531

Lecture 13: Resources and Mocking

Instructor: Mack Joyner

Department of Computer Science, Rice University

mjoyner@rice.edu

http://www.clear.rice.edu/comp431

Headstart: download and install

- https://www.clear.rice.edu/comp431/sample/mocking.zip

Frontend Recap

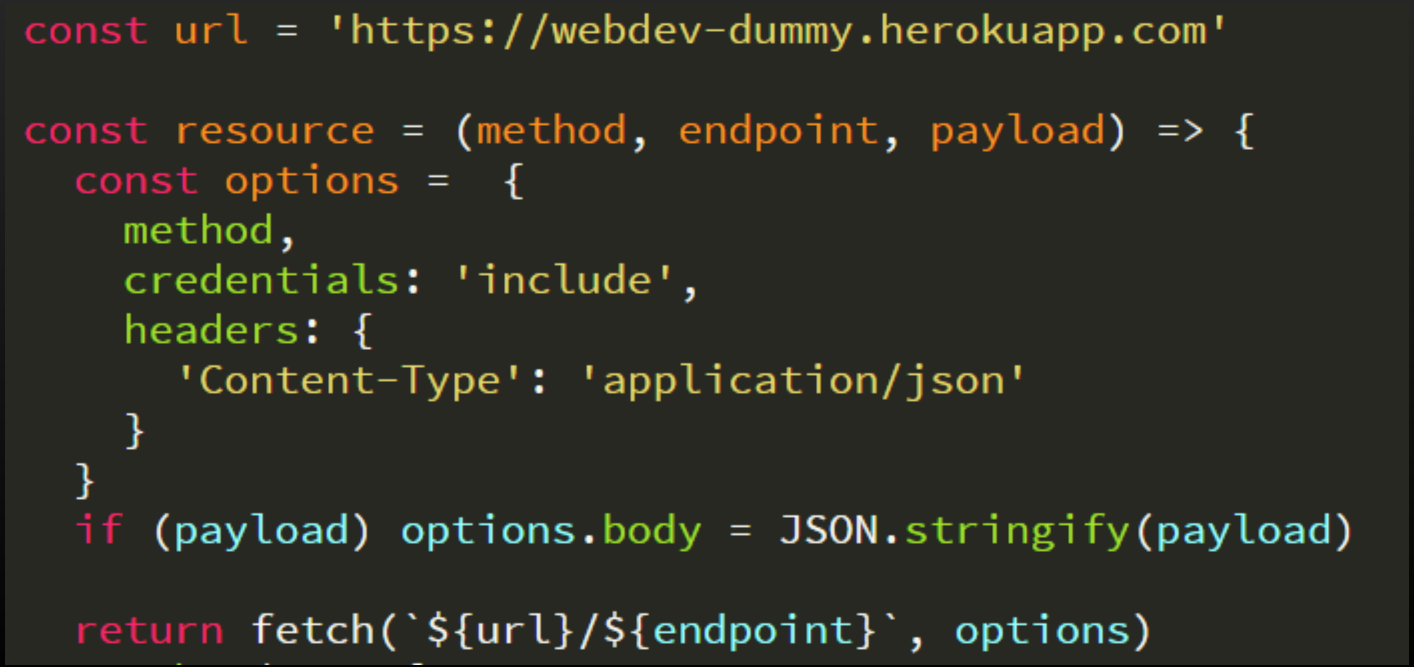
- HTML and HTML5, Storage, Canvas
- JavaScript and Scope
- Forms, CSS, Events
- jQuery, AJAX, and fetch
- Modern JS
- MVC, Angular

Homework Assignment 4
(Draft Front-end)
Due Today

https://jsbin.com/jeliroluni/edit?js,output

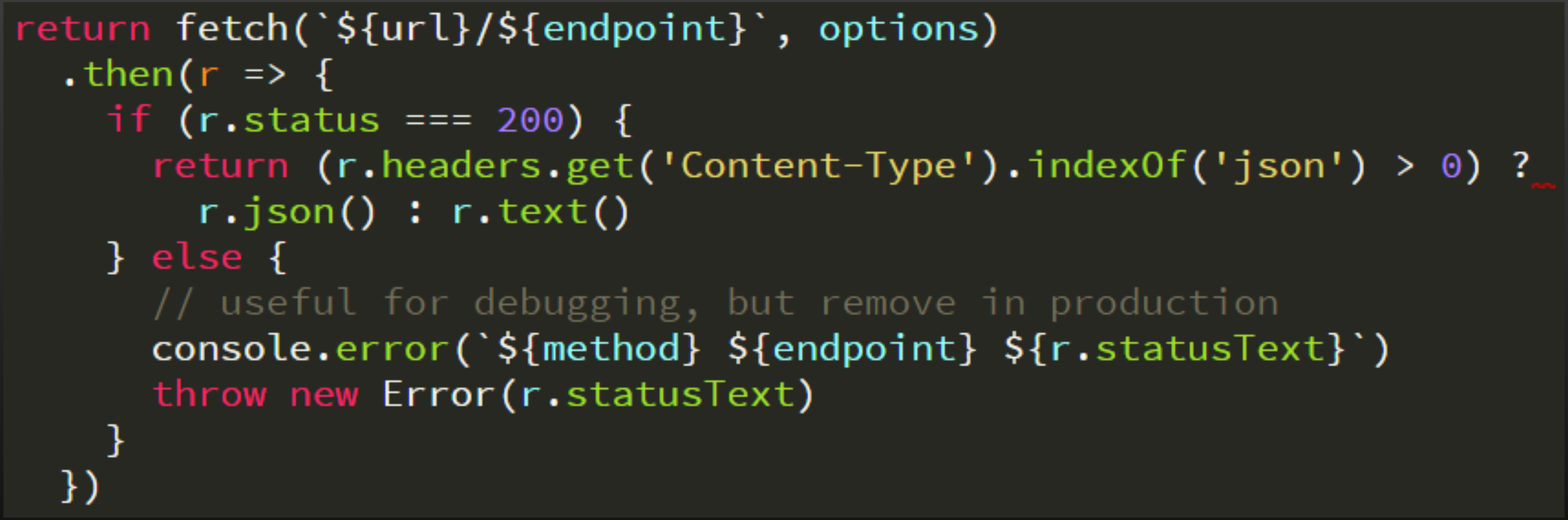
Fetch API calls

- We use fetch() for AJAX calls to the server
- The server will set cookies in the browser
  - Include credentials
- We will be sending JSON payloads.
  - Set a header
- Because we will have many such fetch requests, it is useful to create a wrapper around fetch

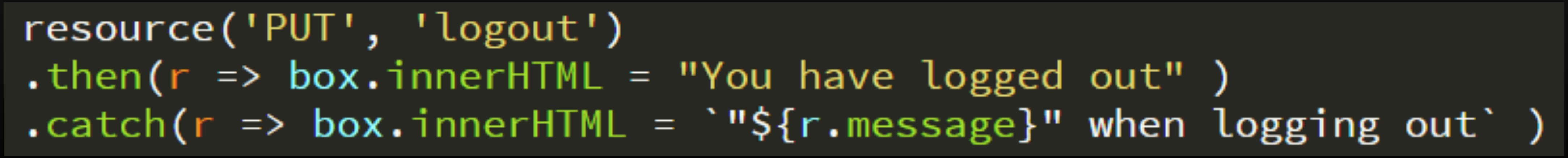


```
const url = 'https://webdev-dummy.herokuapp.com'

const resource = (method, endpoint, payload) => {
  const options =  {
    method,
    credentials: 'include',
    headers: {
      'Content-Type': 'application/json'
    }
  }
  if (payload) options.body = JSON.stringify(payload)

  return fetch(`${url}/${endpoint}`, options)
```

Error Handling



```
return fetch(`${url}/${endpoint}`, options)
  .then(r => {
    if (r.status === 200) {
      return (r.headers.get('Content-Type').indexOf('json') > 0) ?
        r.json() : r.text()
    } else {
      // useful for debugging, but remove in production
      console.error(`${method} ${endpoint} ${r.statusText}`)
      throw new Error(r.statusText)
    }
  })
```



```
resource('PUT', 'logout')
.then(r => box.innerHTML = "You have logged out" )
.catch(r => box.innerHTML = `"${r.message}" when logging out` )
```

Chaining Calls

```
resource('POST', 'login', { username, password })
.then(r => resource('GET', 'headlines'))
.then(r => {
  const user = r.headlines[0]
  box.innerHTML = `you are logged in as ${user.username} "${user.headline}"`
})
.catch(r => box.innerHTML = `"${r.message}" when logging in`)
```

Mocking REST Calls

- In testing we do not want to make actual http calls.
  - **WHY?**

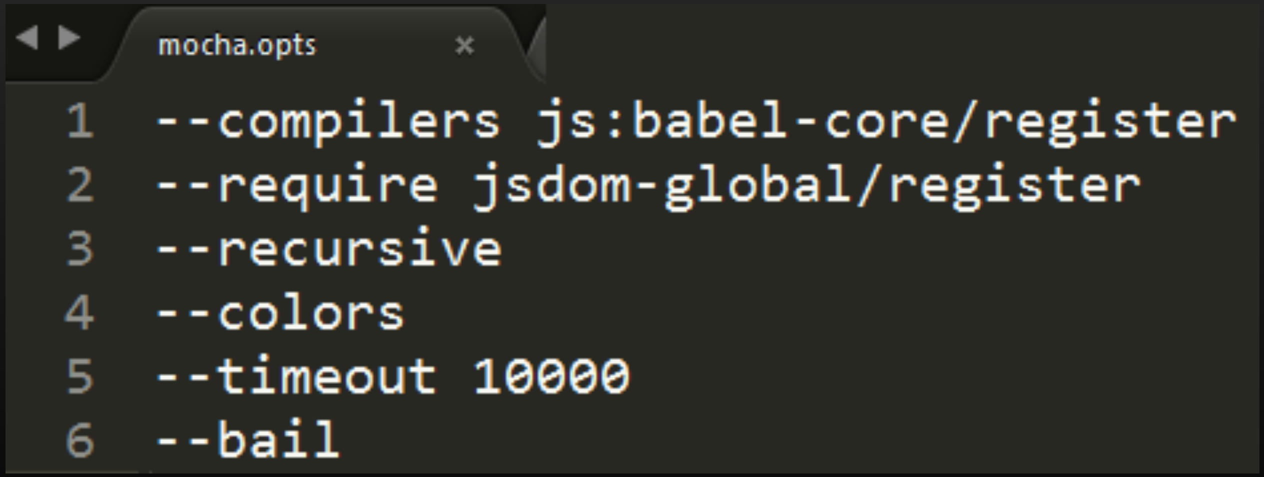
Mocking REST Calls

- In testing we do not want to make actual http calls.

- Fetch is a native function in the browser
- We “override” fetch but without impeding our development
- Tests are run in node, not in the browser

Testing with Mocha

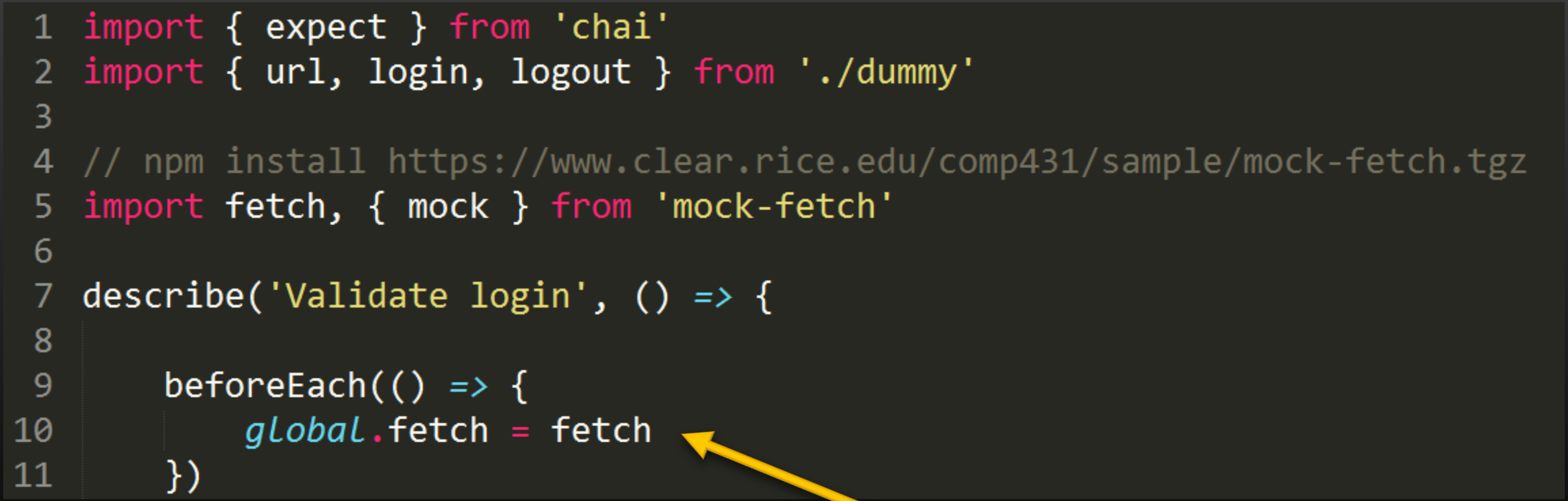
- jsdom provides DOM mocking
- global.document, global.window, etc.
- global.fetch is *not* defined in node.



```
--compilers js:babel-core/register
--require jsdom-global/register
--recursive
--colors
--timeout 10000
--bail
```

Use babel to transpile code while running
Use jsdom in global scope for DOM mocking
recurse directories looking for tests
Use colors in output
Timeout for tests is 10 seconds
Bail on problems

Setup usage of Mocked Fetch



```
import { expect } from 'chai'
import { url, login, logout } from './dummy'

// npm install https://www.clear.rice.edu/comp431/sample/mock-fetch.tgz
import fetch, { mock } from 'mock-fetch'

describe('Validate login', () => {

    beforeEach(() => {
        global.fetch = fetch
    })
```

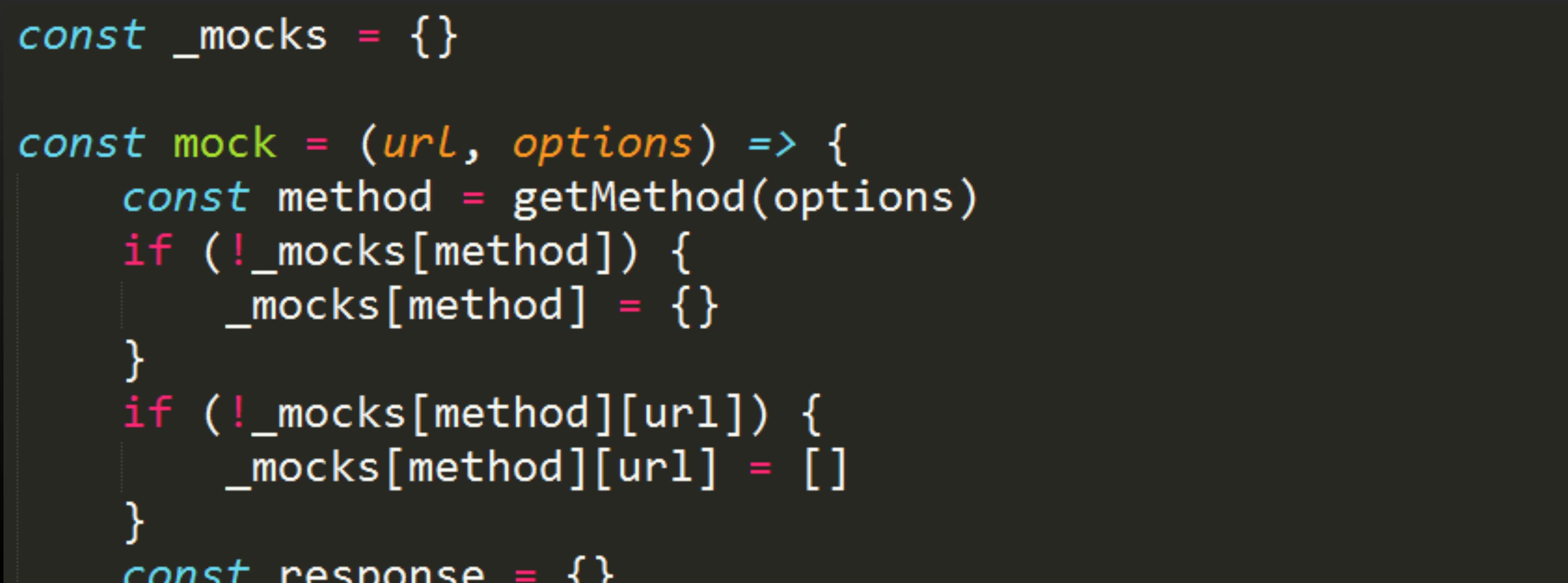
Now “fetch” will be our mocked fetch

Mocked Fetch?

- What should our mocked version of fetch look like?
- What is the API?
- What arguments does it take?
- What does it return?

Mocked Fetch?

- Our test will have a number of fetch requests
- For each fetch request we register a response

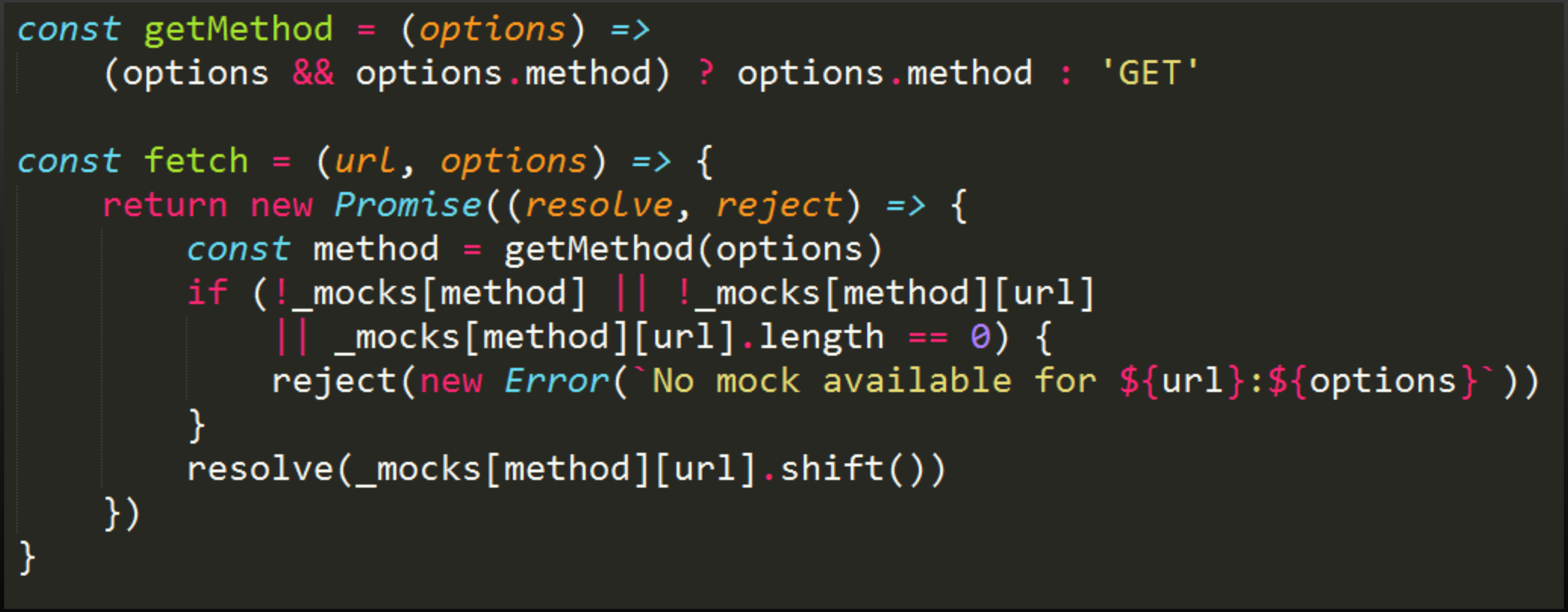


```
const _mocks = {}

const mock = (url, options) => {
    const method = getMethod(options)
    if (!_mocks[method]) {
        _mocks[method] = {}
    }
    if (!_mocks[method][url]) {
        _mocks[method][url] = []
    }
    const response = {}
```

```javascript
const _mocks = {}

const mock = (url, options) => {
    const method = getMethod(options)
    if (!_mocks[method]) {
        _mocks[method] = {}
    }
    if (!_mocks[method][url]) {
        _mocks[method][url] = []
    }
    const response = {}
    Object.keys(options).forEach(key => {
        response[key] = options[key]
    })
    if (!response.status) {
        response.status = 200
    }
    response.headers.get = (key) => options.headers[key]
    response.json = () => new Promise((resolve, reject) => resolve(options.json)
    response.text = () => new Promise((resolve, reject) => resolve(options.text)
    _mocks[method][url].push(response)
}
```

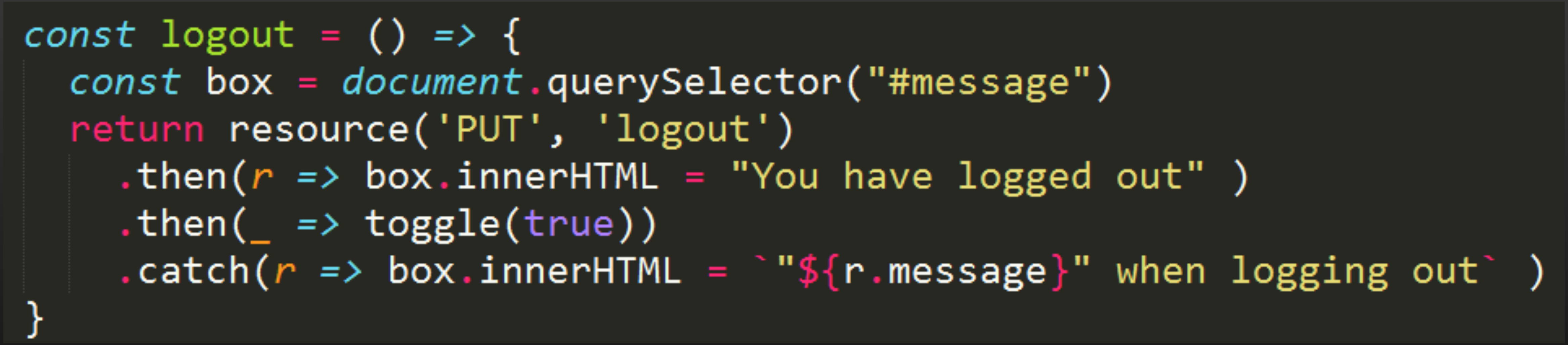
Implementing the fetch API

fetch returns a Promise



```
const getMethod = (options) =>
    (options && options.method) ? options.method : 'GET'

const fetch = (url, options) => {
    return new Promise((resolve, reject) => {
        const method = getMethod(options)
        if (!_mocks[method] || !_mocks[method][url]
            || _mocks[method][url].length == 0) {
            reject(new Error(`No mock available for ${url}:${options}`))
        }
        resolve(_mocks[method][url].shift())
    })
}
```

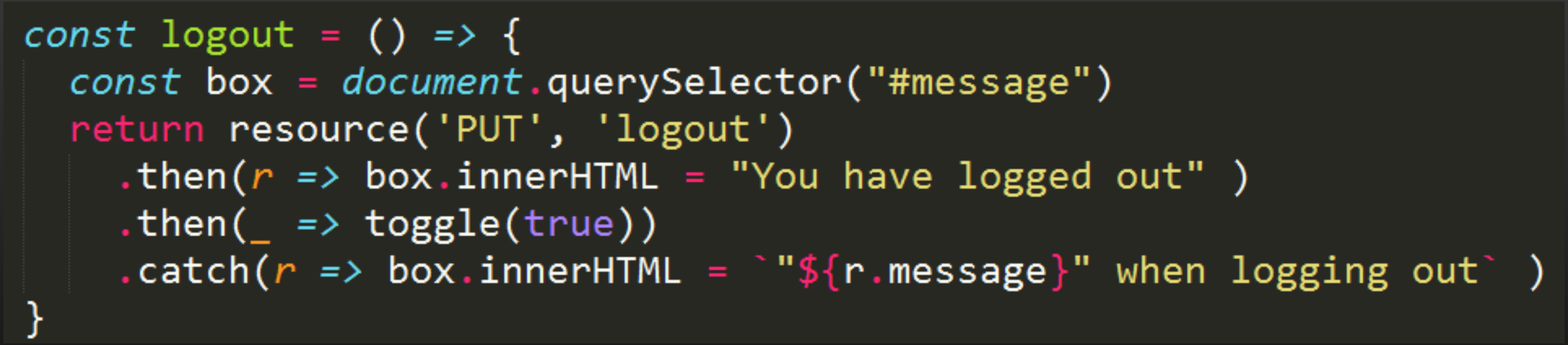
Mock the logout



```
const logout = () => {
  const box = document.querySelector("#message")
  return resource('PUT', 'logout')
    .then(r => box.innerHTML = "You have logged out" )
    .then(_ => toggle(true))
    .catch(r => box.innerHTML = `"${r.message}" when logging out` )
}
```

- What does **PUT /logout** return in this usage?

Mock the logout

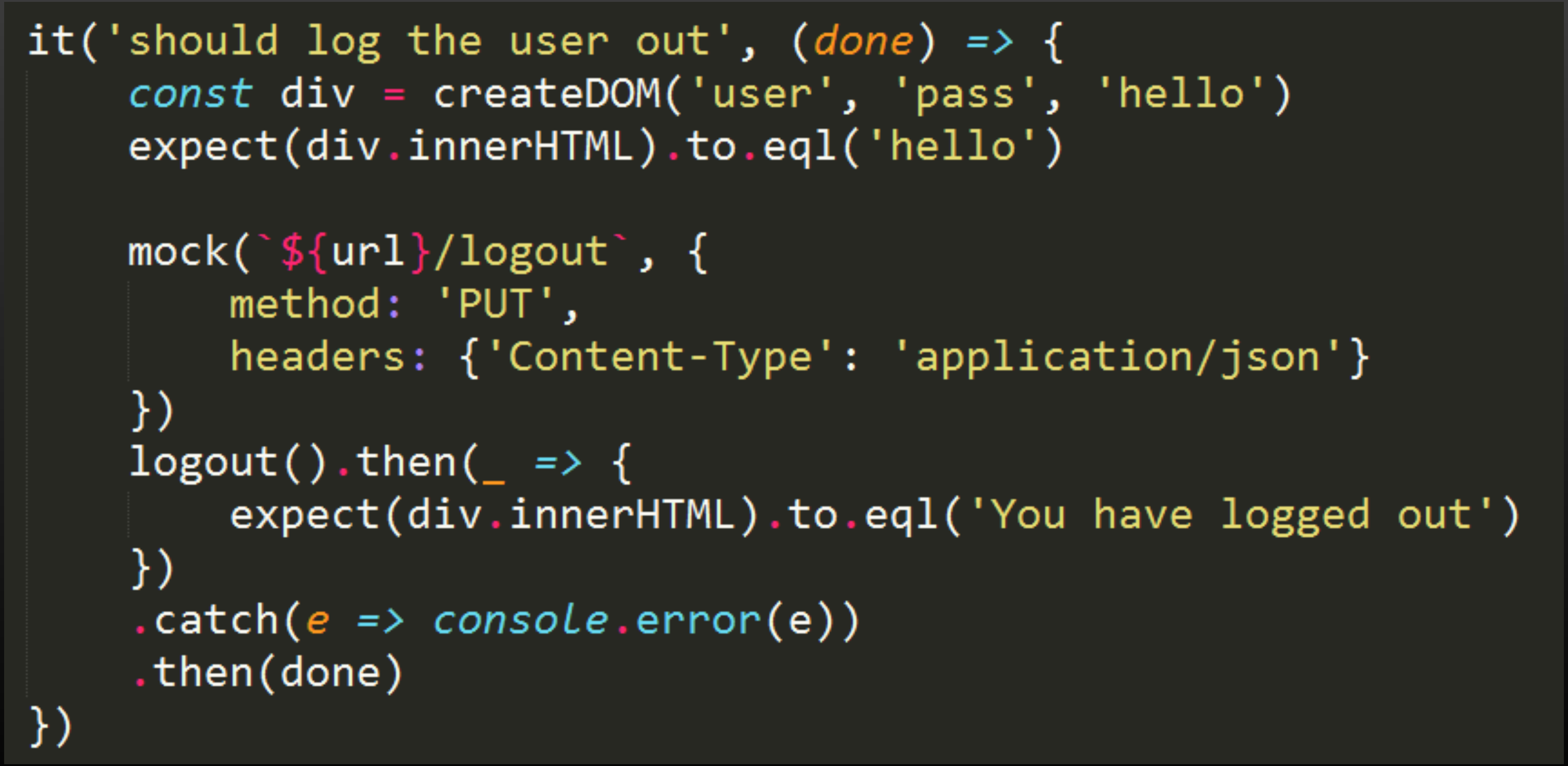


```
const logout = () => {
  const box = document.querySelector("#message")
  return resource('PUT', 'logout')
    .then(r => box.innerHTML = "You have logged out" )
    .then(_ => toggle(true))
    .catch(r => box.innerHTML = `"${r.message}" when logging out` )
}
```

- What does **PUT /logout** return in this usage?

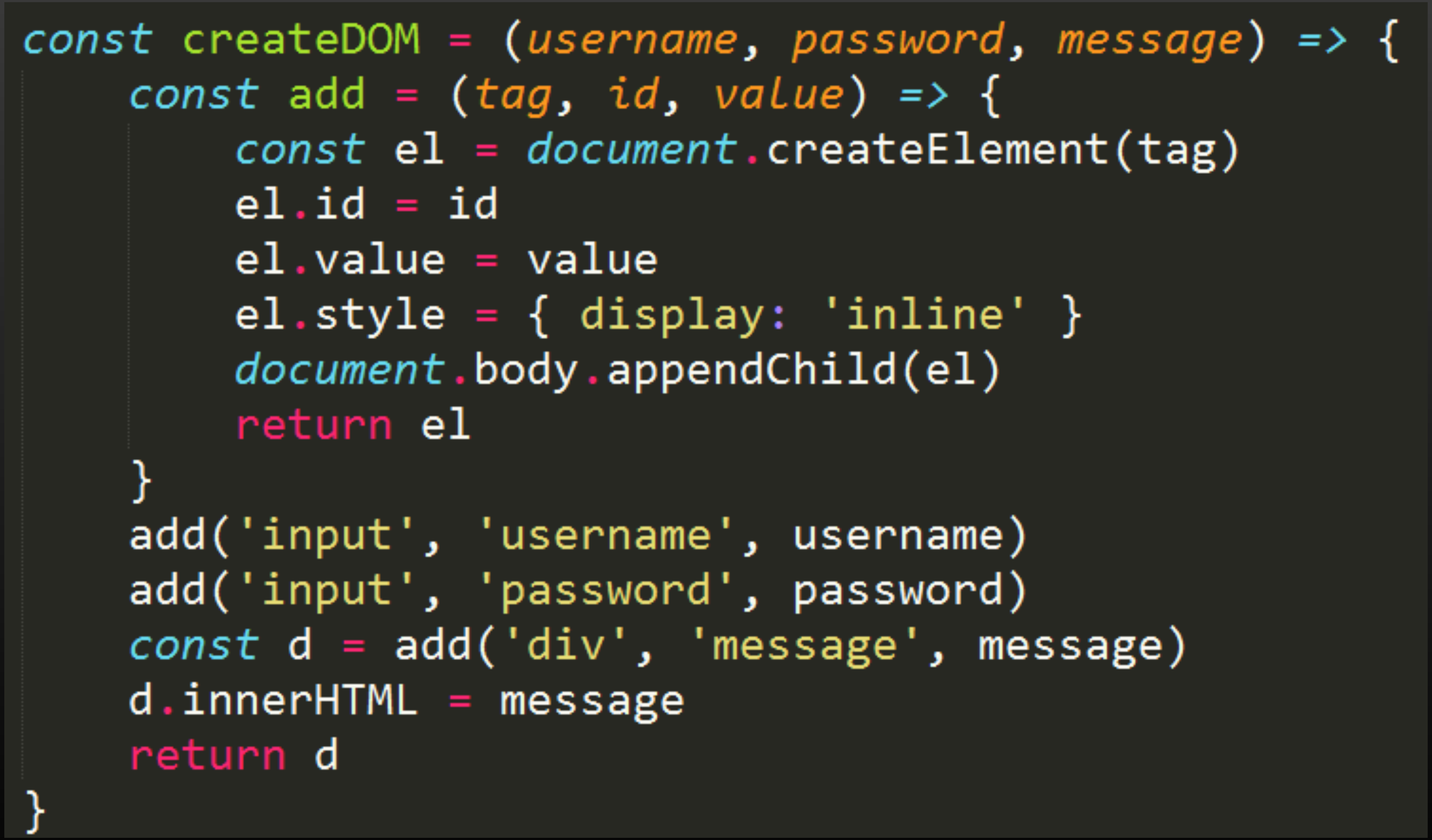
It doesn't matter, we do not use the return payload (from r)
So we don't have to mock anything specific.

Register and use a Mock



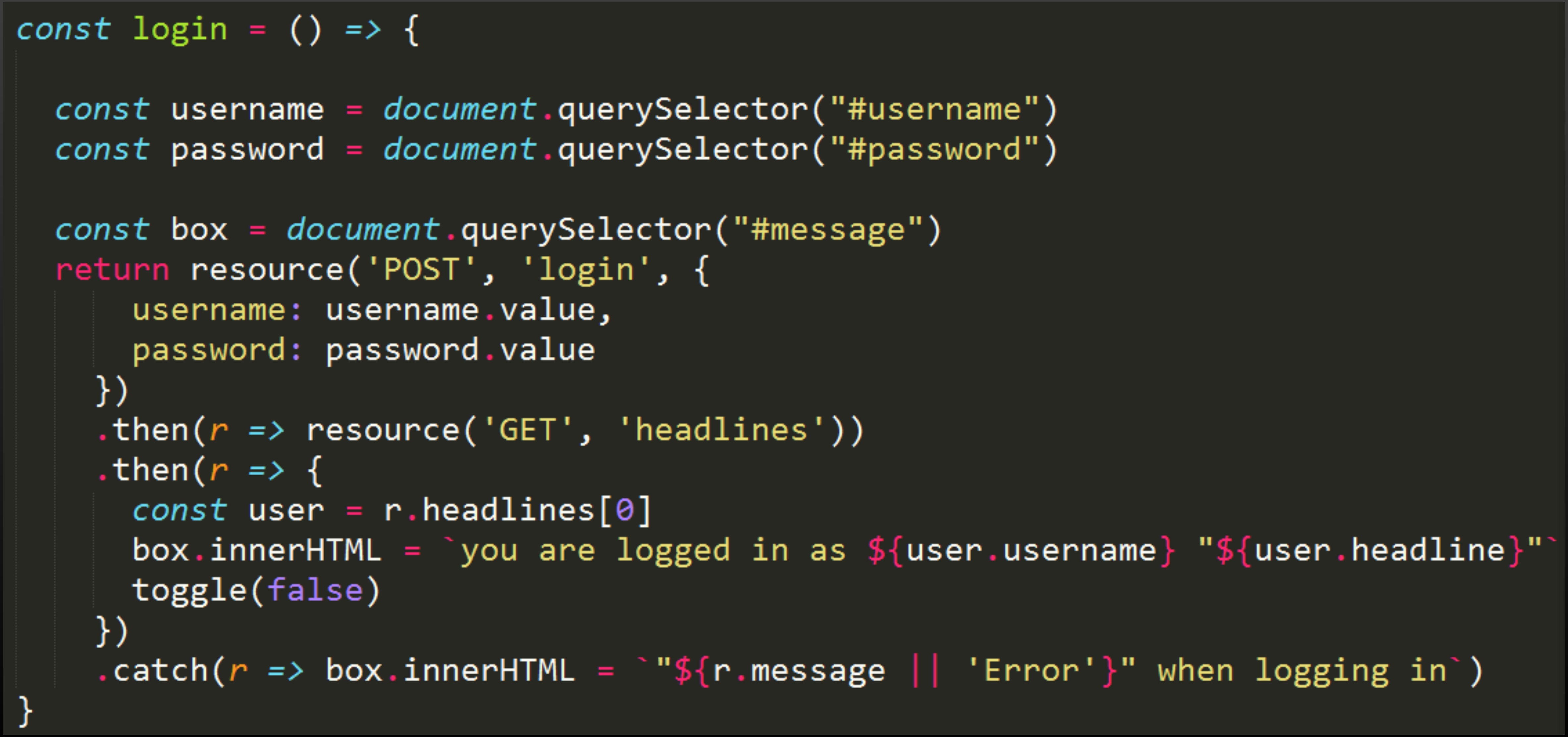
```
it('should log the user out', (done) => {
    const div = createDOM('user', 'pass', 'hello')
    expect(div.innerHTML).to.eql('hello')

    mock(`${url}/logout`, {
        method: 'PUT',
        headers: {'Content-Type': 'application/json'}
    })
    logout().then(_ => {
        expect(div.innerHTML).to.eql('You have logged out')
    })
    .catch(e => console.error(e))
    .then(done)
})
```

Code requires DOM elements (jsdom)



```javascript
const createDOM = (username, password, message) => {
    const add = (tag, id, value) => {
        const el = document.createElement(tag)
        el.id = id
        el.value = value
        el.style = { display: 'inline' }
        document.body.appendChild(el)
        return el
    }
    add('input', 'username', username)
    add('input', 'password', password)
    const d = add('div', 'message', message)
    d.innerHTML = message
    return d
}
```

Mock the login



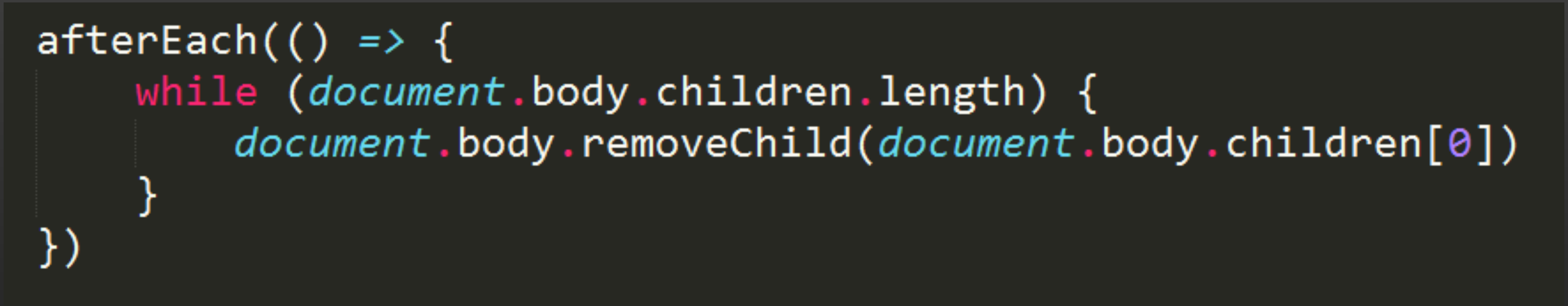
```
const login = () => {

  const username = document.querySelector("#username")
  const password = document.querySelector("#password")

  const box = document.querySelector("#message")
  return resource('POST', 'login', {
        username: username.value,
        password: password.value
    })
    .then(r => resource('GET', 'headlines'))
    .then(r => {
        const user = r.headlines[0]
        box.innerHTML = `you are logged in as ${user.username} "${user.headline}"`
        toggle(false)
    })
    .catch(r => box.innerHTML = `"${r.message || 'Error'}" when logging in`)
}
```

Login Test

```
it('should log the user in', (done) => {
    const div = createDOM('user', 'pass', 'hello')
    expect(div.innerHTML).to.eql('hello')

    mock(`${url}/login`, {
        method: 'POST',
        headers: {'Content-Type': 'application/json'}
    })
    mock(`${url}/headlines`, {
        headers: {'Content-Type': 'application/json'},
        json: {
            headlines: [{username: 'hi', headline:'ok'}]
        }
    })

    login().then(_ => {
        expect(div.innerHTML)
            .to.eql('you are logged in as hi "ok"')
    })
    .catch(e => console.error(e))
    .then(done)
})
```

mocked data

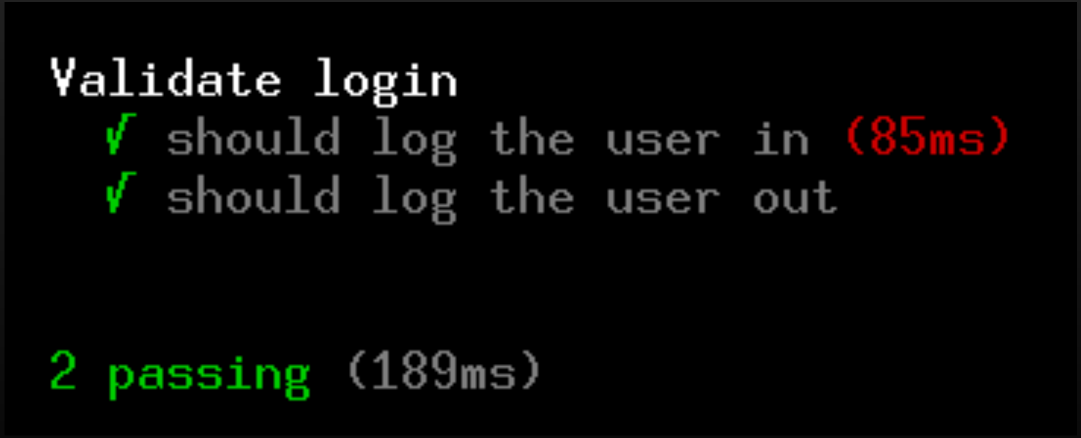
- Clean up the DOM after each test



```
afterEach(() => {
    while (document.body.children.length) {
        document.body.removeChild(document.body.children[0])
    }
})
```

- Exercise the tests:

```
npm test
```



```
Validate login
  √ should log the user in (85ms)
  √ should log the user out


2 passing (189ms)
```