Development Testing

Snapshot testing

16 DECEMBER 2019 • 8 MIN READ

Krzysztof Ciombor

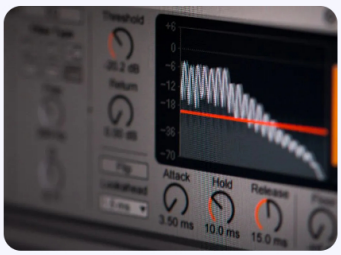


TABLE OF CONTENTS

Deep dive into Snapshot testing

This is a follow-up to the previous **article**. Read it first, if you haven't already. Today, we're going to take a closer look at snapshot testing to show you how we improved it and took it to the next level.

What is snapshot testing?

The original idea of Snapshot Testing was first included in Jest - have a look **here** to learn more. Its main purpose was testing frontend component changes over time to catch any unexpected UI changes. This concept, however, is more universal and can be applied to almost all codebases. The overall process is always the same:

1. Execute your code (either by rendering the UI, calling a method, or issuing an HTTP request).
2. Capture the result.
    1. If you were running the test for the first time, save the result to a "snapshot" file.
    2. Otherwise, find an existing "snapshot" file and compare it with the result. If the contents are the same, pass the test. If the contents don't match, report the test failure.

Why use snapshot testing?

By using snapshot testing, we get a few benefits:

- The tests are easy to write and maintain.
- Since they are essentially similar to integration/acceptance tests, we'll be able to validate the flow through the entire application stack (i.e. HTTP requests, Service layer, DB integration, etc.).)
- They are a "simpler" alternative to Contract Based Testing (see **this source**) that still give us confidence that we're not introducing any breaking changes to our APIs.

Snapshot testing v1

Our initial implementation of Snapshot Tests was based on the **Kotlin Snapshot** library. It worked great at the beginning, but a couple of issues started to arise once we added more and more tests. One of the most frequent ones were non-deterministic JSON outputs, for example, when generating random UUIDs. We were frequently facing failures like this:

```
-Snapshot
+Received

 [{"address":{"street":"street 1","id":"
- b12cd6f9-fa06-4665-ad45-0f76f7acdbd3
+ 0cac663f-2ede-41fd-8265-6a206c7adae8
 "},"geocoordinates":{"latitude":50.0646501,"longitude":19.944544,"id":"
- d9da0939-59a4-49c0-adc2-704277e0efe3
+ b55efbc0-fabc-4f8a-ac99-4ada9632909a
 "},"comment":"","chargePointState":"ACTIVE","id":"
- edf811aa-c57b-4ef8-ad97-4c5be7771880
+ 81926f4a-6502-4922-8e11-ba68745c68c2
 "}]
```

Even though it makes sense from the String equality perspective, it doesn't really matter from business requirements side. As long as the output JSON contains the id field, we don't really care about its content. This can be mitigated by creating deterministic helpers for UUID creation.

The next issue started happening when we compared arrays. Our output was very often serialized with arrays contents in random order.

The second biggest issue we faced was the order of arrays. Even though the serialization itself should preserve the elements order, we can't guarantee that the collections were always sorted the same (for example, a simple .addAll() on a Collection doesn't have any ordering guarantees). Again, this became a problem when comparing snapshots as strings, even though from a logical standpoint, as long as the array elements are the same, the ordering doesn't really matter.

So the question we started asking ourselves: "Can we do better than this?" And the answer is... "Yes we can!"

Custom "Snapshotter"

For our purpose here, we're using snapshots for validating API responses. Since they always come in the form of JSON, we can take advantage of that. As it turns out, there's a library out there that is perfect for our **use-case**. However, this also means that we had to ditch the KotlinSnapshot library in favor of a custom implementation. Fortunately, that's not so complicated to achieve and in our PoC we were able to (almost) replicate all the functionalities of KotlinSnapshot, re-using a couple of its code snippets. One of our goals was to keep the compatibility between both to facilitate the migration of the existing tests. In the end, the only required for existing test code is as simple as:

```
- result.response.contentAsString.matchWithSnapshot()
+ snapshotter.validateSnapshot(result.response.contentAsString)
```

Benefits of JSON snapshots

So what exactly do we get by switching from String comparison to JSON comparison? Here are two major benefits:

Array order independence

Taking advantage of JSONassert's JSONCompareMode.NON_EXTENSIBLE comparison mode, we can compare an array's contents disregarding their order, but still guarantee that no element is missing and no element was added. This means that we no longer have to focus our efforts for making the array order deterministic and can instead concentrate on the array's contents.

Custom validation rules

Maybe a more interesting part comes with custom validation config which can be passed to validateSnapshot method. Using JSONassert's full potential, we can ignore properties that we don't want to validate during the snapshot comparison. Going back to our previous example, we can now easily exclude id fields from being checked for equality by doing:

```
snapshotter.validateSnapshot(
    data = result.response.contentAsString,
    validationConfig = ValidationConfig(
        ignore = listOf("id")
    )
)
```

Moreover, we can use a wildcard operator for more sophisticated exclusions, namely:

- **"id"** - Will match top-level id field
- **"foo.id"** - Will match nested foo.id field
- **"**.id"** - Will match all id fields, no matter the nesting level
- **"****** - Will match ALL fields (useful for doing selected fields comparison only)

Keep in mind that ignoring the field for validation only skips the comparison part. The field still needs to be present in the response, so we don't need to worry that we'll introduce API-breaking changes by removing some response fields.

```
java.lang.AssertionError: [chargePointState=ACTIVE].address.street
Expected: Baker Street 221B
     got: Sesame Street
```

Not a "silver bullet"

One should always keep in mind that there's no such thing as "silver bullet" in

One should always keep in mind that there's no such thing as "silver bullet" in software development. Each solution comes with its tradeoffs. That's the case with snapshot testing as well. There are still a couple of pain points with using it as the main testing method, namely:

- These tests are much "heavier" than unit tests, meaning that the "feedback loop" is considerably longer.
- Sometimes it will be difficult to reason about failures and pinpoint the exact erroneous place in the code.
- Snapshot files require maintenance, they must be updated every time a change in the API is correct and required.
- Assertions are not obvious and it may not be immediately visible what is being tested just by looking at the test code.
- Pull Request diffs "blow up". A single change to add one more parameters to the response DTO may cause dozens of snapshot to be updated. Since snapshot files should always be committed to the repo, those changes will show up in the code review.

Summary

As usual, it's best to always evaluate potential solutions and new approaches with care and adjust them to our needs in the project. What worked well for us may not work so well for others. We are quite happy with our current test stack, and we're able to achieve ~80% code coverage easily with just snapshot tests and a couple of unit tests mixed here and there for the most crucial business logic. This allows us to be confident when making changes, adding new features, and refactoring the old code.

Krzysztof Ciombor

Head of Products

Krzysztof is our Head of Products and he leads the technical development behind the suite of our ready-made solutions. Throughout the years in Solidstudio, he gained an invaluable insight into the eMobilty industry's requirements and is now more than qualified to share his knowledge with others.

Other articles

API Acceptance testing

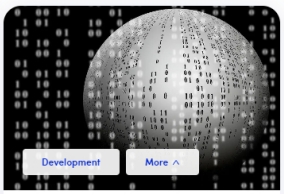
API Acceptance testing with Kotlin, DSL, testcontainers, and snapshots.

Krzysztof Ciombor
12 SEPTEMBER 2019 • 15 MIN READ

How to open API following protocols?

Onboard the API standards into your system.

Piotr Majcher
13 APRIL 2020 • 9 MIN READ



Introduction to serverless testing

Serveless testing - tools and techniques.

Paweł Głogowski
04 APRIL 2019 • 10 MIN READ

Proud member of

- E-mobility solutions
- Custom software development
- Dedicated dev teams
- Web Development
- Mobile development
- IT consulting

> **Show more**

- CPO Platform
- White-label EMSP Application
- OCPP Gateway
- OCPI Gateway

- Zipcharge
- Zeemcoins
- Zimi
- True Energy
- Ave Mobility
- E-mobility team

- eMobility News by Solidstudio
- Video Guide: The role of a Charge Point Operator and the responsibilities
- eMobility News by Solidstudio
- eMobility Fundamentals - the eBook
- eMobility News by Solidstudio
- Alternative Fuel Infrastructure Regulation (AFIR) - what's in it for eMobilty?

contact@solidstudio.io
+48 538 365 618

ul. Czysta 10/3
31-121 Cracow
Poland

Privacy Policy Cookies Policy