(/)

Curriculum
Short Specializations
Average: **58.31%**

0x06. Unittests in JS

UnitTests **Back-end** **JavaScript** **ES6** **NodeJS** **ExpressJS** **Mocha**

Weight: 1

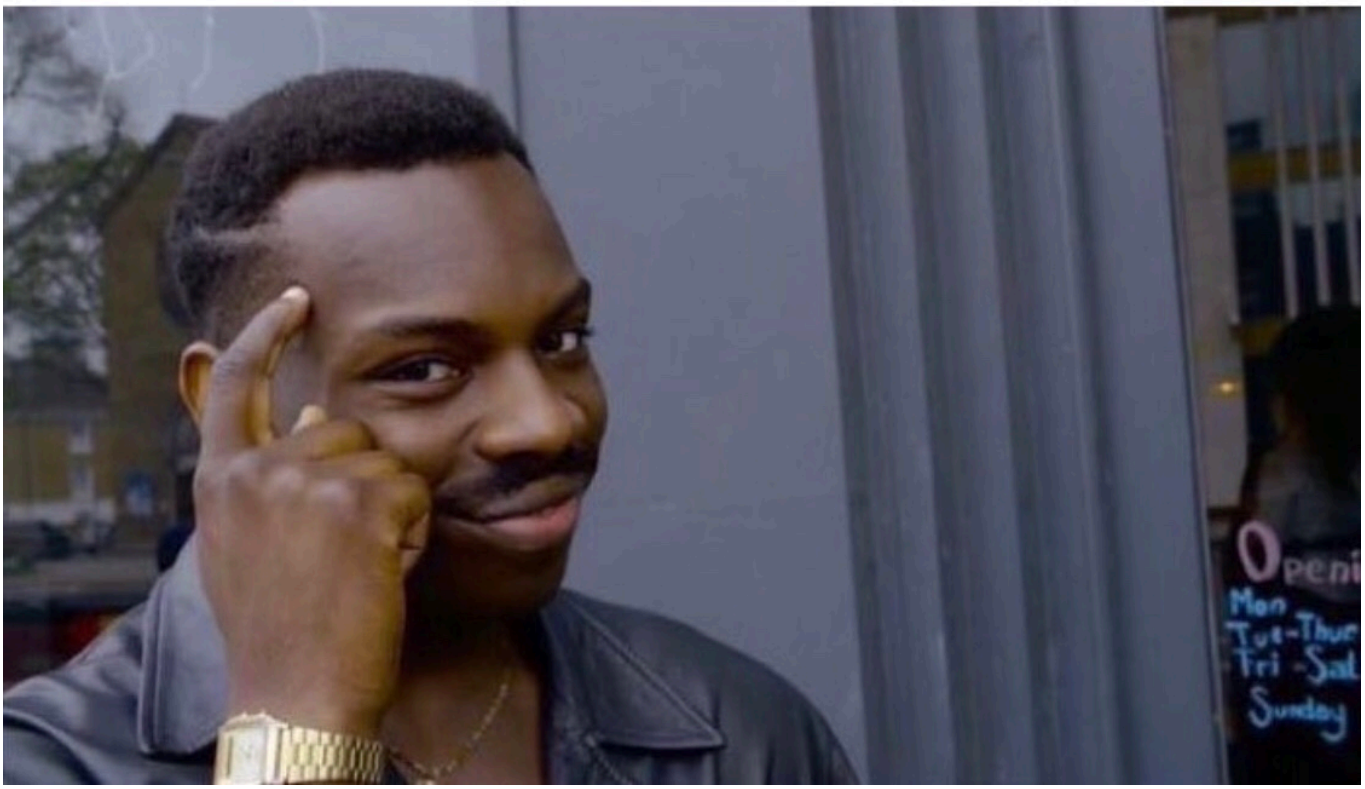
Project over - took place from Aug 21, 2024 4:00 AM to Aug 23, 2024 4:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 0.0/64 mandatory
- **Altogether: 0.0%**
  - Mandatory: 0.0%
  - Optional: no optional tasks

(/) You can't fail tests if you skip them



Resources

Read or watch:

- Mocha documentation (/rltoken/Gx5mfX41__cc2hwepcl0aA)
- Chai (/rltoken/Rs3SrSdr9OxPp-4099A0cg)
- Sinon (/rltoken/5KsW5N9sG3sGWW3z-jkNwA)
- Express (/rltoken/Jq58SNUh8jcZqKoFcuOQdw)
- Request (/rltoken/FcJfzr2jUJSj8Xp3z9L1wg)
- How to Test NodeJS Apps using Mocha, Chai and SinonJS (/rltoken/HwB8gViDosy8znk7H9i4Pw)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/Ge846tiklKJNUSNh60IR7w), **without the help of Google**:

- How to use Mocha to write a test suite
- How to use different assertion libraries (Node or Chai)
- How to present long test suites
- When and how to use spies
- When and how to use stubs
- What are hooks and when to use them
- Unit testing with Async functions
- How to write integration tests with a small node server

Requirements

- All of your code will be executed on Ubuntu 18.04 using Node 12.x.x
- Allowed editors: `vi` , `vim` , `emacs` , `Visual Studio Code`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `js` extension
- When running every test with `npm run test *.test.js` , everything should pass correctly without any warning or error

Tasks

0. Basic test with Mocha and Node assertion library

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Install Mocha using npm:

- Set up a scripts in your `package.json` to quickly run Mocha using `npm test`
- You have to use `assert`

Create a new file named `0-calcul.js` :

- Create a function named `calculateNumber` . It should accepts two arguments (number) `a` and `b`
- The function should round `a` and `b` and return the sum of it

Test cases

- Create a file `0-calcul.test.js` that contains test cases of this function
- You can assume `a` and `b` are always number
- Tests should be around the “rounded” part

Tips:

- For the sake of the example, this test suite is slightly extreme and probably not needed
- However, remember that your tests should not only verify what a function is supposed to do, but also the edge cases

Requirements:

- You have to use `assert`
- You should be able to run the test suite using `npm test 0-calcul.test.js`
- Every test should pass without any warning

Expected output

```
(/)
> const calculateNumber = require("./0-calcul.js");
> calculateNumber(1, 3)
4
> calculateNumber(1, 3.7)
5
> calculateNumber(1.2, 3.7)
5
> calculateNumber(1.5, 3.7)
6
>
```

Run test

```
bob@dylan:~$ npm test 0-calcul.test.js

> task_0@1.0.0 test /root
> ./node_modules/mocha/bin/mocha "0-calcul.test.js"

 calculateNumber
   ✓ ...
   ✓ ...
   ✓ ...
   ...

 130 passing (35ms)
bob@dylan:~$
```

Repo:

- GitHub repository: **alx-backend-javascript**
- Directory: **0x06-unittests_in_js**
- File: **package.json, 0-calcul.js, 0-calcul.test.js**

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

1. Combining descriptions

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Create a new file named `1-calcul.js` :

- Upgrade the function you created in the previous task (`0-calcul.js`)
- Add a new argument named `type` at first argument of the function. `type` can be `SUM` , `SUBTRACT` or `DIVIDE` (string)
- When type is `SUM` , round the two numbers, and add `a` and `b`
- When type is `SUBTRACT` , round the two numbers, and subtract `b` from `a`

- When type is `DIVIDE`, round the two numbers, and divide `a` with `b` - if the rounded value of `b` is equal to 0, return the string `Error`

(/)

Test cases

- Create a file `1-calcul.test.js` that contains test cases of this function
- You can assume `a` and `b` are always number
- Usage of `describe` will help you to organize your test cases

Tips:

- For the sake of the example, this test suite is slightly extreme and probably not needed
- However, remember that your tests should not only verify what a function is supposed to do, but also the edge cases

Requirements:

- You have to use `assert`
- You should be able to run the test suite using `npm test 1-calcul.test.js`
- Every test should pass without any warning

Expected output

```
> const calculateNumber = require("./1-calcul.js");
> calculateNumber('SUM', 1.4, 4.5)
6
> calculateNumber('SUBTRACT', 1.4, 4.5)
-4
> calculateNumber('DIVIDE', 1.4, 4.5)
0.2
> calculateNumber('DIVIDE', 1.4, 0)
'Error'
```

Repo:

- GitHub repository: `alx-backend-javascript`
- Directory: `0x06-unittests_in_js`
- File: `1-calcul.js, 1-calcul.test.js`

☐ Done? | Check your code | Ask for a new correction | >_ Get a sandbox | QA Review

2. Basic test using Chai assertion library

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

While using Node assert library is completely valid, a lot of developers prefer to have a behavior driven development style. This type being easier to read and therefore to maintain.

Let's install Chai with npm:

- Copy the file `1-calcul.js` in a new file `2-calcul_chai.js` (same content, same behavior)

- Copy the file `1-calcul.test.js` in a new file `2-calcul_chai.test.js`
- Rewrite the test suite, using `expect` from `Chai`

Tips:

- Remember that test coverage is always difficult to maintain. Using an easier style for your tests will help you
- The easier your tests are to read and understand, the more other engineers will be able to fix them when they are modifying your code

Requirements:

- You should be able to run the test suite using `npm test 2-calcul_chai.test.js`
- Every test should pass without any warning

Repo:

- GitHub repository: `alx-backend-javascript`
- Directory: `0x06-unittests_in_js`
- File: `2-calcul_chai.js, 2-calcul_chai.test.js`

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

3. Spies

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Spies are a useful wrapper that will execute the wrapped function, and log useful information (e.g. was it called, with what arguments). Sinon is a library allowing you to create spies.

Let's install Sinon with npm:

- Create a new file named `utils.js`
- Create a new module named `Utils`
- Create a property named `calculateNumber` and paste your previous code in the function
- Export the Utils module

Create a new file named `3-payment.js` :

- Create a new function named `sendPaymentRequestToApi` . The function takes two argument `totalAmount` , and `totalShipping`
- The function calls the `Utils.calculateNumber` function with type `SUM` , `totalAmount` as `a` , `totalShipping` as `b` and display in the console the message `The total is: <result of the sum>`

Create a new file named `3-payment.test.js` and add a new suite named `sendPaymentRequestToApi` :

- By using `sinon.spy` , make sure the math used for `sendPaymentRequestToApi(100, 20)` is the same as `Utils.calculateNumber('SUM', 100, 20)` (validate the usage of the `Utils` function)

Requirements:

- You should be able to run the test suite using `npm test 3-payment.test.js`

- Every test should pass without any warning
- You should use a `spy` to complete this exercise

Tips:

- Remember to always restore a spy after using it in a test, it will prevent you from having weird behaviors
- Spies are really useful and allow you to focus only on what your code is doing and not the downstream APIs or functions
- Remember that integration test is different from unit test. Your unit test should test your code, not the code of a different function

Repo:

- GitHub repository: **alx-backend-javascript**
- Directory: **0x06-unittests_in_js**
- File: **utils.js, 3-payment.js, 3-payment.test.js**

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

4. Stubs

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Stubs are similar to spies. Except that you can provide a different implementation of the function you are wrapping. Sinon can be used as well for stubs.

Create a new file `4-payment.js` , and copy the code from `3-payment.js` (same content, same behavior)

Create a new file `4-payment.test.js` , and copy the code from `3-payment.test.js`

- Imagine that calling the function `Utils.calculateNumber` is actually calling an API or a very expensive method. You don't necessarily want to do that on every test run
- Stub the function `Utils.calculateNumber` to always return the same number `10`
- Verify that the stub is being called with `type = SUM` , `a = 100` , and `b = 20`
- Add a spy to verify that `console.log` is logging the correct message `The total is: 10`

Requirements:

- You should be able to run the test suite using `npm test 4-payment.test.js`
- Every test should pass without any warning
- You should use a `stub` to complete this exercise
- Do not forget to restore the spy and the stub

Tips:

- Using stubs allows you to greatly speed up your test. When executing thousands of tests, saving a few seconds is important
- Using stubs allows you to control specific edge case (e.g a function throwing an error or returning a specific result like a number or a timestamp)

Repo:

- GitHub repository: **alx-backend-javascript**
- Directory: **0x06-unittests_in_js**
- File: **4-payment.js, 4-payment.test.js**

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

5. Hooks

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Hooks are useful functions that can be called before execute one or all tests in a suite

Copy the code from `4-payment.js` into a new file `5-payment.js` : (same content/same behavior)

Create a new file `5-payment.test.js` :

- Inside the same `describe` , create 2 tests:
  - The first test will call `sendPaymentRequestToAPI` with 100, and 20:
    - Verify that the console is logging the string `The total is: 120`
    - Verify that the console is only called once
  - The second test will call `sendPaymentRequestToAPI` with 10, and 10:
    - Verify that the console is logging the string `The total is: 20`
    - Verify that the console is only called once

Requirements:

- You should be able to run the test suite using `npm test 5-payment.test.js`
- Every test should pass without any warning
- You should use only one `spy` to complete this exercise
- You should use a `beforeEach` and a `afterEach` hooks to complete this exercise

Repo:

- GitHub repository: **alx-backend-javascript**
- Directory: **0x06-unittests_in_js**
- File: **5-payment.js, 5-payment.test.js**

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

6. Async tests with done

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Look into how to support async testing, for example when waiting for the answer of an API or from a Promise

Create a new file `6-payment_token.js` :

- Create a new function named `getPaymentTokenFromAPI`
- The function will take an argument called `success` (boolean)
- When `success` is true, it should return a resolved promise with the object `{data: 'Successful response from the API' }`
- Otherwise, the function is doing nothing.

Create a new file `6-payment_token.test.js` and write a test suite named `getPaymentTokenFromAPI`

- How to test the result of `getPaymentTokenFromAPI(true)` ?

Tips:

- You should be extremely careful when working with async testing. Without calling `done` properly, your test could be always passing even if what you are actually testing is never executed

Requirements:

- You should be able to run the test suite using `npm test 6-payment_token.test.js`
- Every test should pass without any warning
- You should use the `done` callback to execute this test

Repo:

- GitHub repository: `alx-backend-javascript`
- Directory: `0x06-unittests_in_js`
- File: `6-payment_token.js, 6-payment_token.test.js`

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

7. Skip

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

When you have a long list of tests, and you can't figure out why a test is breaking, avoid commenting out a test, or removing it. **Skip** it instead, and file a ticket to come back to it as soon as possible

You will be using this file, conveniently named `7-skip.test.js`

(7)

```
const { expect } = require('chai');

describe('Testing numbers', () => {
  it('1 is equal to 1', () => {
    expect(1 === 1).to.be.true;
  });

  it('2 is equal to 2', () => {
    expect(2 === 2).to.be.true;
  });

  it('1 is equal to 3', () => {
    expect(1 === 3).to.be.true;
  });

  it('3 is equal to 3', () => {
    expect(3 === 3).to.be.true;
  });

  it('4 is equal to 4', () => {
    expect(4 === 4).to.be.true;
  });

  it('5 is equal to 5', () => {
    expect(5 === 5).to.be.true;
  });

  it('6 is equal to 6', () => {
    expect(6 === 6).to.be.true;
  });

  it('7 is equal to 7', () => {
    expect(7 === 7).to.be.true;
  });
});
```

Using the file `7-skip.test.js` :

- Make the test suite pass **without** fixing or removing the failing test
- `it` description **must stay** the same

Tips:

- Skipping is also very helpful when you only want to execute the test in a particular case (specific environment, or when an API is not behaving correctly)

Requirements:

- You should be able to run the test suite using `npm test 7-skip.test.js`
- Every test should pass without any warning

Repo:

- GitHub repository: alx-backend-javascript
- Directory: 0x06-unittests_in_js
- File: 7-skip.test.js

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

8. Basic Integration testing

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

In a folder `8-api` located at the root of the project directory, copy this `package.json` over.

```
{
  "name": "8-api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "./node_modules/mocha/bin/mocha"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "mocha": "^6.2.2",
    "request": "^2.88.0",
    "sinon": "^7.5.0"
  }
}
```

Create a new file `api.js` :

- By using `express` , create an instance of `express` called `app`
- Listen to port 7865 and log `API available on localhost port 7865` to the browser console when the `express` server is started
- For the route `GET /` , return the message `Welcome to the payment system`

Create a new file `api.test.js` :

- Create one suite for the index page:
  - Correct status code?
  - Correct result?
  - Other?

Server

Terminal 1

(7)

```
bob@dylan:~/8-api$ node api.js
API available on localhost port 7865
```

Terminal 2

```
bob@dylan:~/8-api$ curl http://localhost:7865 ; echo ""
Welcome to the payment system
bob@dylan:~/8-api$
bob@dylan:~/8-api$ npm test api.test.js

> 8-api@1.0.0 test /root/8-api
> ./node_modules/mocha/bin/mocha "api.test.js"



  Index page
    √ ...
    √ ...
    ...

  23 passing (256ms)

bob@dylan:~/8-api$
```

Tips:

- Since this is an integration test, you will need to have your node server running for the test to pass
- You can use the module **request**

Requirements:

- You should be able to run the test suite using **npm test api.test.js**
- Every test should pass without any warnings

Repo:

- GitHub repository: **alx-backend-javascript**
- Directory: **0x06-unittests_in_js**
- File: **8-api/package.json, 8-api/api.js, 8-api/api.test.js**

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

9. Regex integration testing

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

In a folder **9-api** , reusing the previous project in **8-api** (**package.json** , **api.js** and **api.test.js**)

Modify the file api.js :

- Add a new endpoint: `GET /cart/:id`
- `:id` must be only a number (validation must be in the route definition)
- When access, the endpoint should return `Payment methods for cart :id`

(/)

Modify the file `api.test.js` :

- Add a new test suite for the cart page:
  - Correct status code when `:id` is a number?
  - Correct status code when `:id` is NOT a number (=> 404)?
  - etc.

Server

Terminal 1

```
bob@dylan:~$ node api.js
API available on localhost port 7865
```

Terminal 2

(7)

```
bob@dylan:~$ curl http://localhost:7865/cart/12 ; echo ""
Payment methods for cart 12
bob@dylan:~$
bob@dylan:~$ curl http://localhost:7865/cart/hello -v
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 7865 (#0)
> GET /cart/hello HTTP/1.1
> Host: localhost:7865
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 404 Not Found
< X-Powered-By: Express
< Content-Security-Policy: default-src 'none'
< X-Content-Type-Options: nosniff
< Content-Type: text/html; charset=utf-8
< Content-Length: 149
< Date: Wed, 15 Jul 2020 08:33:44 GMT
< Connection: keep-alive
<
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot GET /cart/hello</pre>
</body>
</html>
* Connection #0 to host localhost left intact
bob@dylan:~$
```

Tips:

- You will need to add a small regex in your path to support the usecase

Requirements:

- You should be able to run the test suite using **npm test api.test.js**
- Every test should pass without any warning

Repo:

- GitHub repository: **alx-backend-javascript**
- Directory: **0x06-unittests_in_js**
- File: **9-api/api.js, 9-api/api.test.js, 9-api/package.json**

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

10. Deep equality & Post integration testing (7)

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

In a folder `10-api`, reusing the previous project in `9-api` (`package.json`, `api.js` and `api.test.js`)

Modify the file `api.js`:

- Add an endpoint `GET /available_payments` that returns an object with the following structure:

```
{
  payment_methods: {
    credit_cards: true,
    paypal: false
  }
}
```

- Add an endpoint `POST /login` that returns the message `Welcome :username` where `:username` is the value of the body variable `userName`.

Modify the file `api.test.js`:

- Add a test suite for the `/login` endpoint
- Add a test suite for the `/available_payments` endpoint

Server

Terminal 1

```
bob@dylan:~$ node api.js
API available on localhost port 7865
```

Terminal 2

```
bob@dylan:~$ curl http://localhost:7865/available_payments ; echo ""
{"payment_methods":{"credit_cards":true,"paypal":false}}
bob@dylan:~$
bob@dylan:~$ curl -XPOST http://localhost:7865/login -d '{ "userName": "Betty" }' -H 'Content-Type: applicati
on/json' ; echo ""
Welcome Betty
bob@dylan:~$
```

Tips:

- Look at deep equality to compare objects

Requirements:

- You should be able to run the test suite using `npm test api.test.js`
- Every test should pass without any warning
- Your server should not display any error

Repo:

- GitHub repository: alx-backend-javascript
- Directory: 0x06-unittests_in_js
- File: 10-api/api.js, 10-api/api.test.js, 10-api/package.json

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review