





Marie Drake

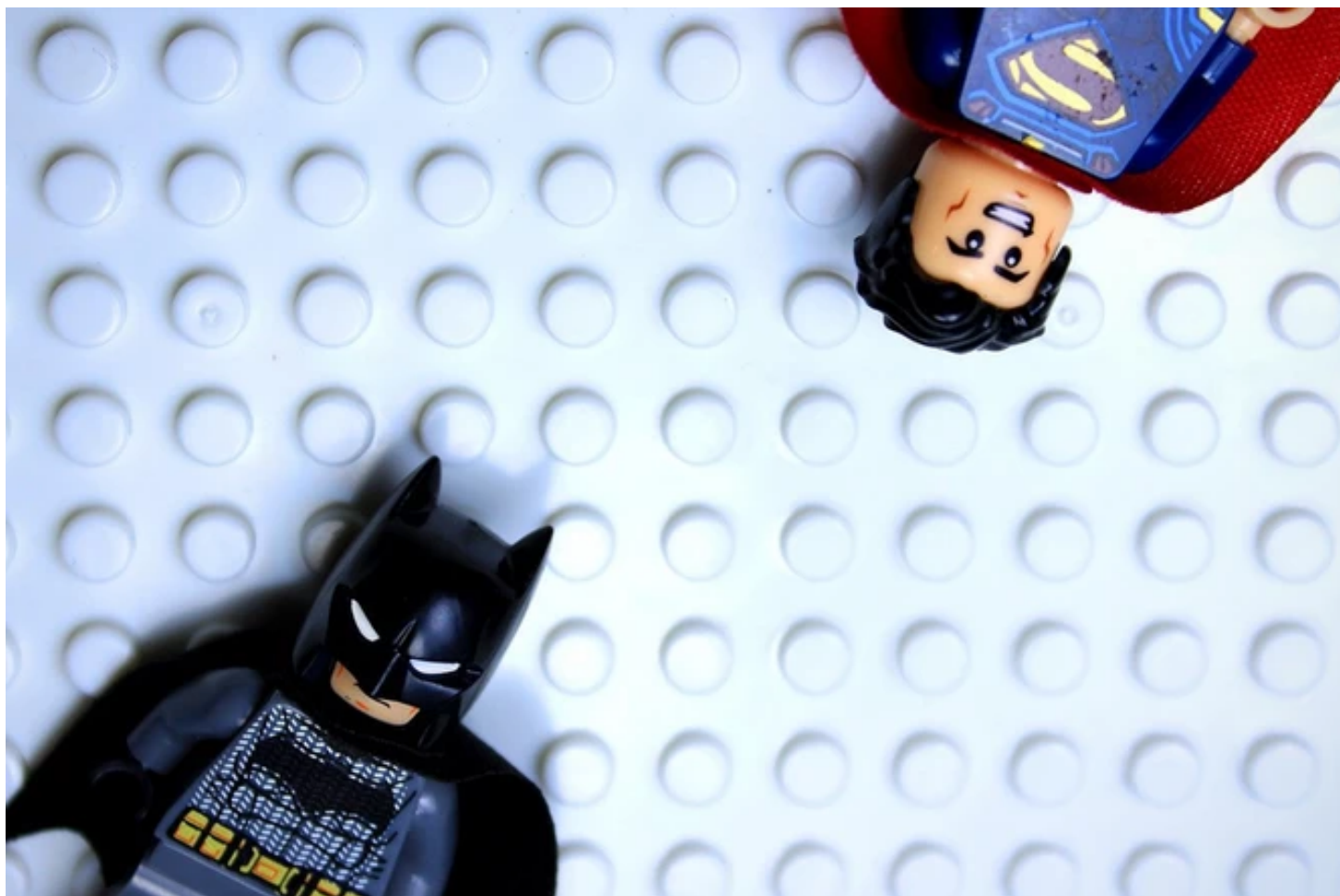


Jul 2, 2021

3 min read

# API Testing using Jest and SuperTest

Updated: Jan 27

Image by [Adithya Rajeev](#) from

Last week, I've been playing around with how to integrate SuperTest, a library for testing HTTP servers. With over 2 million weekly downloads, it seems to be a standard choice for testing our backend API. Previously, I have blogged on how to use Cypress for API testing. While I could have use this to test our API, I'm a big advocate of using the right tool within the right context. As our team is all focused on backend work, it doesn't make sense to introduce a front end testing tool like Cypress to test our APIs. In this blog post, I will show how you can integrate SuperTest with Jest to test your APIs.

## Installing Dependencies

The first step that we need to do as always is to install the dependencies that we need. For this, you need to add the following to your project's dev dependencies. If you are using TypeScript, you might need to install

add the following to your project's dev dependencies. If you are using typescript, you might need to install the type definitions as well.

```
npm i -D jest supertest
```

```
// if using TypeScript
```

```
npm i -D jest supertest @types/jest @types/supertest typescript
```

## Babel Setup

In order to use the latest features of JavaScript, let's quickly set up Babel. Babel allows us to convert new ES6 syntax into older versions of JavaScript that are backwards compatible. Since I'll be using new syntax like the "import" statement on my test, this setup is needed.

For this specific Babel setup, I have chosen to use the [@babel/preset-env plugin](#). This preset contains all the additional plugins that we would need to transpile all of our ES6 features.

Let's again add it to our dev dependencies.

```
npm i -D @babel/preset-env
```


Then, create a `.babelrc` file and add the following setup. This setup will let us compile our code based on the current node version that we are using.

```
{
  "presets": [
    [
      "@babel/preset-env", {
        "targets": {
          "node": "current"
        }
      ]
    ]
  ]
}
```

If this is still confusing and you want to know more about this plugin, I highly recommend giving their [documentation](#) a read through.

## Writing our First Test

The next step is to structure our first test. For the first test, it's a simple `get` request to this endpoint `[todos/1](#)` and verifying that the status code is 200 as shown in the code snippet below.



```
import request from "supertest";

const baseUrl =
  'https://jsonplaceholder.typicode.com/';

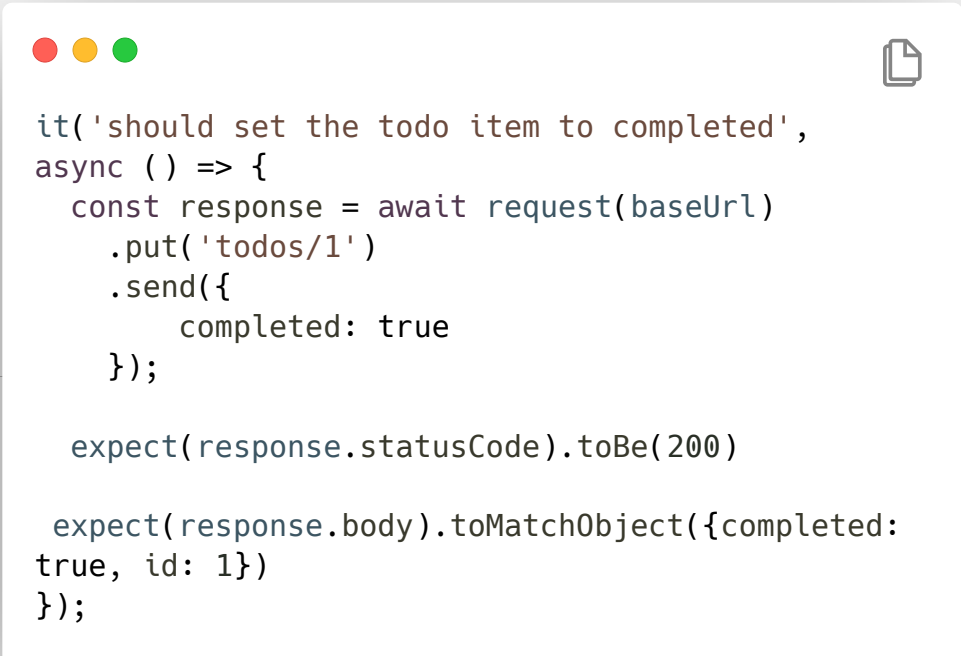
describe('Todos endpoint', () => {
  it('should return a 200 status code',
  async () => {
    const response = await
    request(baseUrl)
      .get('todos/1');

    expect(response.statusCode).toBe(200);
  });
});
```

SuperTest has a function called `request` which is used to initialise the API call based on what's being passed to it. Our base url here is the [jsonplaceholder fake api](#) which I've hard coded for simplicity. Afterwards, I'm simply calling the `get` function to simulate a get request to `todos/1` endpoint. The async/await keyword is used to make our request synchronous and wait for the request to complete before doing the assertion that the status code should be 200.

## Marking Todo Item as Completed

Now, let's try to mark a todo item as completed and show how we can achieve that with SuperTest. For this test, we want to set completed to have the value of true when we do a `put` request to the same endpoint and then check that it's successful by asserting the status code to be 200 and verifying that the response body matches our expectation.



```
it('should set the todo item to completed',
  async () => {
    const response = await request(baseUrl)
      .put('todos/1')
      .send({
        completed: true
      });

    expect(response.statusCode).toBe(200)

    expect(response.body).toMatchObject({completed:
true, id: 1})
  });
```

Rather than calling the `get` function, we are now using the `put` function to simulate a put request to `todos/1` endpoint. Next, I'm calling the `send` function and passing in an object that will set completed to true.

## Wrapping Up

That's pretty much it! SuperTest is a very easy to use library for testing APIs whether it's REST based or GraphQL. The examples above are pretty basic but should help you get started. Apart from doing a `get` or `put` request, you can also invoke other HTTP actions, set authorisation token, cookies or any custom headers. For more examples on how to use the library, check out [SuperTest documentation](#).

All the code snippets above can be seen on my [supertest-demo](#) github repository so feel free to clone or download it 😊