Approval Tests WorkFlow

The Approval Tests library works by comparing a “known good” piece of data to a new piece of data and highlighting the differences if any. For example, a request could be made to an existing API endpoint (v1) and the response saved to disk, Approval Tests can then be used to make a request to a newer version of the endpoint (v2). Approval tests will then automatically compare the two responses and produce an, industry standard format, diff file to show any differences. If there are no differences then the test will automatically pass.

It should be noted that the first time a call is made to a data source, Approval Tests will store that response for comparison to the next call as a convenience.

A common use case if that there are acceptable differences in the two responses. These might include, for example, date/time stamps, randomly generated IDs etc. The Approval Tests library allows for a straightforward mechanism to capture these differences and still allow for the test case to pass.

Step 1

Make the call to v1 of the API

```
String resp =
when().get("{{_serverUrl}}/v1/persons/{{_partnerId}}?includeContacts=true&includeLcr=false”).th
en().extract().body().asPrettyString();
```

Step 2

Ask the Approval Tests library to verify the response

```
Approvals.verify(resp);
```

As this is the first time we have requested this data, ApprovalTests has no known baseline to compare to so it stores this response for future comparisons, it does this by creating a file on disk with a file ending of “*.received.txt”

Step 3

After this response has been analyzed and approved, it can become the baseline for future comparisons (often referred to as the “Golden Master”). This is achieved by renaming the file to “*approved.txt”. This now becomes the baseline for future comparisons.

Step 4

Make the call to v2 of the API

String resp = when().get("{{_serverUrl}}/v2/persons/{{_partnerId}}?includeContacts=true&includeLcr=false”).then().extract().body().asPrettyString();

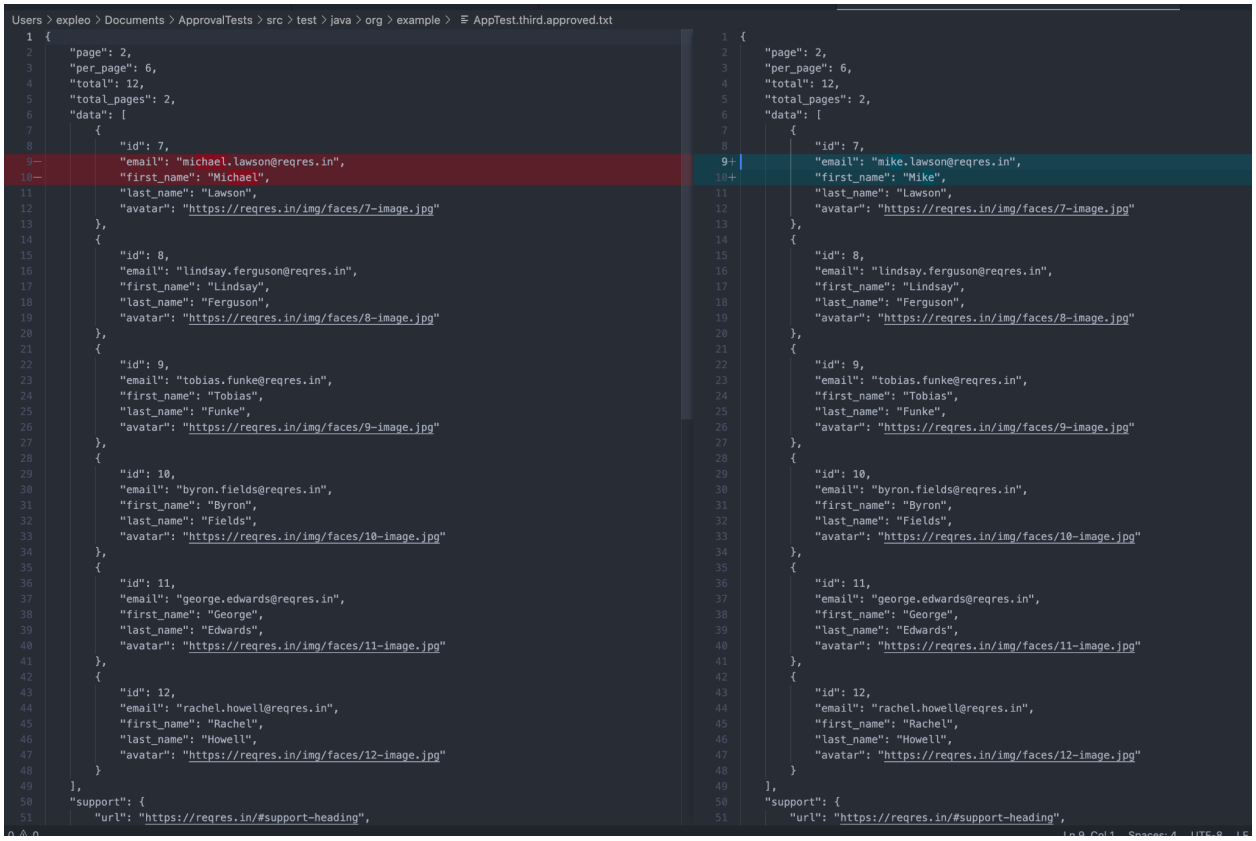
Step 5

Ask the Approval Tests library to verify the response

Approvals.verify(resp);

As this is NOT the first time we have requested this data, ApprovalTests has a known baseline to compare to (it searches for an “approved.txt” file) and generates a diff file if there are any differences.

As you can see in the screenshot below, there is a difference detected. This can then be triaged as being an acceptable (expected) difference (An exception can be allowed for in the test script) or an unexpected difference in which case a defect can be logged to the development team



Step 6

If an expected difference is found, the tester can script an exception into the test script to allow for that difference so that it will not affect the pass rate. These take the form of regex substitutions as below:

```
final Scrubber portScrubber = new RegExScrubber(":\\d+/", ":[port]/");
final Scrubber dateScrubber = DateScrubber.getScrubberFor("20210505T091112Z");
final Scrubber signatureScrubber = new RegExScrubber("Signature=.+",
"Signature=[signature]");
        Scrubber scrubber = Scrubbers.scrubAll(portScrubber, dateScrubber, signatureScrubber);

Approvals.verify("http://127.0.0.1:55079/foo/bar?Date=20210505T091112Z&Signature=4a7dd6f
09c1e",
                new Options(scrubber));
```