

End-to-End (E2E) Testing

Author(s): Paulo Jer™nimo

[HTML version](#)

Last updated: 2025-05-15 11:31:38 -0300

Last commit: 93fe9da

1. Introduction	1
2. Prerequisites	2
2.1. Change your current directory	2
2.2. Install Playwright and configure it to run in a devcontainer	2
3. Steps	3
3.1. Start all the services	3
3.2. Testing the ContinuousViewProcessor	4
3.2.1. Saving and analysing the generated data	4
3.3. Restarting the topics	6
3.4. Testing the MostViewedProcessor	7
3.4.1. Saving and analysing the generated data	7
4. Expected final results	9

1. Introduction

The following [Steps](#) allow you to test this project by automating user behaviour and executing scripts that access the front end through the [Playwright](#) framework.

If you're unfamiliar with Playwright, the only thing you need to know is that it's like a robot that accesses a website and simulates the behaviour of a real human. However, as a robot, its steps are completely predetermined through automated scripts.

Technically speaking, these Playwright tests are written in this project using [TypeScript](#). Additionally, there are some [Bash](#) and [jq](#) scripts that assist this process.

So, this is an end-to-end test that almost completely validates the architecture described in the [main document](#). See the [Expected final results](#) for this test.

2. Prerequisites

2.1. Change your current directory

1. To execute the following commands, from the project's root, you must ensure that your current directory is `scripts/e2e`.

2.2. Install Playwright and configure it to run in a devcontainer

1. We will use more than one terminal to execute the following commands. I will use [Tmux](#). Feel free to open terminals however you like.
2. Before starting, playwright needs to be installed using these commands:

```
$ npx playwright install-deps  
$ npx playwright install
```

3. Since we will be running playwright in a non-interactive environment, run this command:

```
$ export DEBIAN_FRONTEND=noninteractive
```

3. Steps

3.1. Start all the services

1. Open another terminal window.
2. Start all docker services (if they are not started yet):

```
$ ../docker/up.sh
```

3. Wait until all services are available
4. You can type `Ctrl+C` on the windows showing the logs for all services. Then, watch only the `discounts-processor` logs through this command:

```
$ ../docker/logs.sh discounts-processor
```

Alternatively, running the following commands has the same effect as above:

!

```
$ cd ../../docker
```

```
$ # run the following if you are starting the devcontainer for the first time:
```

```
$ ./setup.sh
```

```
$ # tip: use --build below only if you changed some code
```

```
$ docker compose up --build -d
```

```
$ docker compose logs -f discounts-processor
```

3.2. Testing the ContinuousViewProcessor

1. Open another terminal to start testing the "ContinuousView" processor:

```
$ ./test.sh
Usage: ./test.sh <test>
Available tests: ContinuousView MostViewed

$ rm -rf data

$ ./test.sh ContinuousView
```

2. Wait for the test script to finish. This should take up to 5 minutes since this is an end-to-end test that simulates real user behaviour.

3.2.1. Saving and analysing the generated data

1. See the generated log for the last command:

```
$ tree data

$ cat data/ContinuousView.log
```

2. Save the data generated in the topics:

```
$ ./save-data.sh ContinuousView

$ cat data/ContinuousView/snowplow-enriched-good.jsonl

$ !! | jq . # <- to see the results pretty formatted.
```

3. The discount event, if it has been successfully generated, can be viewed with this command:

```
$ cat data/ContinuousView/shopper-discounts.jsonl | jq .
```

4. Use the `summarize.sh` script. It will provide details about the events generated.
 - a. It executes a small [Kotlin](#) app to get details about these events. This app uses the same [Java](#) code created to generate the discount event.

```
$ ./summarize.sh ContinuousView

$ cat data/ContinuousView/snowplow-enriched-good.summary-by-kt.json | jq .
```

- b. Alternatively, you can summarize by using a different implementation (written in

JavaScript). This is useful to double check if Java implementation is correct.

```
$ version=js ./summarize.sh ContinuousView  
  
$ cat data/ContinuousView/snowplow-enriched-good.summary-by-js.json | jq .
```

- c. It should be obvious that summarisation, regardless of the algorithm, should produce the same result for the same set of events. So, test this with the following command:

```
$ ./diff-between-summaries.sh ContinuousView
```

- i. You should not see any results in the output of the above command. However, there may be a minor difference in the formatting of some timestamps presented.

3.3. Restarting the topics

1. To execute the next test and get only the date related to it, you need to restart the topics:

```
$ ./restart-topics.sh
```

2. At this point, if you had a terminal open to display the discounts-processor logs, you will need to reopen it.

3.4. Testing the MostViewedProcessor

1. Start testing the `MostViewed` processor:

```
$ ./test.sh MostViewed
```

2. Wait about 5 minutes (since this is an end-to-end testing by simulating the real user behaviour).
3. Up to this point, this should be the data already generated:

```
$ tree data/
data/
|-- ContinuousView
|   |-- shopper-discounts.jsonl
|   |-- shopper-discounts.rpk.jsonl
|   |-- snowplow-enriched-good.jsonl
|   |-- snowplow-enriched-good.rpk.jsonl
|   |-- snowplow-enriched-good.summary-by-js.json
|   `-- snowplow-enriched-good.summary-by-kt.json
|-- ContinuousView.log
`-- MostViewed.log

2 directories, 8 files
```

3.4.1. Saving and analysing the generated data

1. Save the data generated in the topics and check it:

```
$ ./save-data.sh MostViewed

$ cat data/MostViewed/snowplow-enriched-good.jsonl

$ !! | jq .
```

2. The discount event, if it has been successfully generated, can be viewed with this command:

```
$ cat data/MostViewed/shopper-discounts.jsonl | jq .
```

3. Use the `summarize.sh` script:

```
$ ./summarize.sh MostViewed
$ cat data/MostViewed/snowplow-enriched-good.summary-by-kt.json | jq .

$ version=js ./summarize.sh MostViewed
$ cat data/MostViewed/snowplow-enriched-good.summary-by-js.json | jq .
```

4. As done previously, check if there is any differences between the generated summaries:

```
$ ./diff-between-summaries.sh MostViewed
```


4. Expected final results

As final results of this end-to-end test it is expected that:

1. The following directory and file structure has been generated on your machine:

```
$ tree data/
data/
|-- ContinuousView
|   |-- shopper-discounts.jsonl
|   |-- shopper-discounts.rpk.jsonl
|   |-- snowplow-enriched-good.jsonl
|   |-- snowplow-enriched-good.rpk.jsonl
|   |-- snowplow-enriched-good.summary-by-js.json
|   `-- snowplow-enriched-good.summary-by-kt.json
|-- ContinuousView.log
|-- MostViewed
|   |-- shopper-discounts.jsonl
|   |-- shopper-discounts.rpk.jsonl
|   |-- snowplow-enriched-good.jsonl
|   |-- snowplow-enriched-good.rpk.jsonl
|   |-- snowplow-enriched-good.summary-by-js.json
|   `-- snowplow-enriched-good.summary-by-kt.json
`-- MostViewed.log

3 directories, 14 files
```

2. Discount events must have been generated and must be available in the `shopper-discounts.jsonl` files.
3. The content of the `snowplow-enriched-good.summary-by-*.json` files should be practically the same (minor differences may occur).