Snapshot Testing Your GraphQL API

Easily achieve large coverage of your API through integration tests using Jest snapshots in Node.js

Warren Day · Follow

Published in Better Programming · 3 min read · Aug 6, 2020

Photo by Ferenc Almasi on Unsplash.

Snapshot testing was popularized by frameworks like Jest in order to quickly and effectively track unintended changes to React components. However, the concept can be taken away from the front end and find equal utility for back-end testing.

In this article, I'll show you how to easily achieve large coverage of your API through integration tests using Jest snapshots in Node.js.

In the next few sections, we'll cover:

- The benefits of snapshot testing
- How to set up a testing environment
- Managing snapshots tests over time

So let's get started.

. . .

Benefits of Snapshot testing

Testing often has a sweet spot. The goal is rarely 100% coverage, but you should aim to use the right kind of testing and find a balance between fluidity and stability.

Unit testing is king when covering core parts of your application, and I'm not advocating that snapshots should be used here.

Rather their utility is better suited for integration testing and asserting that a given request returns a given response. Through doing this, you will gain wide coverage of all your resolvers while easily being able to change tests as your application grows.

. . .

Setting Up a Test Environment

We need a way to send requests to our GraphQL server and capture the response as a snapshot. I am using apollo-server-testing to do this, but you could use something like supertest instead.

To make fewer assumptions about your specific project, I will show an agnostic approach to snapshot testing. In reality, you would want to seed/tear down databases and mock third-party requests for each test.

Install our two core dependencies:

```
yarn add apollo-server-testing jest
```

We will need two files:

1. The test cases, as GraphQL query strings for each request.
2. Our test runner to process and record snapshots for each test case.

Test cases

At scale, we would likely create a different test file for each group of resolvers, but for now, let's just assume we have one file called `resolvers.test.js`.

Here, we can add our test cases, which are an array of objects containing the GraphQL query string and any variables associated with the query:

```
import { runTestCases } from './test-runner';

const testCases = [
  {
    id: 'getBooking',
    query: `
      query ($id: Int!) {
        getBooking (id: $id) {
          id
          cartReference
          paymentAmount
          receiptNumber
          cancelled
        }
      }
    `,
    variables: {
      id: 1,
    },
  },
  {
    id: 'cancelBooking',
    query: `
      mutation ($id: Int!) {
        cancelBooking (id: $id) {
          id
          cancelled
        }
      }
    `,
    variables: {
      id: 1,
    },
  },
];

runTestCases('Bookings', testCases);
```

snapshot-testing-test-cases-1.js hosted with ❤ by GitHub view raw

At the bottom of the file, you can see we pass the `testCases` into a function called `runTestCases`. We also provide a name for this set of test cases that will be applied to the outputted snapshots.

So let's go ahead and write the test runner itself.

Creating the test runner

As you know, snapshot testing is a really quick process. So in its simplest form, we will create a test client and pass each query into it. We then wait for the response and record it.

```
import { createTestClient } from 'apollo-server-testing';
import { createServer } from './createServer';

export const runTestCases = (
  groupName,
  testCases,
) => {
  const { query } = createTestClient(createServer());

  describe(`${groupName} resolvers`, () => {
    for (let testCase of testCases) {
      it(testCase.id, async () => {
        const res = await query({
          query: testCase.query,
          variables: testCase.variables || {},
        });
        expect(res).toMatchSnapshot();
      });
    };
  });
};
```

snapshot-testing-runner.js hosted with ❤ by GitHub view raw

Now when running the `jest` command, Jest will record the response under `__snapshots__/resolvers.test.js.snap`:

```
exports[`Bookings Integration Tests getBooking`] = `
Object {
  "data": Object {
    "getBooking": Object {
      "id": 1,
      "cartReference": "123-123-456",
      "paymentAmount": 2000,
      "cancelled": false,
```

```
      "receiptNumber": "456464565",
    },
  },
  "errors": undefined,
  "extensions": undefined,
  "http": Object {
    "headers": Headers {
      Symbol(map): Object {},
    },
  },
}
`;
```

snapshot-testing-output.js hosted with ♥ by GitHub view raw

From this point forward, you can quickly add and remove tests for each resolver. Depending on your use case, you could even seed/drop a database for each test, providing isolated end-to-end coverage.

. . .

Managing Snapshot Tests Over Time

As your project grows over time and large refactors take place, you can continue to run your tests and see that the snapshots provide the same output. If you need to update a snapshot, run `jest -u`, which will replace the content.

When updating snapshots, it is extremely important to validate that the new output is what you expect. I have seen many developers run `jest -u` and commit the code without a second thought. This can easily be caught through PR reviews. But when working alone, it's imperative to be strict with this process.

I use VSCode and have found the diff tool under the source control tab a fantastic way to monitor unintended changes to snapshots when completing refactors.

. . .

Extra Info

What do I pass into `createTestClient`?

In the example above, I passed a method into `createTestClient` called `createServer`. This is just an ApolloServer instance containing your resolvers:

```
import { ApolloServer } from 'apollo-server';
import { schema } from './schema';

export const createServer = () => {
  return new ApolloServer({
    schema,
  });
};
```

snapshot-testing-server.js hosted with ♥ by GitHub view raw

How do I handle mutations?

`createTestClient` returns an object containing `{ query, mutate }`. You must pass the corresponding string into the correct method. In the `testCases` objects, you could add mutations under a key called `mutate` or write a small function to detect this from the string itself:

```
export const getIsQueryOrMutation = (queryString: string) => {
  return queryString.replace(/\s/g, '').startsWith('query')
    ? 'query'
    : 'mutate';
};
```

snapshot-testing-query-or-mutation.js hosted with ♥ by GitHub view raw

Let me know if you have any further questions.

Programming Testing Software Development GraphQL JavaScript

134

Written by Warren Day

Follow

22 Followers · Writer for Better Programming

Software Engineer | Creator of https://tomorrowapp.io | Follow me on Twitter https://twitter.com/warrenjday

More from Warren Day and Better Programming

Warren Day in tomorrowapp

The complex world of calendars: Database design

In part 1 we discuss the use of RRule to describe the recurrences of a repeating...

2 min read · May 30, 2020

70

Timothy Mugayi in Better Programming

How To Build Your Own Custom ChatGPT With Custom Knowledg...

Feed your ChatGPT bot with custom data sources

11 min read · Apr 7

3.9K 91

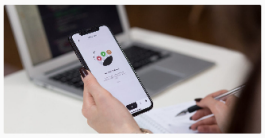
Vinita in Better Programming

How to Build Credibility at Work

Building credibility requires more than just competence and knowledge.

8 min read · Apr 3

3.5K 81



Warren Day in tomorrowapp

Running end-to-end tests with GitHub Actions

I always liked the idea of Github actions because it means less third-party...

3 min read · Jul 24, 2020

73 1

See all from Warren Day | See all from Better Programming

Recommended from Medium

Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...

They don't meet the bar.

4 min read · Apr 13

3.2K 103

Somnath Singh in JavaScript in Plain English

Coding Won't Exist In 5 Years. This Is Why

Those who won't adapt would cease to exist.

8 min read · Jan 20

7.7K 262

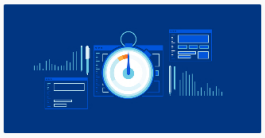
The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of...

Master ChatGPT by learning prompt engineering.

7 min read · Mar 17

17.2K 308



Ibrahim Ahmed in Bootcamp

How I Optimized An API Endpoint To Make It 10x Faster

When it comes to building web applications, performance is one of the most important...

3 min read · Jan 11

270 7

aruva - empowering ideas

Using ChatGPT to build System Diagrams—Part I

The popularity of ChatGPT keeps growing and it's now being used in a wide range of...

✦ · 7 min read · Jan 17

4.5K 50

Jacob Bennett in Level Up Coding

Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

✦ · 4 min read · Nov 15, 2022

4.5K 51

See more recommendations