Learn Jest & Expect

Learn jest and expect

Quick Links:

- ❤️**Vitest Notes and Example Usage Google Doc**: Click here
- ❤️ with-jest-supertest-axios: learn-express/with-jest-supertest-axios
- **Jest Cheatsheet:** Click here
- **react testing:** Click here
- ❤️**Awesome Medium Article about Jest Mocks suggested by Eric**: Click here ❤️
  - (✅**PRINTED simply via Chrome's print to pdf — then print in booklet mode via Adobe Acrobat Reader)**
  - ❤️**:** ^^ Above article mentions this — Justin Searls – Please don't mock me (seems awesome)
- ❤️❤️**Martin Fowler - Test Double: Dummy, Fake, Stubs, Spies, Mocks -** ❤️Click here
  - Source: Stackoverflow Question
- **Miscellaneous:**
  - ChatGPT
    - Who was the first person to write tests for code and when?
    - Who wrote tests in higher levels of languages?

Syntax

```
expect(received).toBe(expected)
```

❤️Bare jest setup with just `expect` and typescript setup

Step 1: Run this command:

```
mkdir my-jest-app; cd my-jest-app; npm init -y; mkdir __tests__ && echo "test('simple', () => expect(1).toBe(1))" > __tests__/a.ts && npm i -D expect jest @types/jest
```

Step 2: Add script to `package.json` file: `"test": "jest --watchAll"`
Step 3: Now you can use the `npm run test` command to start testing.

IMPORTANT: For typescript support: docs, you need to install deps: `npm i -D ts-jest @babel/preset-env @babel/preset-typescript` and create a file in root project i.e, `babel.config.js` with below contents:

```
module.exports = {
      presets: [
        ['@babel/preset-env', {targets: {node: 'current'}}],
        '@babel/preset-typescript',
      ],
  };
```

❤️Solution to Error - A worker process has failed to exit gracefully…

Error - A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

Solution: Make sure you do not have any code like below on the top-level in any of you javascript/typescript file(s):

```
setInterval(() => { }, 2_000)
```

Can we have multiple `beforeAll(..)` in jest? (ChatGPT)

tldr; **YES!** (Source: Click here)
Also, I implemented this solution in the qr project so please refer to that for an easy reusability function for this.

Remove default console.log format of jest

Check this issue on jest's github: Click here

Jest debugging command from official docs:

```
node --inspect-brk node_modules/.bin/jest --runInBand [any other arguments here]
```

❤️ `.toEqual` Use this as much as possible (or even better `.toStrictEqual`)

⭐*For Luxon Notes, please refer the other tab named "Learn Luxon".*

```
// For services:
expect(...).toEqual({
      _id: expect.any(mongoose.Types.ObjectId),
      createdAt: expect.any(Date),
      updatedAt: expect.any(Date),
})

// For e2e (HTTP API responses):
expect(...).toEqual({
      _id: expect.stringMatching(SIMPLE_MONGODB_ID_REGEX),
```

```
      createdAt: expect.any(String),
      updatedAt: expect.any(String),
      startDate: '2022-10-24T00:00:00.000Z',
      endDate: '2022-10-25T23:59:59.000Z',
})
```

1. Difference between `toEqual` and `toStrictEqual` (ChatGPT)

Click here

`toEqual`

```
test('toEqual ignores undefined properties', () => {
  expect({ a: 1, b: undefined }).toEqual({ a: 1 });
});
✅ Test passes because toEqual ignores undefined properties.
```

`toStrictEqual`

Example 1:

```
test('toStrictEqual does not ignore undefined properties', () => {
  expect({ a: 1, b: undefined }).toStrictEqual({ a: 1 });
});
❌ Test fails because b: undefined is explicitly different.
```

Example 2:

```
class Person {
  constructor(name) {
    this.name = name;
  }
}

test('toStrictEqual checks class instances', () => {
  expect(new Person('Alice')).toStrictEqual({ name: 'Alice' });
});

❌ Test fails because { name: 'Alice' } is a plain object, while new
Person('Alice') is an instance of Person.
```

So, use toStrictEqual when you want precise comparisons, including class instances and undefined properties. Use toEqual when you need looser deep equality.

`toMatchobject` - Used to check that a JavaScript object matches a subset of the properties of an object (docs).

Summary: When NOT to Use `toMatchObject` *(Source: ChatGPT)*

❌ Avoid `toMatchObject` When...	✅ Use Instead
Exact object comparison is required	toStrictEqual
Checking for missing/extra properties	toStrictEqual
Matching objects inside arrays	toContainEqual
Comparing objects with undefined values	toStrictEqual
Verifying class instances	toBeInstanceOf

2. More Examples

Following notes covers these methods:

- `expect(response).toEqual(..)` **along with**
  - `expect.stringMatching(SIMPLE_MONGODB_ID_REGEX)`
  - `expect.stringContaining('sahil')`
  - `expect.not.stringContaining('om')`

Tip: If you don't have string matches you can store sample data in a `.json` file as well.

```
import expect from 'expect'
const SIMPLE_MONGODB_ID_REGEX = /^[a-f\d]{24}$/i // 24 characters ~Sahil
const response = [
      {
            _id: '63ab12b2fa6c5356d22a298d',
            firstName: 'Nelson Mandela',
      },
]

// Learn Usage of
// ==============
// 1. .toEqual(...) with expect.stringMatching
expect(response).toEqual([
      {
            _id: expect.stringMatching(SIMPLE_MONGODB_ID_REGEX),
            firstName: 'Nelson Mandela',
      },
])

// 2. .toEqual(...) with expect.stringMatching(...)
expect([{name: 'sahil rajput'}]).toEqual([
```

```
      {
            name: expect.stringContaining('sahil'),
      },
])

// 3. .toEqual(...) with expect.not.stringContaining(...)
expect([{name: 'sahil rajput'}]).toEqual([
      {
            name: expect.not.stringContaining('om'),
      },
])
```

🗑️File deletion and exists check with jest

```
it.only('if given path is exists then it will return that same path', async () => {
  function runTest() {
    const tempFilePath = path.join(os.tmpdir(), 'myfile.jpg');
    const fh = fs.openSync(tempFilePath, 'w');
    fs.writeSync(fh, '.', 3);
    fs.closeSync(fh);
    expect(existsSync(tempFilePath)).toBe(true);
    unlinkSync(tempFilePath);
    expect(existsSync(tempFilePath)).toBe(false);
  }

  for (let i = 0; i < 10_000; i += 1) {
    console.log('iter:', i);
    runTest();
  }
});
```

What's the difference between `foo.spec.ts` and `foo.test.ts`?

Source: Click here

It is just a "matter of taste" as you describe it. Just a way to group your test files, so that the test runners know what files to load / look in for test methods.

(If you look at the documentation for runner like Jasmine and mocha. You configure "file globs" to tell them how to find the files to run. Some thing like `/**/*.test.ts`)

In your projects, you could configure anything you want. (Or even just move all your test code to a "test" folder, and not have any special file name conventions)

The `*.test.ts` and `*.spec.ts` are just common conventions recommended by the different testing frameworks.

❤️Mock Functions

1. Mock Functions - `mockImplementationOnce`

Source (Jest Docs): Click here

```
const myMockFn = jest
  .fn(() => 'default')
  .mockImplementationOnce(() => 'first call')
  .mockImplementationOnce(() => 'second call');

console.log(myMockFn(), myMockFn(), myMockFn(), myMockFn());
// > 'first call', 'second call', 'default', 'default'
```

2. Mock Function - Mock Implementations

Source (Jest Docs): Click here

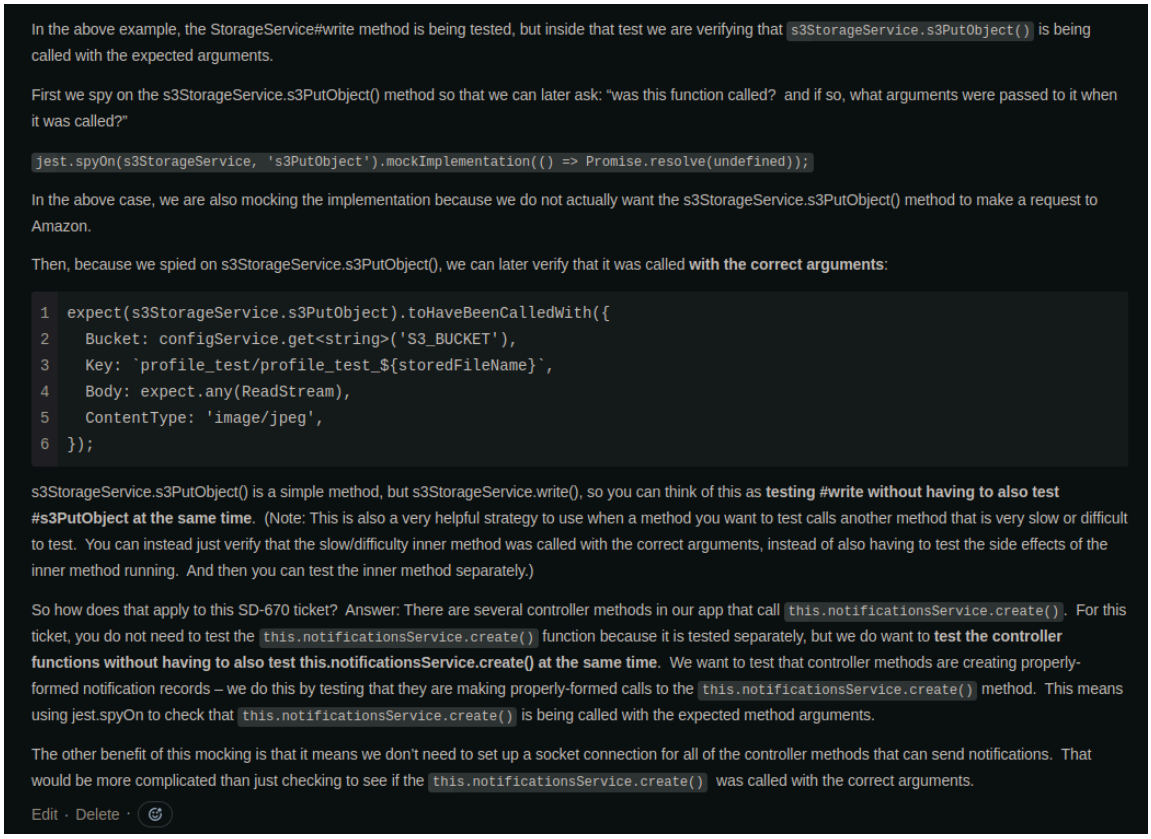
```
jest.mock('../foo'); // this happens automatically with automocking
const foo = require('../foo');

// foo is a mock function
foo.mockImplementation(() => 42);
foo();
// > 42
```

3. Mock Functions - Mocking a method of Object (class instance) via `mockImplementation`

Documentation Links:

- ❤️Medium Article suggested by Eric: Click here ❤️
- ❤️Docs - Mock Functions: Click here
- Docs - Using with MongoDB with jest using **inmemory-mongodb** Setup: Click here
  - npm: **jest-mongodb** (uses a mongodb memory server npm library - **mongodb-memory-server**)

❤️**From Eric in Sla****:**

- 

We cover following methods in below notes:

- `jest.spyOn(..., ...).mockImplementation(() => Promise.resolve(undefined))`
  - ❤️Learn usage of **`spyOn`** with mock implementation: Check file — `qr-solution-backend/__tests__/e2e/whatsapp-bot.test.ts`
- `expect(fn).toHaveBeenCalledWith(...)`

4. Mock Functions - `toHaveBeenNthCalledWith`

- Inspiration: Click here (includes the link to below referred `expect` docs.
- Source of below text - `expect` Docs: Click here

We discuss following in below notes:

- `expect(fn).toHaveBeenNthCalledWith(fnArguments)`

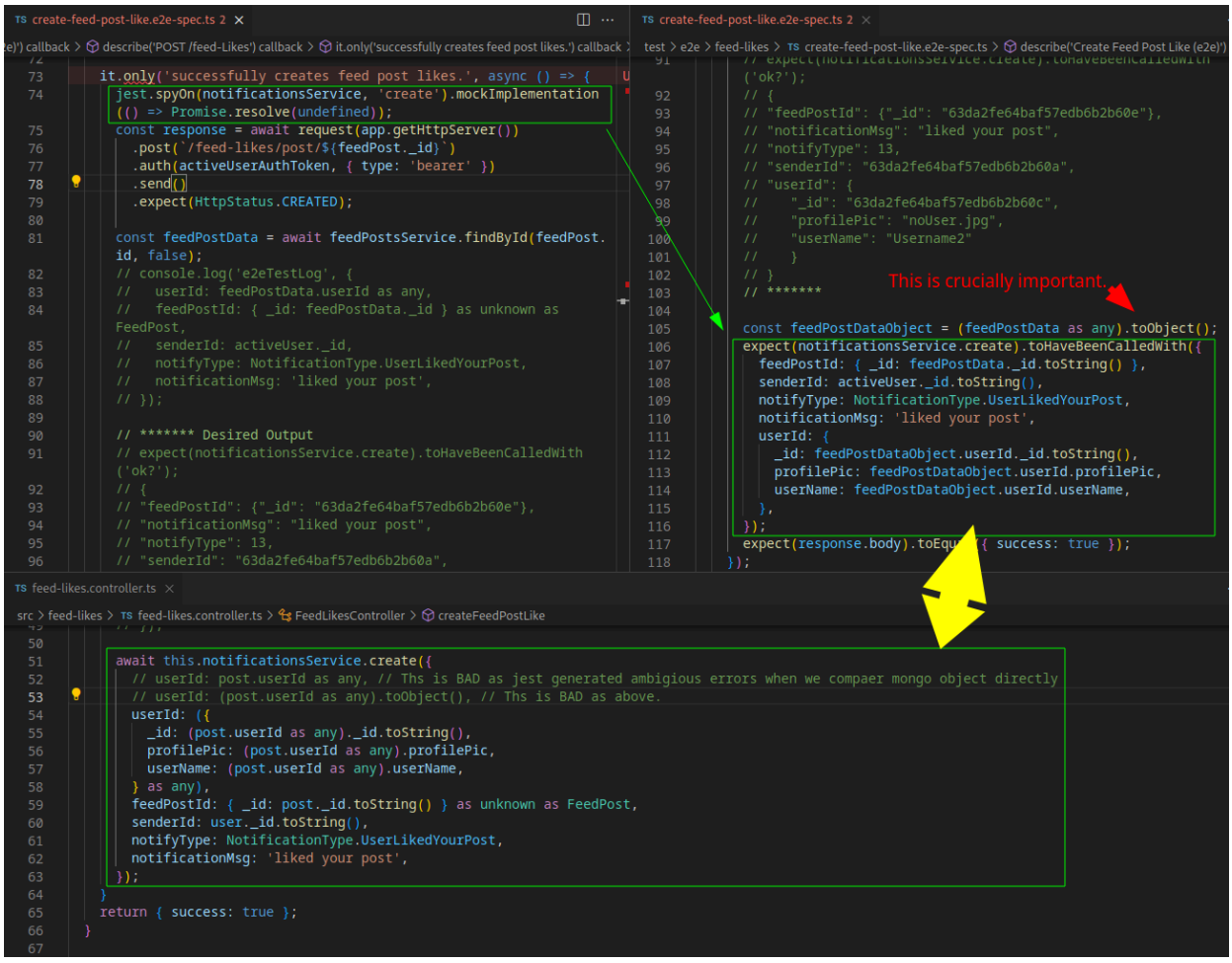
`.toHaveBeenNthCalledWith(nthCall, arg1, arg2, ....)` *[Also under the alias: `.nthCalledWith(nthCall, arg1, arg2, ...)`]*

If you have a mock function, you can use .toHaveBeenNthCalledWith to test what arguments it was nth called with. For example, let's say you have a drinkEach(drink, Array) function that applies f to a bunch of flavors, and you want to ensure that when you call it, the first flavor it operates on is 'lemon' and the second one is 'octopus'. You can write:

```
test('drinkEach drinks each drink', () => {
  const drink = jest.fn();
  drinkEach(drink, ['lemon', 'octopus']);
  expect(drink).toHaveBeenNthCalledWith(1, 'lemon');
  expect(drink).toHaveBeenNthCalledWith(2, 'octopus');
});
```

Note: The nth argument must be a positive integer starting from 1.

From project:



❤️Testing any callback operation with jest using done

```
describe.only('something', () => {

  it.only('should it be?', (done) => {
```

```
    console.log('hello');
    done() // if done is not called then the test will not pass at all. In fact the setTimeout warning is thrown.
  });
});
```

❤️Socket.io Testing with jest

Official Docs - [Click here]

Function to wait for a callback function but reject after 2 seconds if it didn't receive the event Socket.io

Inspiration: `slas***-web-new/test/helpers/gateway-test-helpers.ts`

```
export async function waitForAuthSuccessMessage(client: Socket) {
  return new Promise<void>((resolve, reject) => {
    const timeout = setTimeout(() => {
     reject(new Error('Did not receive authSuccess message in a reasonable amount of time'));
    }, AUTH_SUCCESS_WAIT_TIMEOUT);
    client.once('authSuccess', (payload) => {
      expect(payload).toEqual({ success: true });
      clearTimeout(timeout);
      resolve();
    });
  });
}
```

Usage of above function in any test:

```
const client = io(baseAddress, { auth: { token: activeUserAuthToken }, transports: ['websocket'] });
await waitForAuthSuccessMessage(client); // <<< This....
// ... more statements here
```

❤️Test Timeout

1. Define test timeout for a individual test like that

```
it('does a lot of stuff exceeding default 5 sec timeout', async () => {
  // ...
}, 10_000)
```

2. You can define jest config (e.g., test timeout for particular file) for a single file basis by that

```
jest.setTimeout(20_000)
```

Learn Fishery (factory)

Learn Fishery

https://github.com/thoughtbot/fishery

❤️What is the use of transient params in fishery?

In **Fishery** (a factory library for JavaScript/TypeScript used to generate test data), **transient params** are temporary values that you can pass to the factory when building an object. These values:

- **Affect how the object is built**, but
- **Do not appear in the final object**.

They are useful when you want to modify the behavior of the factory for a specific case without changing the resulting data structure.

Example:

```
import { Factory } from 'fishery';

const userFactory = Factory.define(({ transientParams }) => {
  const { withAdminRole = false } = transientParams;

  return {
    name: 'John Doe',
    role: withAdminRole ? 'admin' : 'user',
  };
});
```

Usage:

```
// Normal user
userFactory.build();
// => { name: 'John Doe', role: 'user' }

// Admin user using transient param
userFactory.build({ transient: { withAdminRole: true } });
// => { name: 'John Doe', role: 'admin' }
```

Note: Here, the `withAdminRole` param **only influences** the returned object; it is not included in the output.

Summary:

- `transientParams` are **temporary, optional** inputs.

- Used to **control logic** inside the factory.
- They are **not included** in the built object.

♥ Learn Luxon

Learn Luxon

Quick Links:

- ❤️**Luxon - Full Sheet: Click here**
- 🚀Learn Date in Javascript: Click here
- moment vs. luxon on npmtrends.com: Click here

🚀***Tip:*** *You can run below commands in browser console on site: https://moment.github.io/luxon/*

```
// creating date from luxon
const now = luxon.DateTime.now();

// ❤️ Creating ISO string from luxon date (way 1/2)
console.log(DateTime.now().toUTC().toISO()); // Output
"2025-08-06T17:44:10.123Z"

// ❤️ Creating ISO string from luxon date (way 2/2)
console.log(now.toJSDate().toISOString()) // OUTPUT:
'2023-10-24T16:34:17.155Z'

// ❤️ Creating luxon date from ISO string
luxon.DateTime.fromISO('2022-10-17T00:00:00Z')
```

Duration

```
// const { Duration } = require('luxon');

// Create a Duration from a number of seconds
const numberOfSeconds = 3600; // 1 hour
const duration1 = luxon.Duration.fromObject({ seconds: numberOfSeconds });
console.log(duration1.toFormat("h 'hours,' m 'minutes,' s 'seconds'"));
// OUTPUT: 1 hours, 0 minutes, 0 seconds


// Create a Duration from difference b/w two dates
const startTime = luxon.DateTime.now();
// Do some computations....
const now = luxon.DateTime.now();
const duration2 = now.diff(startTime);
console.log(duration2.toFormat("h 'hours,' m 'minutes,' s 'seconds'"));
// OUTPUT: 0 hours, 0 minutes, 7 seconds
```

Weekday

```
// Get the day of the week. 1 is Monday and 7 is Sunday
DateTime.(...)
            .weekday
// OUTPUT: 1,2,3,4,5,6,7
```

Time zone, plus, endOf:

```
// Others
DateTime.(...)
            .setZone("Europe/Berlin") // to get time in a particular timezone
            .plus({month: 1}) // to add certain days
            .endOf('month') // to get end of that month
```

For usage with jest

```
// For services:
expect(...).toEqual({
      _id: expect.any(mongoose.Types.ObjectId),
      createdAt: expect.any(Date),
      updatedAt: expect.any(Date),
})

// For e2e (HTTP API responses):
expect(...).toEqual({
      _id: expect.stringMatching(SIMPLE_MONGODB_ID_REGEX),
      createdAt: expect.any(String),
      updatedAt: expect.any(String),
      startDate: '2022-10-24T00:00:00.000Z',
      endDate: '2022-10-25T23:59:59.000Z',
})
```

❤️**We store values in db via luxon in project sla***, for field `startDate`, we do**

```
// Learn: Both '2022-10-24T00:00:00Z' (input) and '2022-10-24T00:00:00.000Z' (output)  are
valid UTC, but the second has higher precision i.e., the extra .000 before Z (milliseconds).
console.log(luxon.DateTime.fromISO('2022-10-24T00:00:00Z').toJSDate().toISOString()) // Way1
// '2022-10-24T00:00:00.000Z' // date_1
console.log(luxon.DateTime.fromISO('2022-10-24T00:00:00Z').toUTC().toISO()) // Way2
// '2022-10-24T00:00:00.000Z'
```

Similarly for `endDate`:

```
console.log(luxon.DateTime.fromISO('2022-10-25T23:59:59Z').toJSDate().toISOString())
// '2022-10-25T23:59:59.000Z' // ❤️NOTE: The extra .000 before Z is milliseconds
```