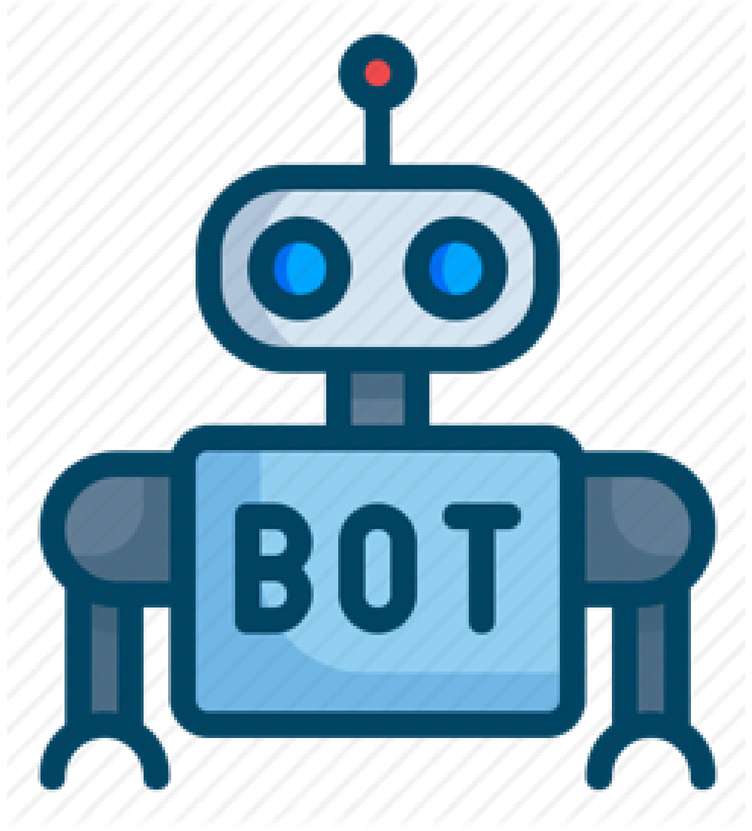
Testing your awesome code!

Armando Cifuentes González

Software Development Engineer in Test

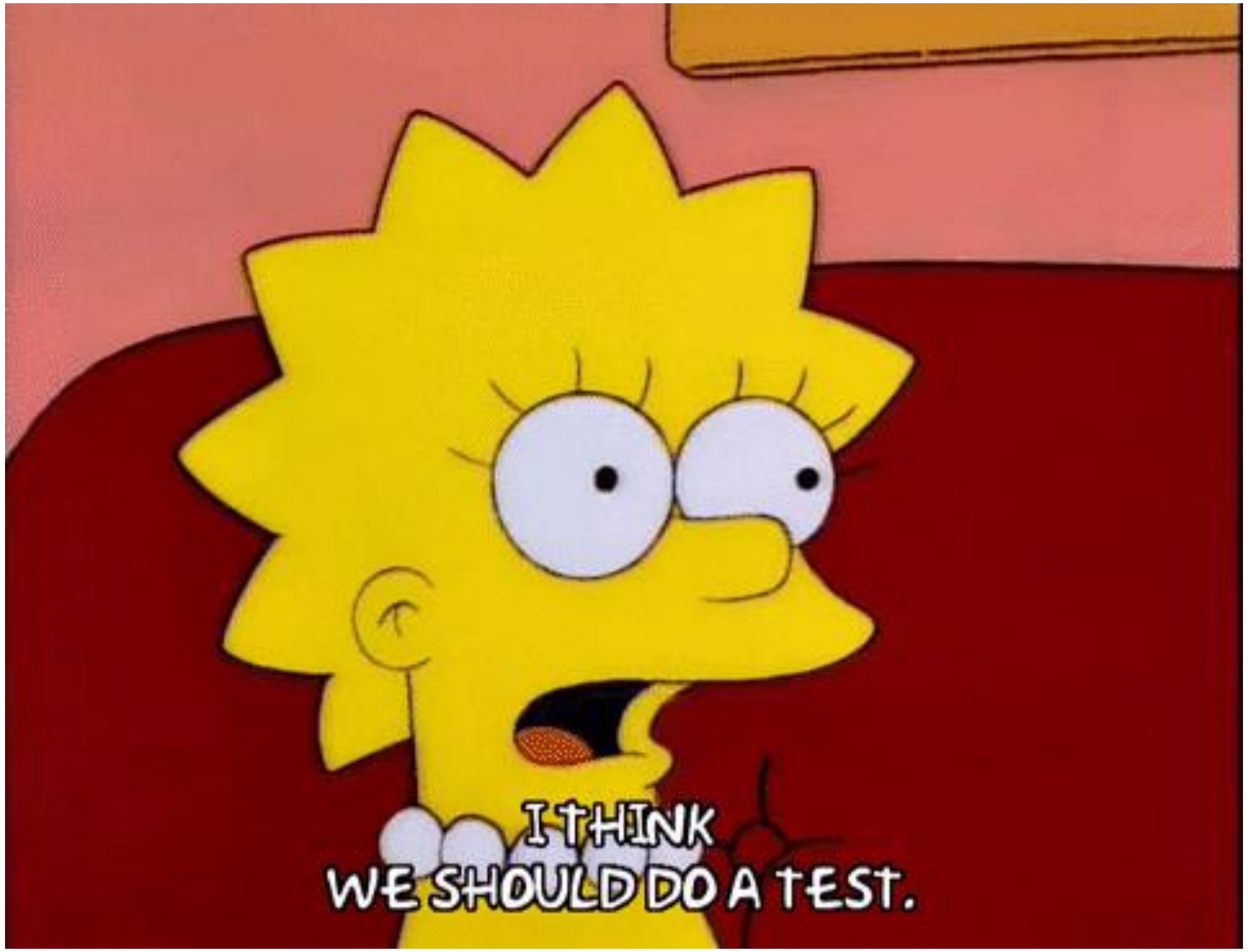
- **LinkedIn:** *https://www.linkedin.com/in/arcigo*
- **GitHub:** *https://github.com/ArCiGo*
- **Twitter:** @_ArCIGo
- *armandocifuentes_2@yahoo.com.mx*



What is testing?

There are many aspects to software testing. It does not always involve using the product. It is not just about finding bugs. Testing can start around the requirements stage. Thinking about what the product should do, where risks could be, and how the user/customer navigates the product is all part of testing.

"Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation."


I THINK
WE SHOULD DO A TEST.

What is unit testing?

A unit test (component testing, according to the **ISTQB**) is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions turn out to be wrong, the unit test has failed. A unit is a method or function.

Objectives of unit testing

- Reducing risk.
- Verifying whether the functional and non-functional behaviours of the component are as designed and specified.
- Building confidence in the component’s quality.
- Finding defects in the component
- Preventing defects from escaping to higher test levels.

Test elements

Test basis

- Detailed design.
- Code.
- Data model.
- Component specifications.

Test objects

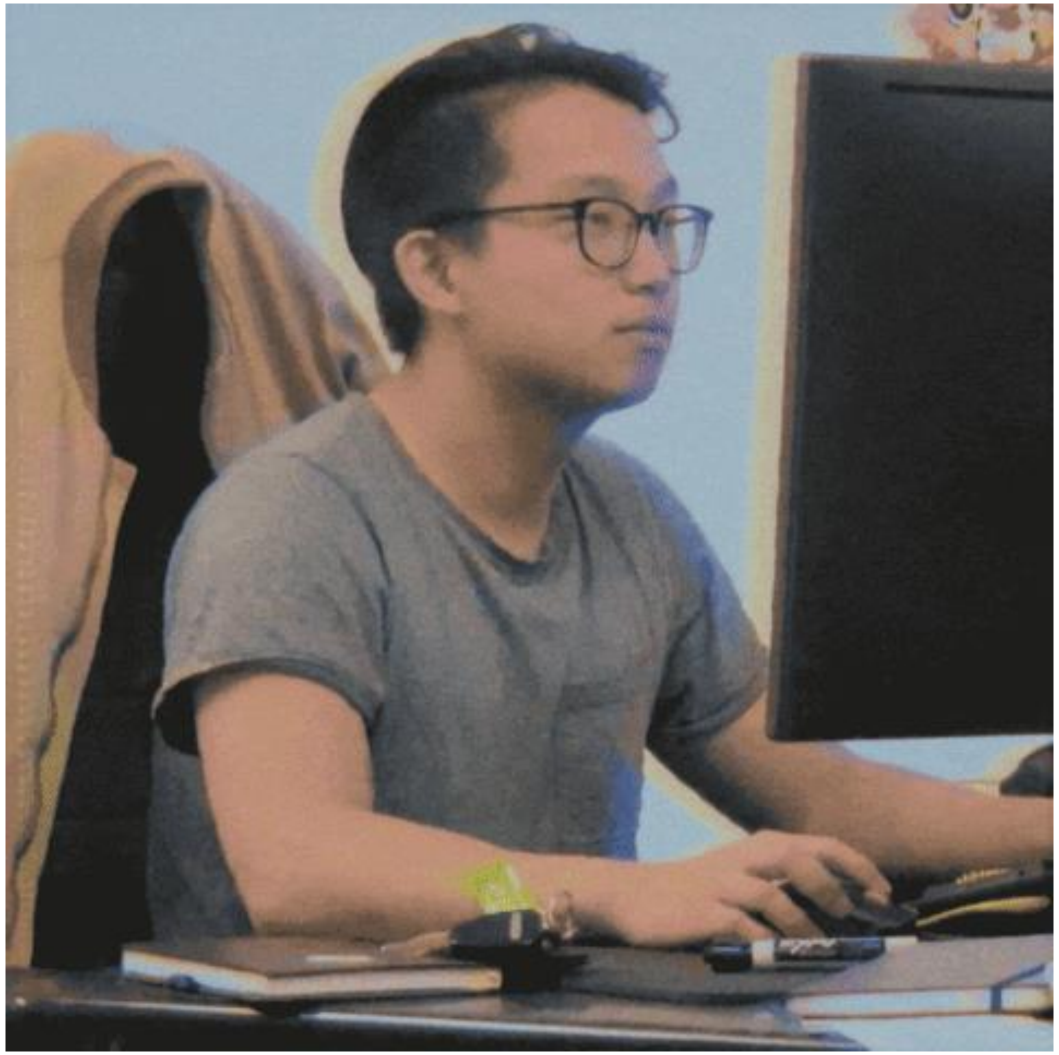
- Components, units or modules.
- Code and data structures.
- Classes.
- Database modules.

Typical defects and failures

- Incorrect functionality (e.g., not as described in design specifications).
- Data flow problems.
- Incorrect code and logic.

Benefits of unit testing

- Makes the process agile.
- Quality of code.
- Finds software bugs early.
- Facilitates changes and simplifies integration.
- Provides documentation.
- Debugging process.
- Design.
- Reduce costs.



What is e2e testing?

e2e testing is a technique that tests the entire software product from beginning to end to ensure the application flow behaves as expected. It defines the product’s system dependencies and ensures all integrated pieces work together as expected.

The main purpose of e2e testing is to test from the end user’s experience by simulating the real user scenario and validating the system under test and its components for integration and data integrity.

Benefits of e2e testing

- Expand test coverage.
- Reduce time to market.
- Reduce cost.
- Detect bugs.

But, where to start?

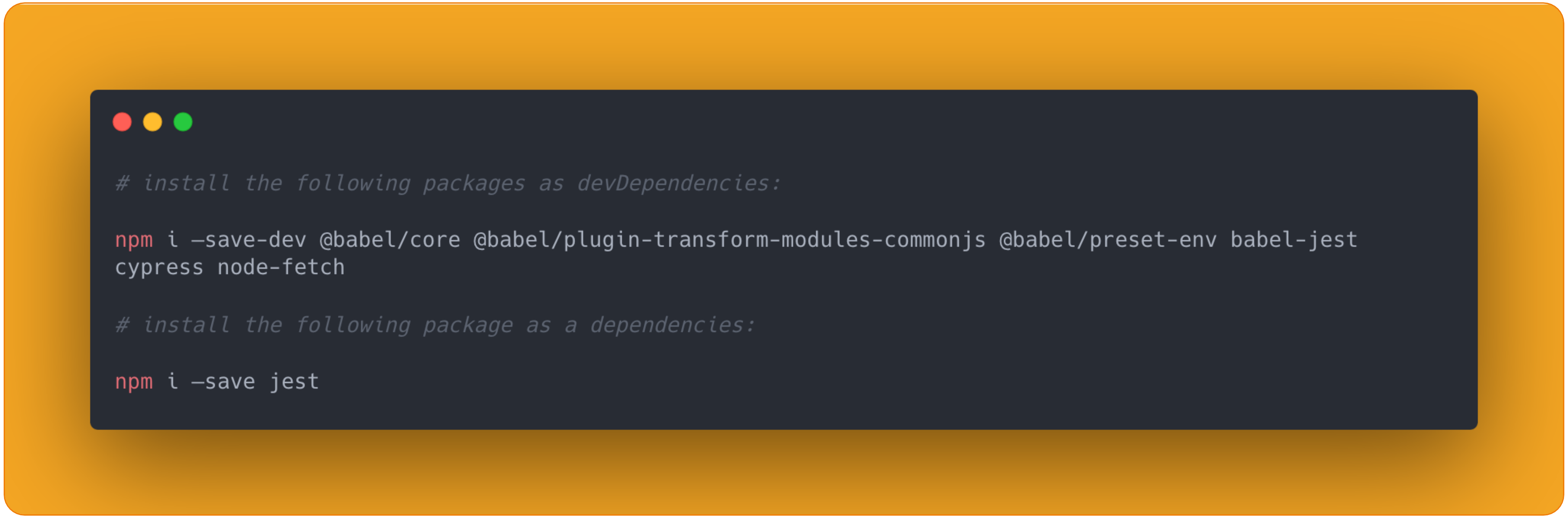
I think is one of the most difficult questions to answer. In the first place, before starting writing code think in your tests and think like a final (common) user (and tester). How can your code be violated? How the user is going to use the application? Imagine all the scenarios, including the ridiculous ones, in which the application can be used.

How can I do my code testable? Do I need to refactor my code to test? What pieces of code can I test? Remember, is not possible to test all your code, and it's not possible to get coverage of 100%. Focus on functionality.

Which tools do I need to use? Another difficult question. There are a lot of tools in the market, but it depends on the stack you are using to code (and sometimes the available documentation).

Use **TDD** and **BDD**. *Test-Driven Development* is one of the most popular techniques to start writing code using tests. I know it's difficult to follow, but you can get good results at the moment of developing your application. *Behaviour-Driven Development* could help you to think like a final user, following the business rules of the project.

Example

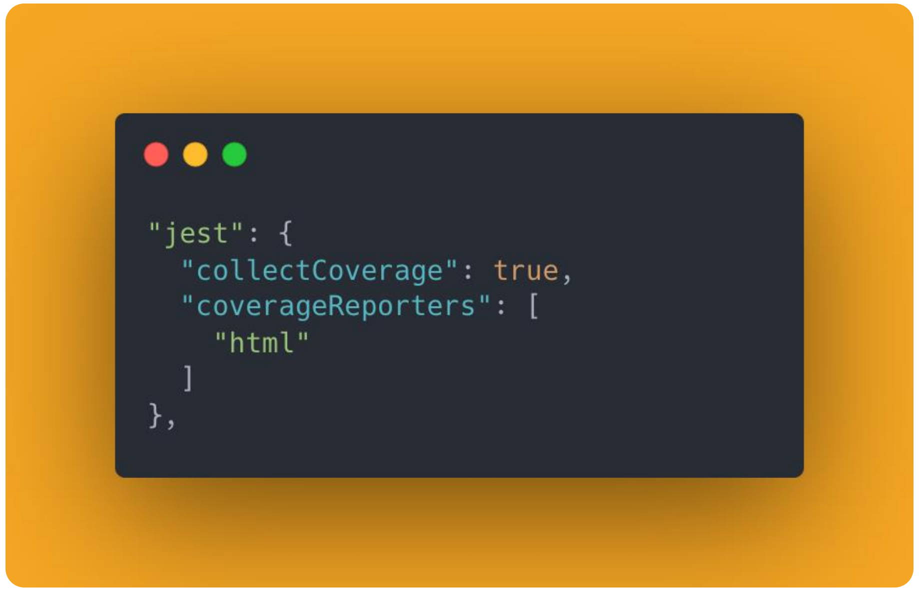


```
# install the following packages as devDependencies:

npm i –save-dev @babel/core @babel/plugin-transform-modules-commonjs @babel/preset-env babel-jest
cypress node-fetch

# install the following package as a dependencies:

npm i –save jest
```

Setting up our project

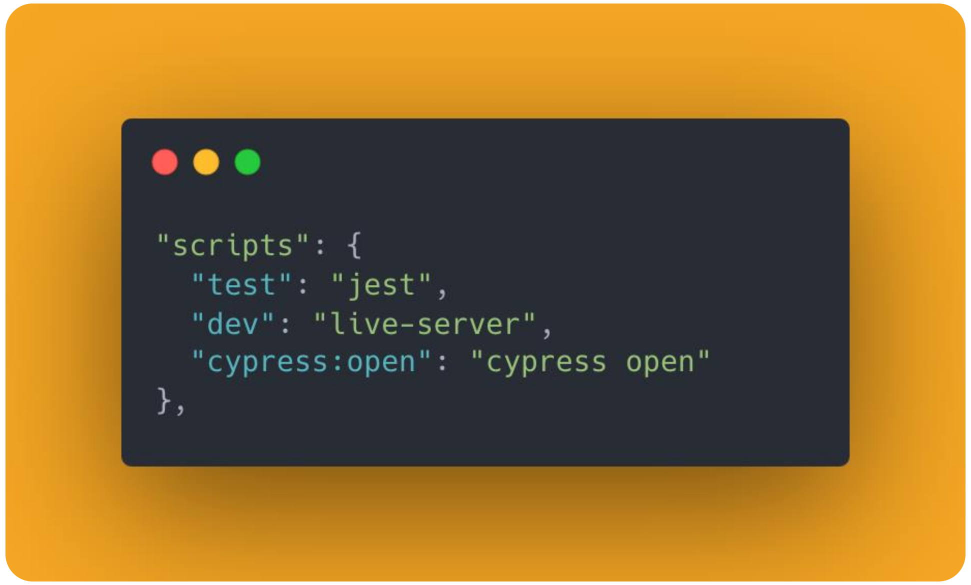
In order to create and execute our tests, open a terminal, pointing to the project, and execute the commands shown in the image.

In the `package.json`, add the following section.-



```
"jest": {
  "collectCoverage": true,
  "coverageReporters": [
    "html"
  ]
},
```

In the `scripts` section, add the following values.-



```
"scripts": {
  "test": "jest",
  "dev": "live-server",
  "cypress:open": "cypress open"
},
```

At the root project level, add a folder to contain the unit tests. The structure should be as follows:

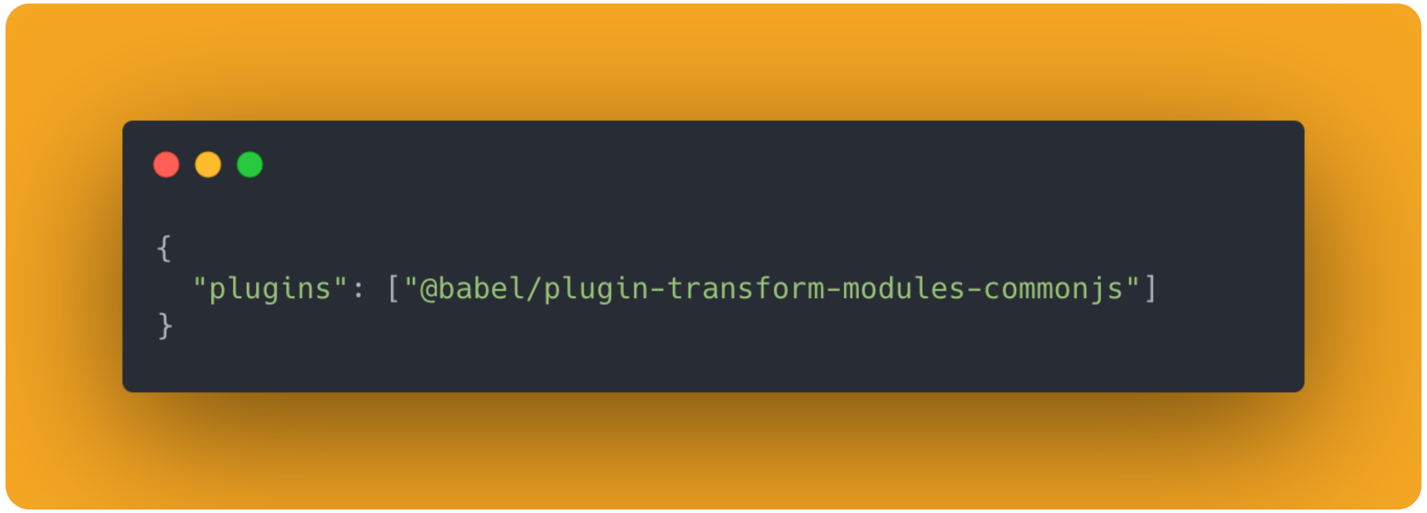
`test => unit => app.test.js`.

Inside of the `unit` folder, add a `resources` folder where you can add your **`JS`** files that will contain ob jects, requests, responses, etcetera, that will be used by your unit tests.

Enable the transform object in the `jest.config.js` file. Add the following value:

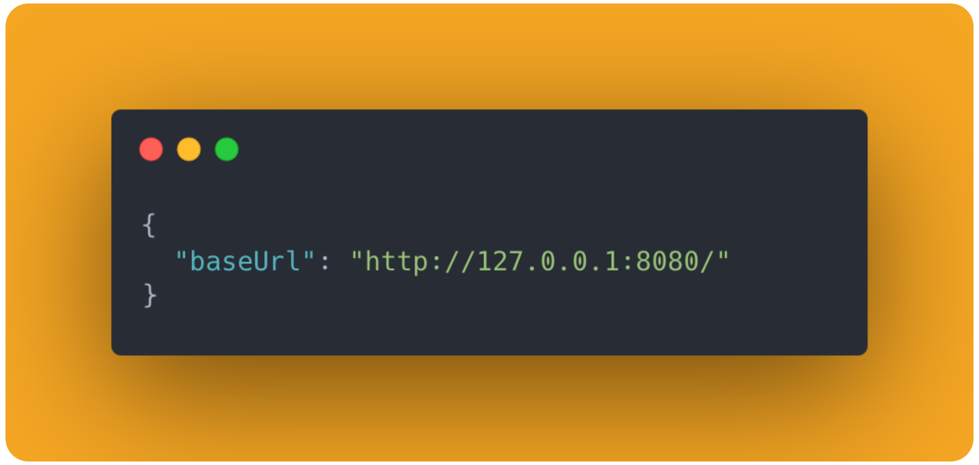
```
"^.+\\.[j|t]sx?$": "babel-jest"
```

At the root project level, add a `.babelrc` file, and write the following line:



```
{
  "plugins": ["@babel/plugin-transform-modules-commonjs"]
}
```

Modify the `cypress.json` file as follows:

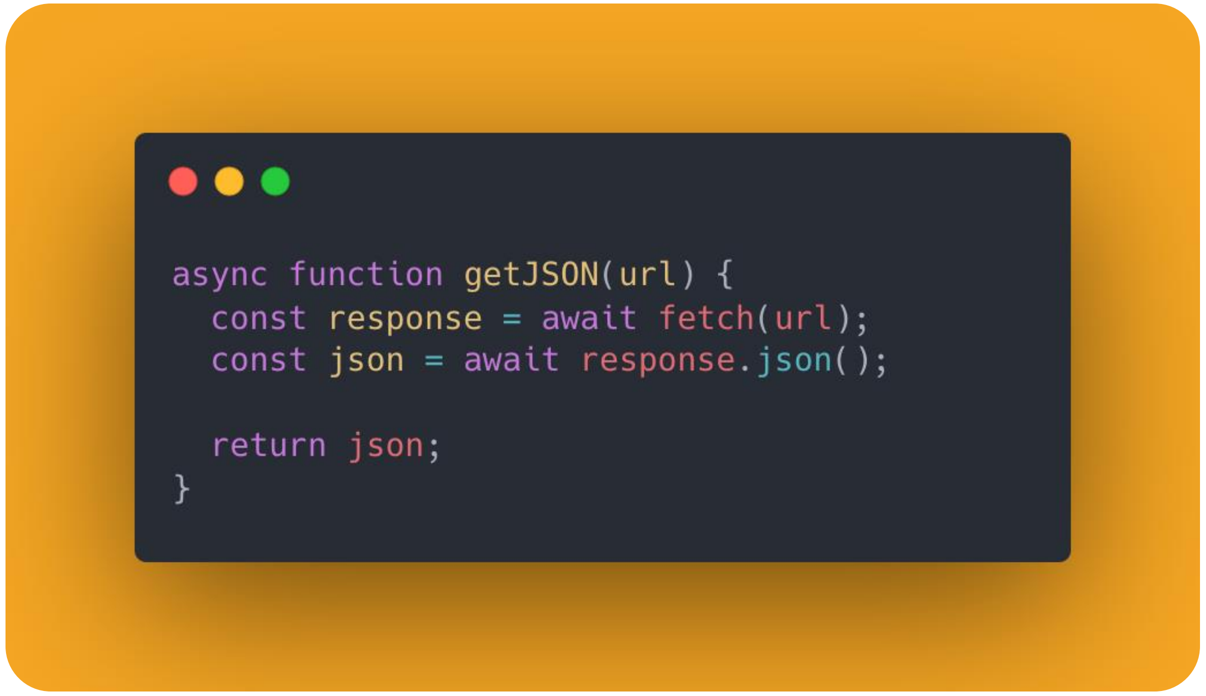


```json
{
  "baseUrl": "http://127.0.0.1:8080/"
}
```

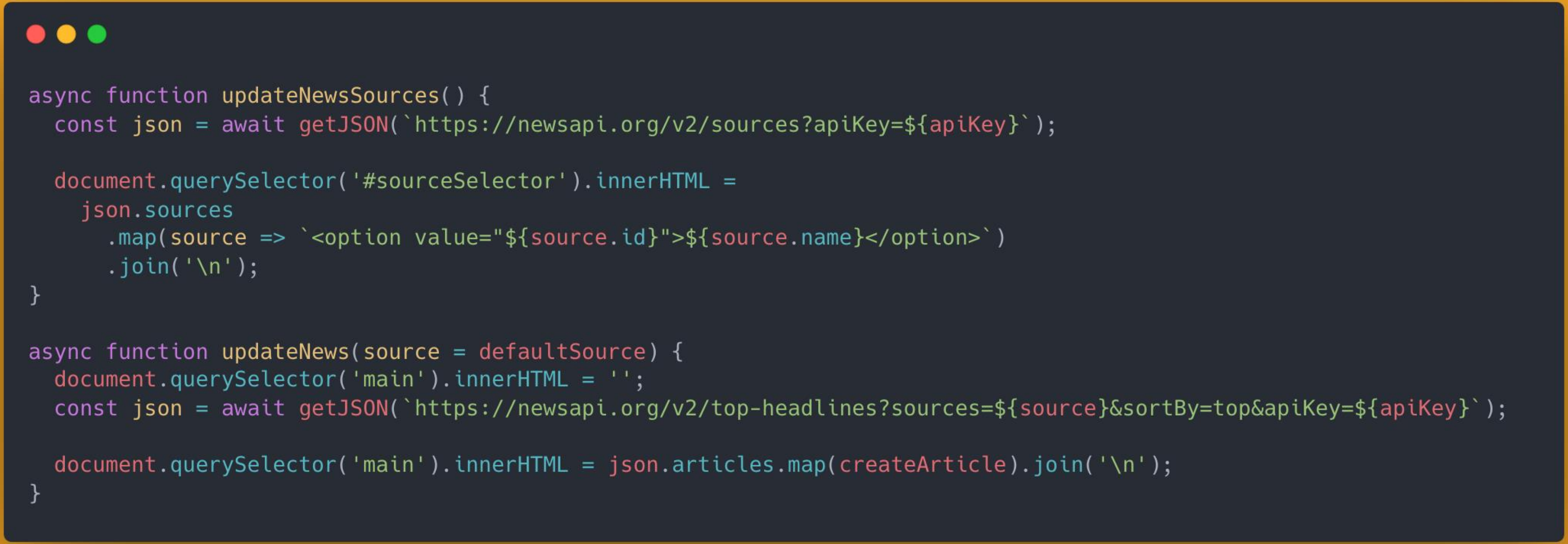
Inside of the `cypress => integration => examples` folder, delete the integration tests created durin the installation of **Cypress** and add a `PWA.spec.js` file test.

Unit tests

We did some changes to the `app.js` file to make it testable. We separated the lines of code that calls the **News API** into a new function called `getJSON(param)` as follows:

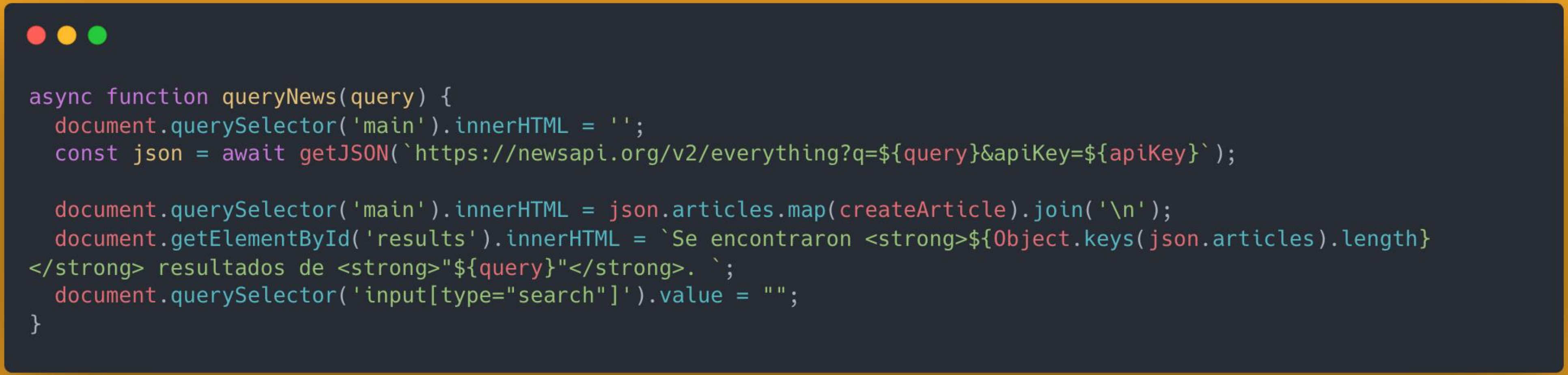


```
async function getJSON(url) {
  const response = await fetch(url);
  const json = await response.json();

  return json;
}
```

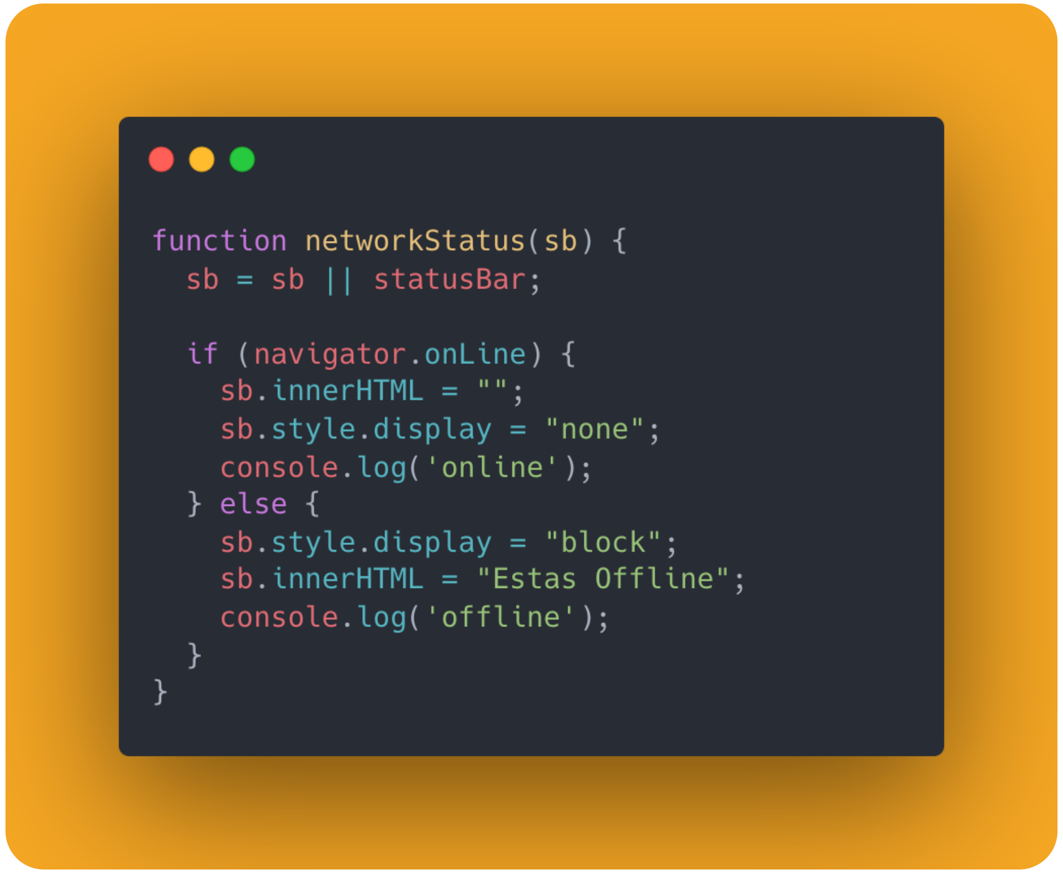


```
async function updateNewsSources() {
  const json = await getJSON(`https://newsapi.org/v2/sources?apiKey=${apiKey}`);

  document.querySelector('#sourceSelector').innerHTML =
    json.sources
      .map(source => `<option value="${source.id}">${source.name}</option>`)
      .join('\n');
}

async function updateNews(source = defaultSource) {
  document.querySelector('main').innerHTML = '';
  const json = await getJSON(`https://newsapi.org/v2/top-headlines?sources=${source}&sortBy=top&apiKey=${apiKey}`);

  document.querySelector('main').innerHTML = json.articles.map(createArticle).join('\n');
}
```

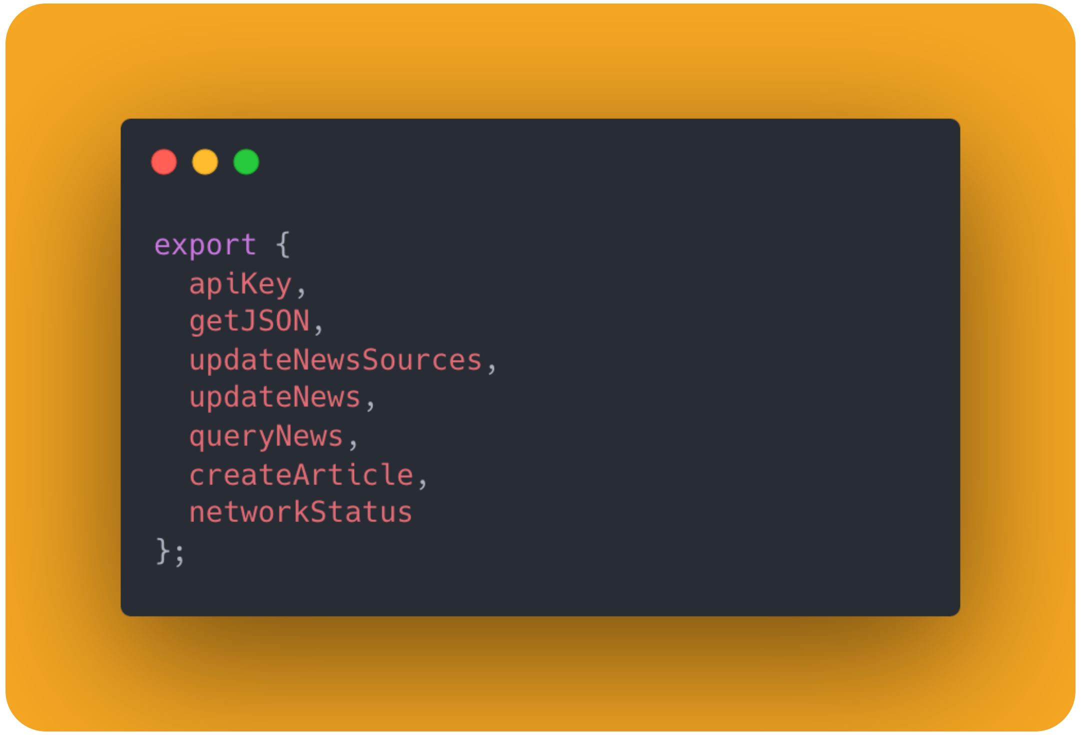
Now, our code looks like this…



```
async function queryNews(query) {
  document.querySelector('main').innerHTML = '';
  const json = await getJSON(`https://newsapi.org/v2/everything?q=${query}&apiKey=${apiKey}`);

  document.querySelector('main').innerHTML = json.articles.map(createArticle).join('\n');
  document.getElementById('results').innerHTML = `Se encontraron <strong>${Object.keys(json.articles).length}
</strong> resultados de <strong>"${query}"</strong>. `;
  document.querySelector('input[type="search"]').value = "";
}
```



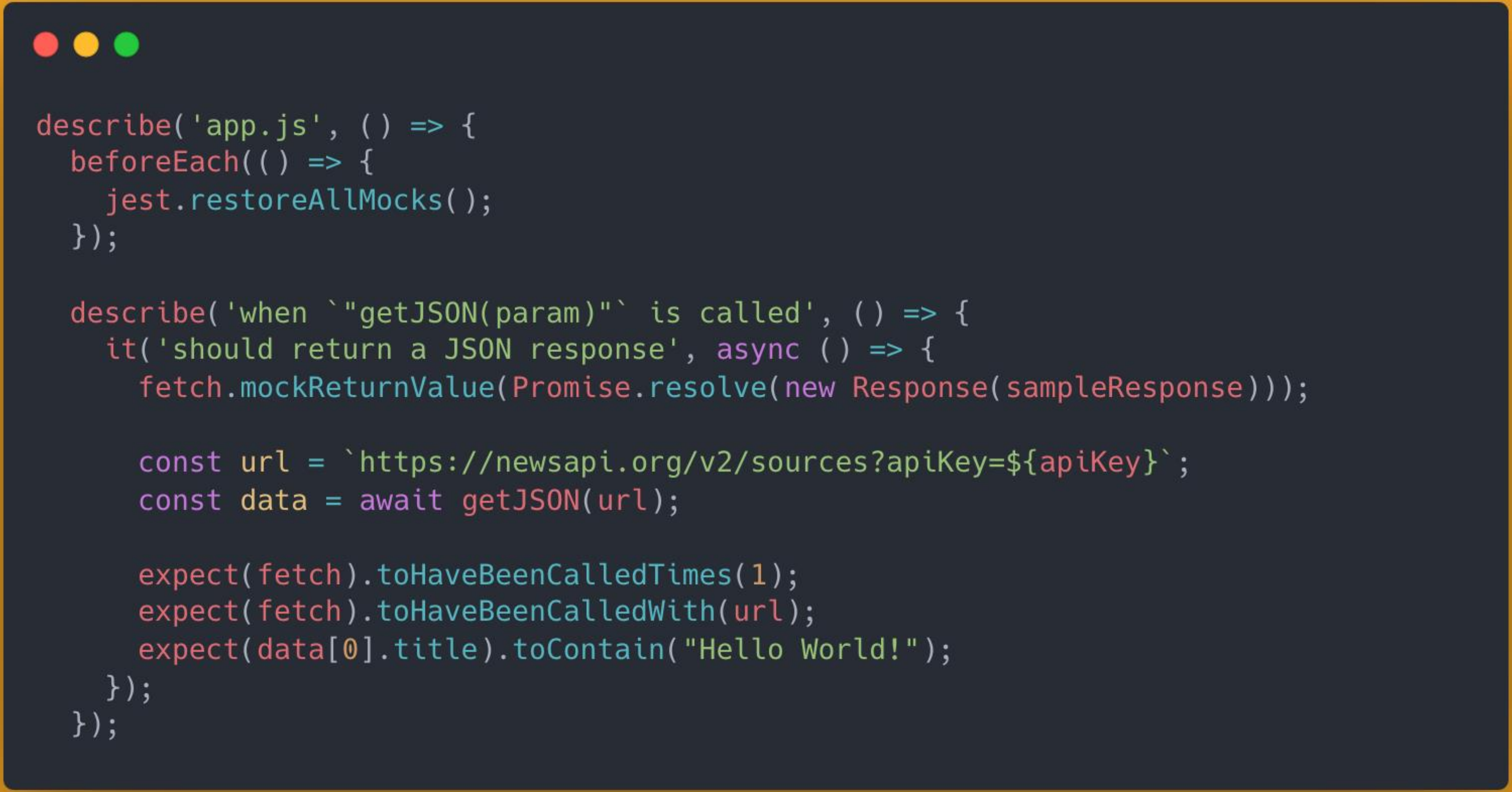
```
function networkStatus(sb) {
  sb = sb || statusBar;

  if (navigator.onLine) {
    sb.innerHTML = "";
    sb.style.display = "none";
    console.log('online');
  } else {
    sb.style.display = "block";
    sb.innerHTML = "Estas Offline";
    console.log('offline');
  }
}
```



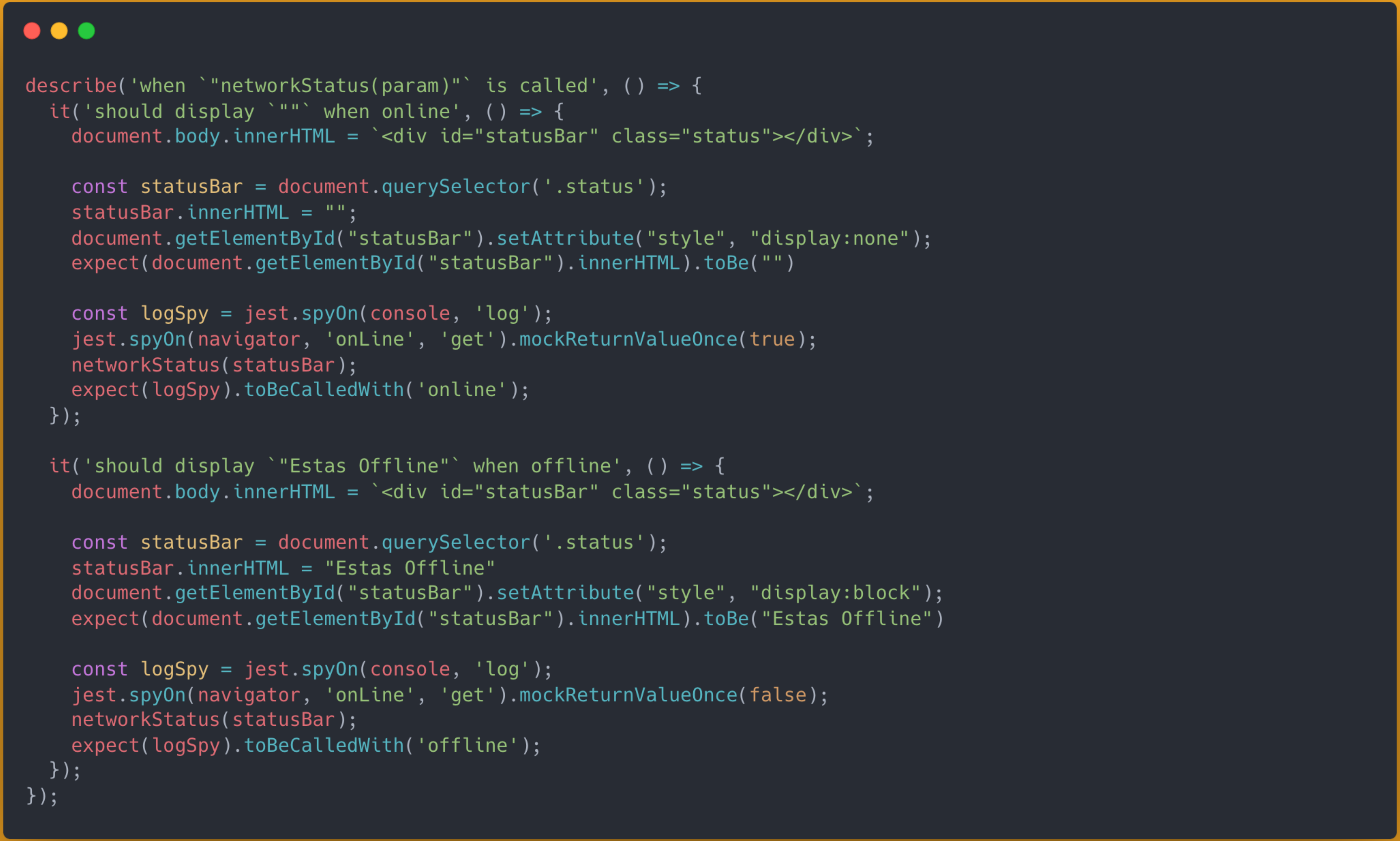
```
export {
  apiKey,
  getJSON,
  updateNewsSources,
  updateNews,
  queryNews,
  createArticle,
  networkStatus
};
```

Other updates we did with the code were the following…

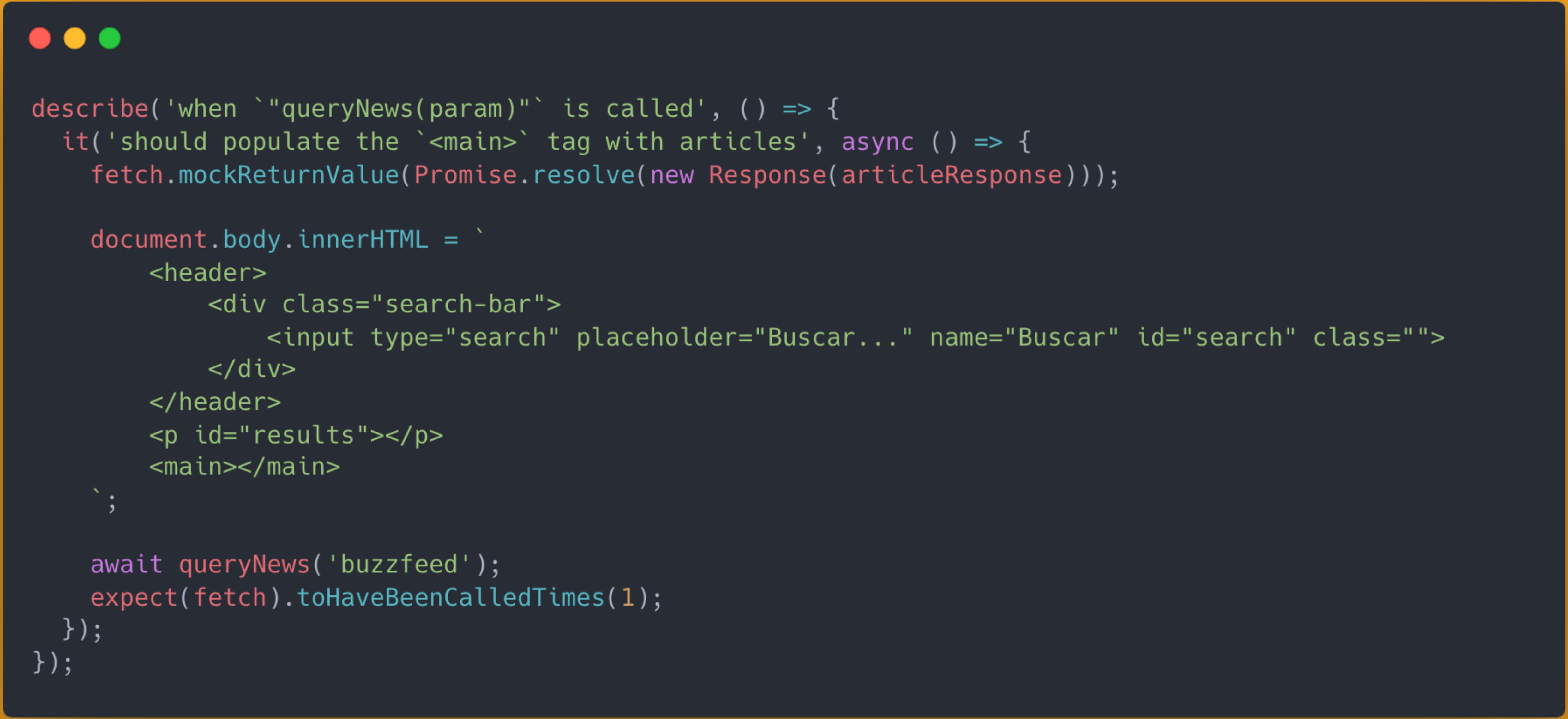
Let’s take a look at our suite of unit tests. We created a main suite called `app.js`, and inside of it, there are another sub-suites to test the different functions available in our code. Inside of every sub-suite, there are different tests to prove the different scenarios available for our code.



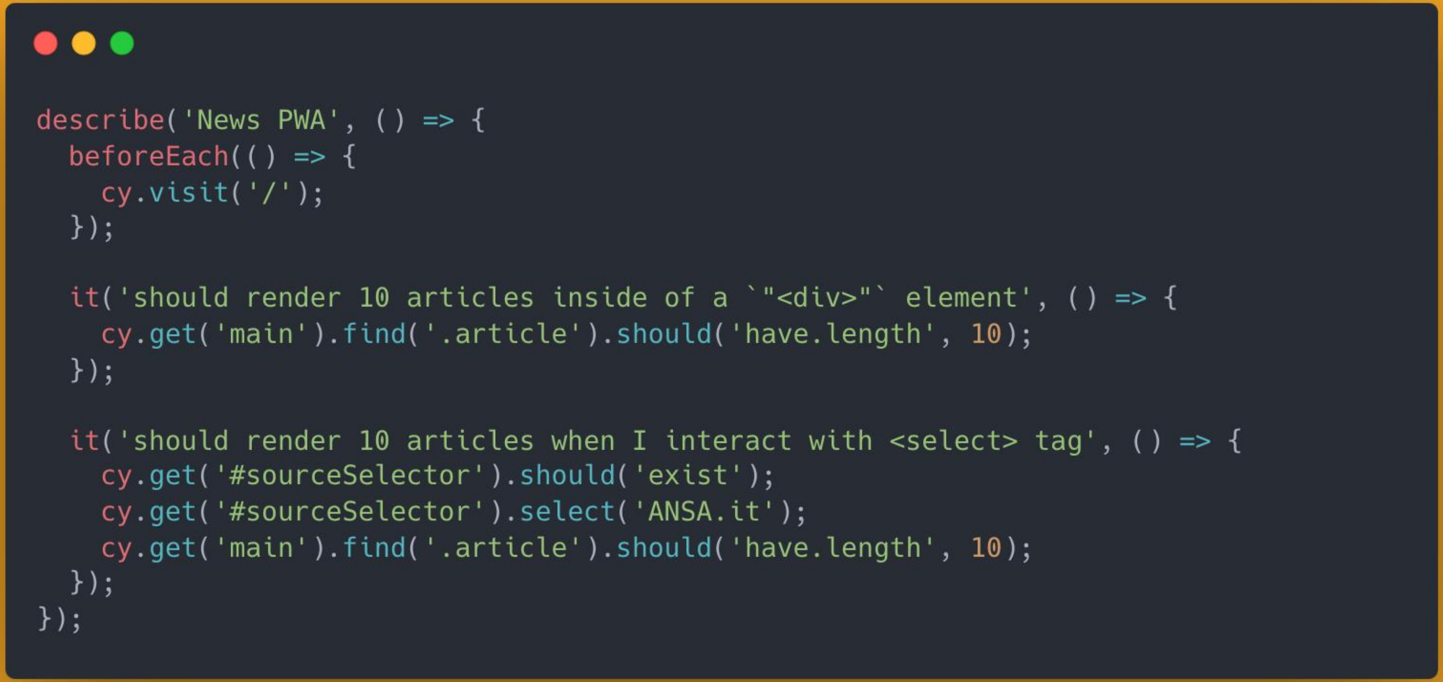
```javascript
describe('app.js', () => {
  beforeEach(() => {
    jest.restoreAllMocks();
  });

  describe('when `"getJSON(param)"` is called', () => {
    it('should return a JSON response', async () => {
      fetch.mockReturnValue(Promise.resolve(new Response(sampleResponse)));

      const url = `https://newsapi.org/v2/sources?apiKey=${apiKey}`;
      const data = await getJSON(url);

      expect(fetch).toHaveBeenCalledTimes(1);
      expect(fetch).toHaveBeenCalledWith(url);
      expect(data[0].title).toContain("Hello World!");
    });
  });
```



```
describe('when `"networkStatus(param)"` is called', () => {
  it('should display `""` when online', () => {
    document.body.innerHTML = `<div id="statusBar" class="status"></div>`;

    const statusBar = document.querySelector('.status');
    statusBar.innerHTML = "";
    document.getElementById("statusBar").setAttribute("style", "display:none");
    expect(document.getElementById("statusBar").innerHTML).toBe("")

    const logSpy = jest.spyOn(console, 'log');
    jest.spyOn(navigator, 'onLine', 'get').mockReturnValueOnce(true);
    networkStatus(statusBar);
    expect(logSpy).toBeCalledWith('online');
  });

  it('should display `"Estas Offline"` when offline', () => {
    document.body.innerHTML = `<div id="statusBar" class="status"></div>`;

    const statusBar = document.querySelector('.status');
    statusBar.innerHTML = "Estas Offline"
    document.getElementById("statusBar").setAttribute("style", "display:block");
    expect(document.getElementById("statusBar").innerHTML).toBe("Estas Offline")

    const logSpy = jest.spyOn(console, 'log');
    jest.spyOn(navigator, 'onLine', 'get').mockReturnValueOnce(false);
    networkStatus(statusBar);
    expect(logSpy).toBeCalledWith('offline');
  });
});
```

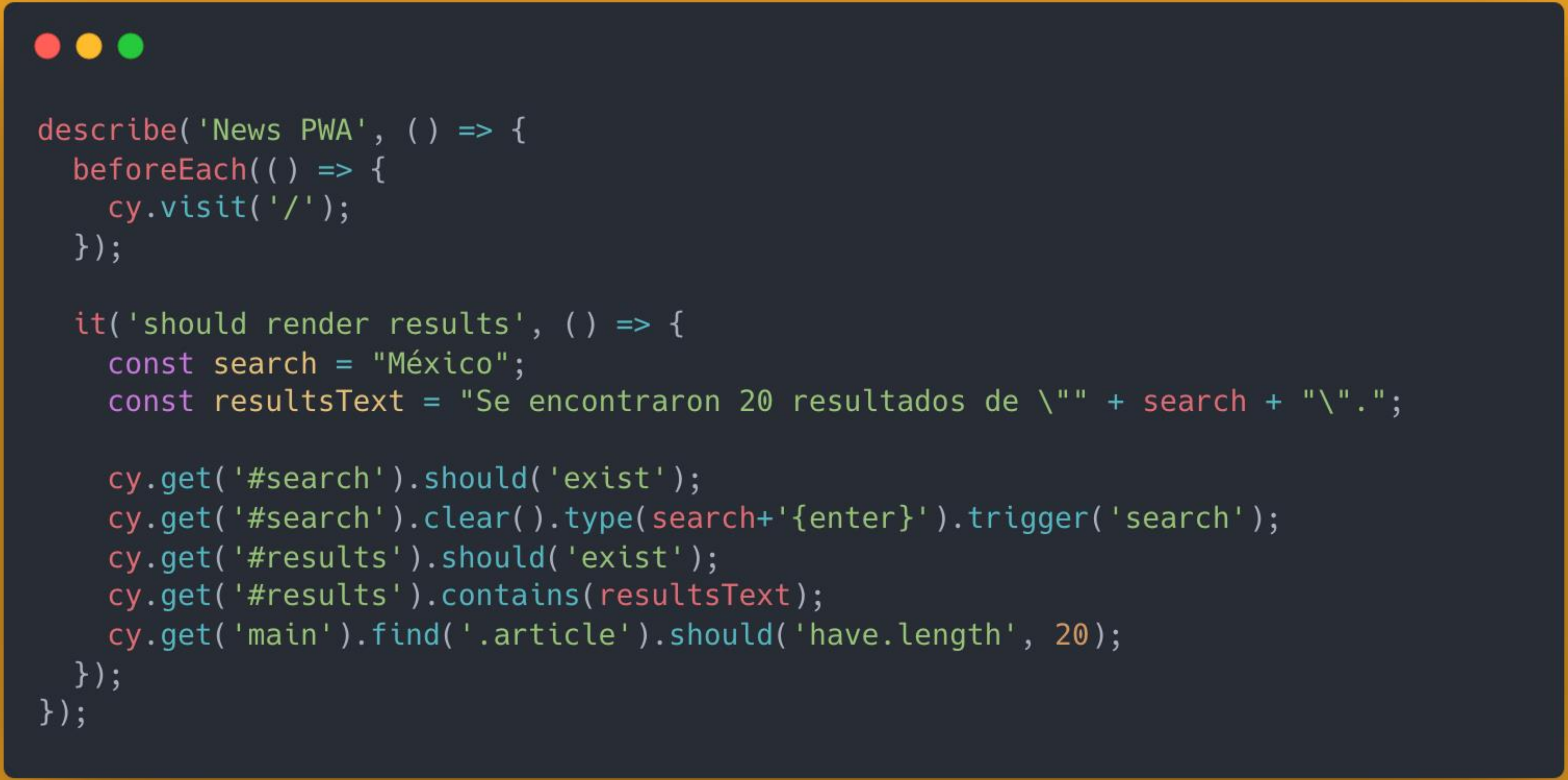


```
describe('when `"queryNews(param)"` is called', () => {
  it('should populate the `<main>` tag with articles', async () => {
    fetch.mockReturnValue(Promise.resolve(new Response(articleResponse)));

    document.body.innerHTML = `
        <header>
            <div class="search-bar">
                <input type="search" placeholder="Buscar..." name="Buscar" id="search" class="">
            </div>
        </header>
        <p id="results"></p>
        <main></main>
    `;

    await queryNews('buzzfeed');
    expect(fetch).toHaveBeenCalledTimes(1);
  });
});
```



```
describe('News PWA', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('should render 10 articles inside of a `"<div>"` element', () => {
    cy.get('main').find('.article').should('have.length', 10);
  });

  it('should render 10 articles when I interact with <select> tag', () => {
    cy.get('#sourceSelector').should('exist');
    cy.get('#sourceSelector').select('ANSA.it');
    cy.get('main').find('.article').should('have.length', 10);
  });
});
```

e2e tests



```javascript
describe('News PWA', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('should render results', () => {
    const search = "México";
    const resultsText = "Se encontraron 20 resultados de \"" + search + "\".";

    cy.get('#search').should('exist');
    cy.get('#search').clear().type(search+'{enter}').trigger('search');
    cy.get('#results').should('exist');
    cy.get('#results').contains(resultsText);
    cy.get('main').find('.article').should('have.length', 20);
  });
});
```

Q&A

Bibliography

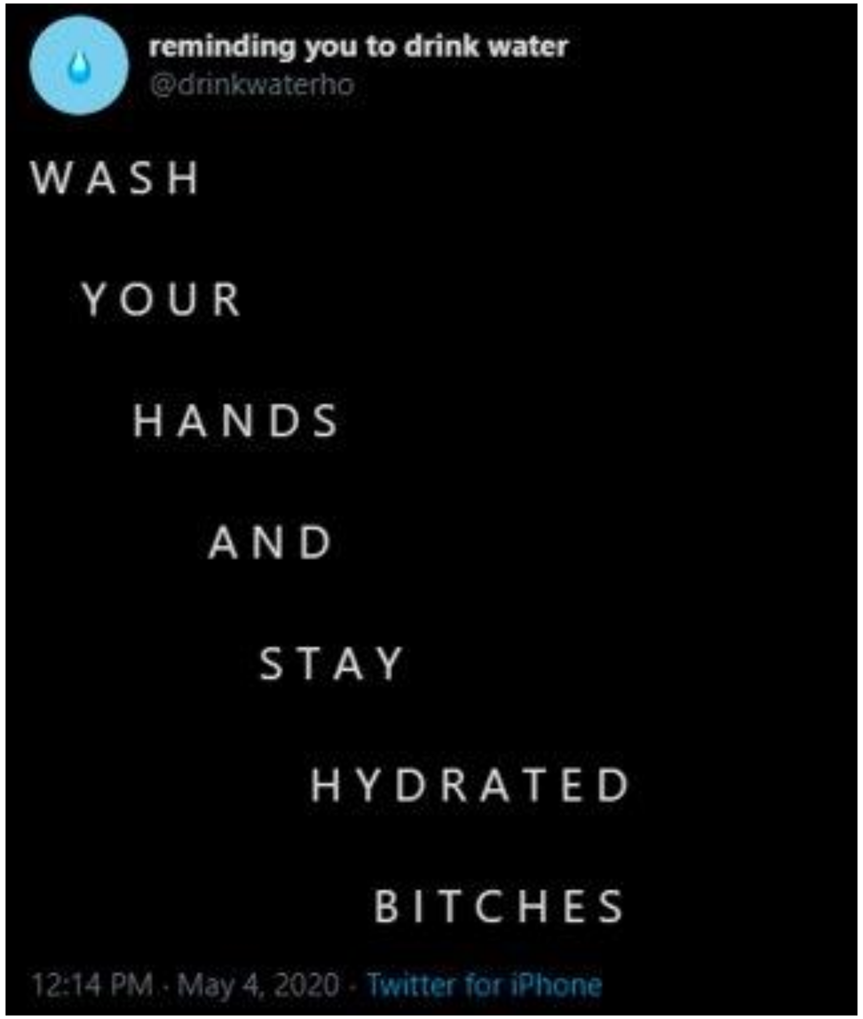
30 things every new software tester should learn. *Heather Reid*. Ministry of Testing, 2017: *https://www.ministryoftesting.com/dojo/lessons/30-things-every-new-software-tester-should-learn*

Defining unit testing, step by step. The Art of Unit Testing *(with examples in C#). Roy Osherove*. Manning Publications Co. 2014

Component Testing. Certified Tester. Foundation Level Syllabus. International Software Testing Qualifications Board. 2018.

8 benefits of Unit Testing. DZone. 2019: *https://dzone.com/articles/top-8-benefits-of-unit-testing*

What is End-to-End (E2E) Testing? All You Need to Know. Katalon. 2020: *https://www.katalon.com/resources-center/blog/end-to-end-e2e-testing/*



reminding you to drink water
@drinkwaterho

WASH

YOUR

HANDS

AND

STAY

HYDRATED

BITCHES

12:14 PM · May 4, 2020 · Twitter for iPhone

Thank you!