Aurora Tech Assignment Report

Alexandru Sirbu

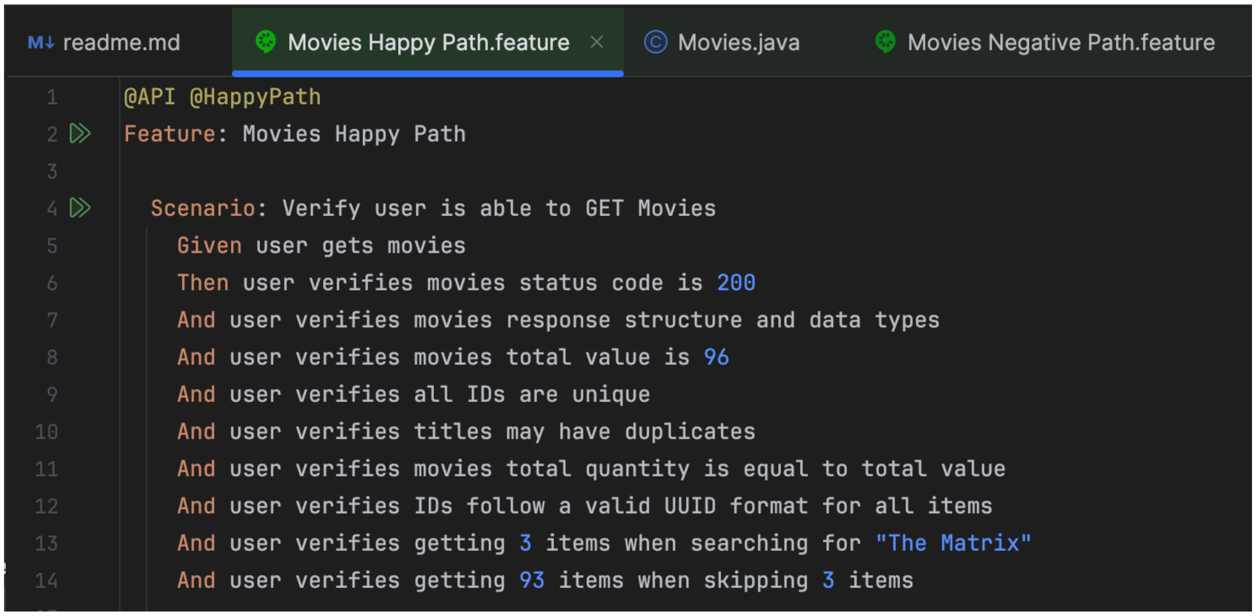
Disclaimer

Please read **readme.md** file for basic introduction.

Contact me regarding any questions - **alexandru.sirbu.work@outlook.com**

How does Automation Testing Framework work?

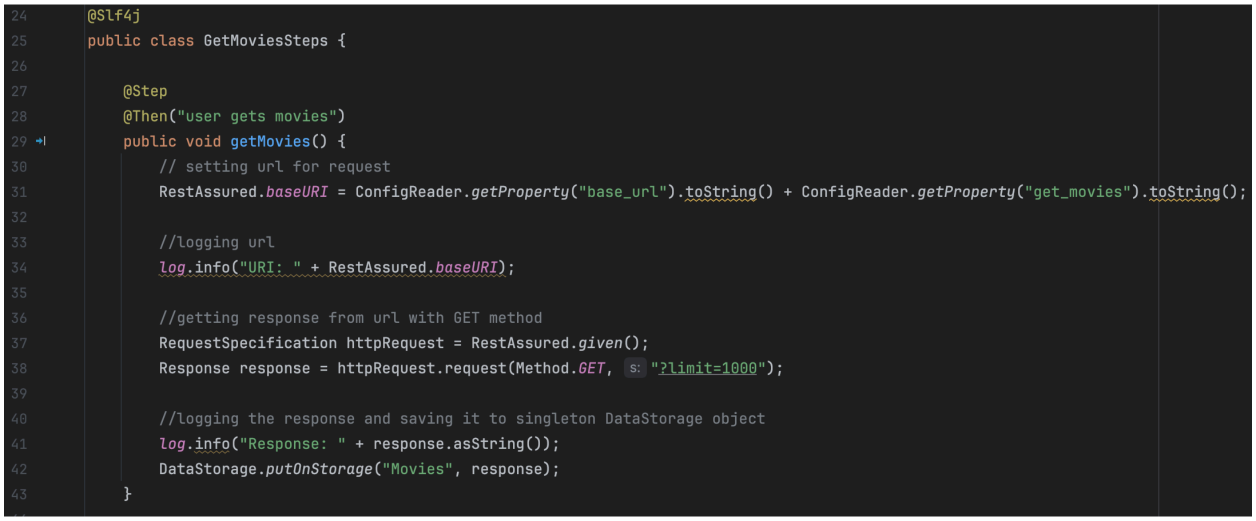
If you already clicked test cases links in readme.md file then you should've seen so called .feature files, which contain test scenarios with steps written in Gherkin language. I decided to connect different test cases into 3 scenarios (1 Positive and 2 Negative) that execute all the test cases automatically using such tools as Java, RestAssured, Cucumber, Junit, Maven, Allure, Logback, SLF4J, AspectJ, AssertJ and Lombok.



```
@API @HappyPath
Feature: Movies Happy Path

  Scenario: Verify user is able to GET Movies
    Given user gets movies
    Then user verifies movies status code is 200
    And user verifies movies response structure and data types
    And user verifies movies total value is 96
    And user verifies all IDs are unique
    And user verifies titles may have duplicates
    And user verifies movies total quantity is equal to total value
    And user verifies IDs follow a valid UUID format for all items
    And user verifies getting 3 items when searching for "The Matrix"
    And user verifies getting 93 items when skipping 3 items
```

Using Cucumber framework with Gherkin language is a convenient way of maintaining and developing test cases, following clean code practices and best QA practices. These steps are easy to understand by both stakeholders and other team members, but if you don't know how they work then it looks like it's a set of AI prompts that are executing tests and returning us results. Let me shortly explain you all the magic happening under the hood.

So, when we run a .feature file, each step executes specified piece of code that is defined in Step Definition files (GetMoviesSteps.java in our case):



```java
@Slf4j
public class GetMoviesSteps {

    @Step
    @Then("user gets movies")
    public void getMovies() {
        // setting url for request
        RestAssured.baseURI = ConfigReader.getProperty("base_url").toString() + ConfigReader.getProperty("get_movies").toString();

        //logging url
        log.info("URI: " + RestAssured.baseURI);

        //getting response from url with GET method
        RequestSpecification httpRequest = RestAssured.given();
        Response response = httpRequest.request(Method.GET, "?limit=1000");

        //logging the response and saving it to singleton DataStorage object
        log.info("Response: " + response.asString());
        DataStorage.putOnStorage("Movies", response);
    }
```

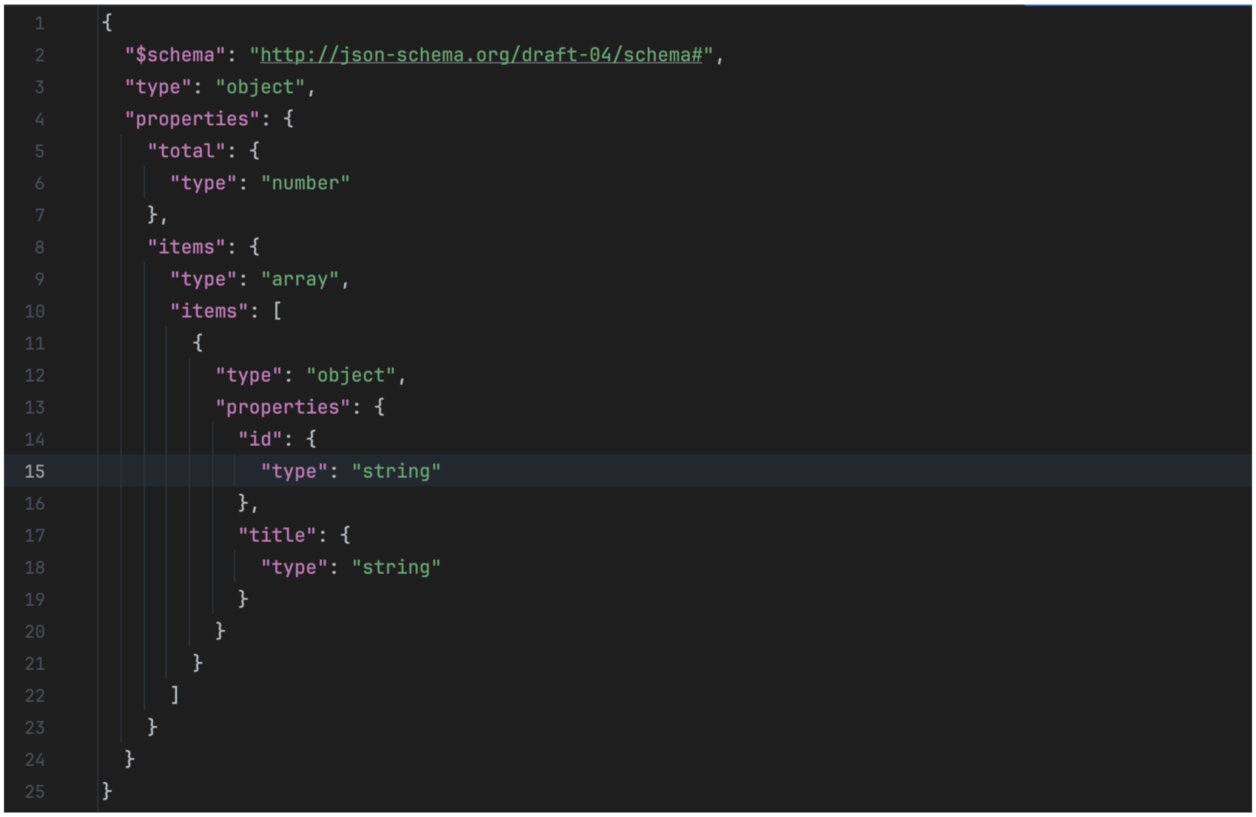
As example here: “user gets movies” step implementation, sends a GET request to https://november7-730026606190.europe-west1.run.app/movies?limit=1000 and saves the response to internal singleton DataStorage class, so it can be used by other steps or tests during this test execution. ConfigReader class retrieves URL values from config.properties to reduce repeatability and improve code maintainability.

In our case **GetMoviesSteps.java** contains all the steps I used for testing this endpoint and you can see all the code that stands behind .feature files to check how the tests are build and how validations are done.

Of course there is a principle that “**Exhaustive testing is impossible**” and I could’ve tried much more ways to break the parameters and media types of our endpoint, but I think this is redundant in such cases because most modern APIs have some level of security that prevents execution of things like ‘DROP DATABASE’ or trying to break the string. Hackers wouldn’t be able to hack the system using parameters and users are not using REST API to communicate with application.

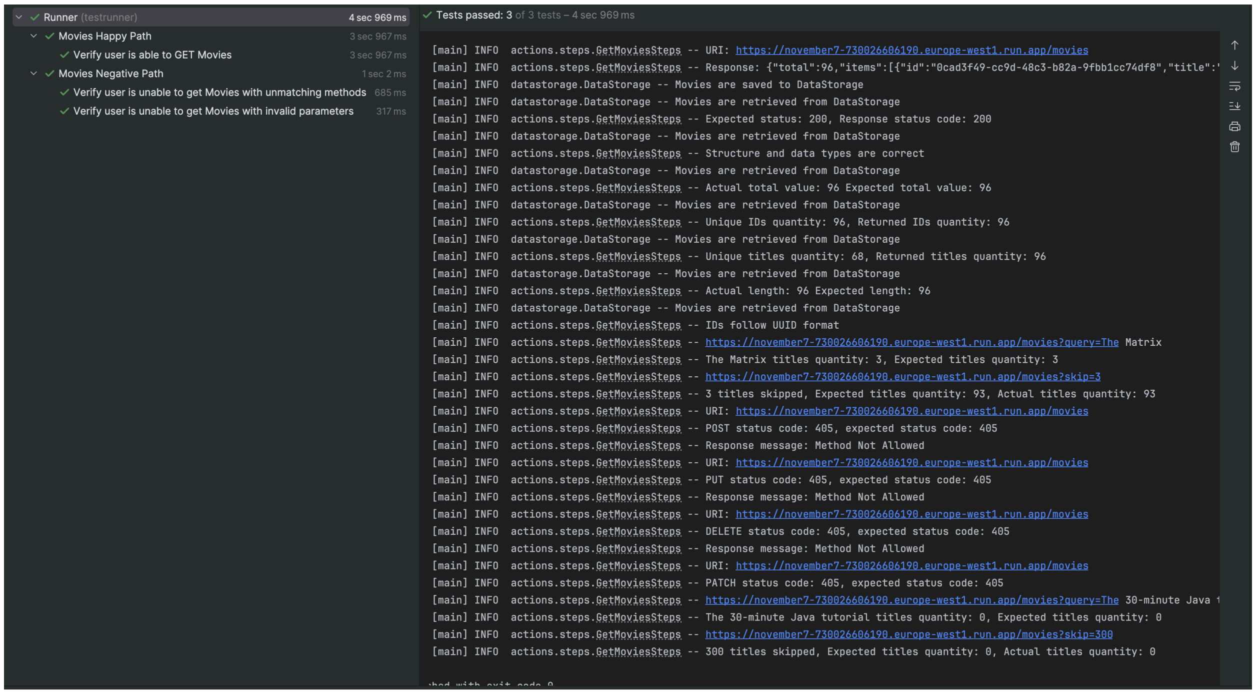
Other classes and files in this repository serve as auxiliaries and help us maintain and develop this test automation framework following clean code practices and best Test Automation Framework practices.

For example **movies-schema.json** is used to validate the response's format, structure and data types in Test Cases 3-5



```json
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "total": {
      "type": "number"
    },
    "items": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "id": {
              "type": "string"
            },
            "title": {
              "type": "string"
            }
          }
        }
      ]
    }
  }
}
```

Now let's take a look on the logs we get after test execution:



```
Runner (testrunner)                                                    4 sec 969 ms
  Movies Happy Path                                                    3 sec 967 ms
    Verify user is able to GET Movies                                  3 sec 967 ms
  Movies Negative Path                                                 1 sec 2 ms
    Verify user is unable to get Movies with unmatching methods        685 ms
    Verify user is unable to get Movies with invalid parameters        317 ms
```

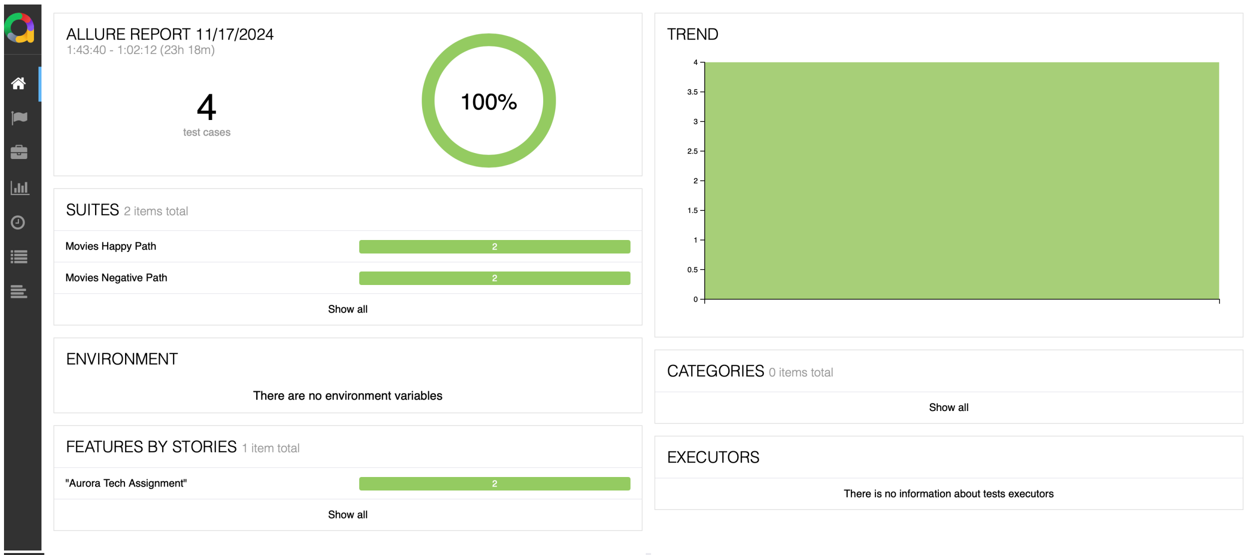
```
Tests passed: 3 of 3 tests – 4 sec 969 ms

[main] INFO  actions.steps.GetMoviesSteps -- URI: https://november7-730026606190.europe-west1.run.app/movies
[main] INFO  actions.steps.GetMoviesSteps -- Response: {"total":96,"items":[{"id":"0cad3f49-cc9d-48c3-b82a-9fbb1cc74df8","title":'
[main] INFO  datastorage.DataStorage -- Movies are saved to DataStorage
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- Expected status: 200, Response status code: 200
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- Structure and data types are correct
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- Actual total value: 96 Expected total value: 96
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- Unique IDs quantity: 96, Returned IDs quantity: 96
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- Unique titles quantity: 68, Returned titles quantity: 96
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- Actual length: 96 Expected length: 96
[main] INFO  datastorage.DataStorage -- Movies are retrieved from DataStorage
[main] INFO  actions.steps.GetMoviesSteps -- IDs follow UUID format
[main] INFO  actions.steps.GetMoviesSteps -- https://november7-730026606190.europe-west1.run.app/movies?query=The Matrix
[main] INFO  actions.steps.GetMoviesSteps -- The Matrix titles quantity: 3, Expected titles quantity: 3
[main] INFO  actions.steps.GetMoviesSteps -- https://november7-730026606190.europe-west1.run.app/movies?skip=3
[main] INFO  actions.steps.GetMoviesSteps -- 3 titles skipped, Expected titles quantity: 93, Actual titles quantity: 93
[main] INFO  actions.steps.GetMoviesSteps -- URI: https://november7-730026606190.europe-west1.run.app/movies
[main] INFO  actions.steps.GetMoviesSteps -- POST status code: 405, expected status code: 405
[main] INFO  actions.steps.GetMoviesSteps -- Response message: Method Not Allowed
[main] INFO  actions.steps.GetMoviesSteps -- URI: https://november7-730026606190.europe-west1.run.app/movies
[main] INFO  actions.steps.GetMoviesSteps -- PUT status code: 405, expected status code: 405
[main] INFO  actions.steps.GetMoviesSteps -- Response message: Method Not Allowed
[main] INFO  actions.steps.GetMoviesSteps -- URI: https://november7-730026606190.europe-west1.run.app/movies
[main] INFO  actions.steps.GetMoviesSteps -- DELETE status code: 405, expected status code: 405
[main] INFO  actions.steps.GetMoviesSteps -- Response message: Method Not Allowed
[main] INFO  actions.steps.GetMoviesSteps -- URI: https://november7-730026606190.europe-west1.run.app/movies
[main] INFO  actions.steps.GetMoviesSteps -- PATCH status code: 405, expected status code: 405
[main] INFO  actions.steps.GetMoviesSteps -- https://november7-730026606190.europe-west1.run.app/movies?query=The 30-minute Java t
[main] INFO  actions.steps.GetMoviesSteps -- The 30-minute Java tutorial titles quantity: 0, Expected titles quantity: 0
[main] INFO  actions.steps.GetMoviesSteps -- https://november7-730026606190.europe-west1.run.app/movies?skip=300
[main] INFO  actions.steps.GetMoviesSteps -- 300 titles skipped, Expected titles quantity: 0, Actual titles quantity: 0
```

Here we can see the output of each log.info() method with internal data and processes being displayed to framework’s end user and saved to logs.log file.

Sure it’s hard to quickly understand what’s going on here, so I added more convenient and metrics oriented reports, which can be generated after each test execution using following commands: `allure generate target/allure-results --clean -o Output/allure-report —single-file`

```
allure serve target/allure-results --clean -o Output/allure-report —single-file
```

These reports may be very useful for reporting test execution results with non-technical people. Save location: Output/allure-report/index.html



ALLURE REPORT 11/17/2024
1:43:40 - 1:02:12 (23h 18m)

4 test cases

100%

SUITES 2 items total

Movies Happy Path	2
Movies Negative Path	2

Show all

ENVIRONMENT

There are no environment variables

FEATURES BY STORIES 1 item total

"Aurora Tech Assignment"	2

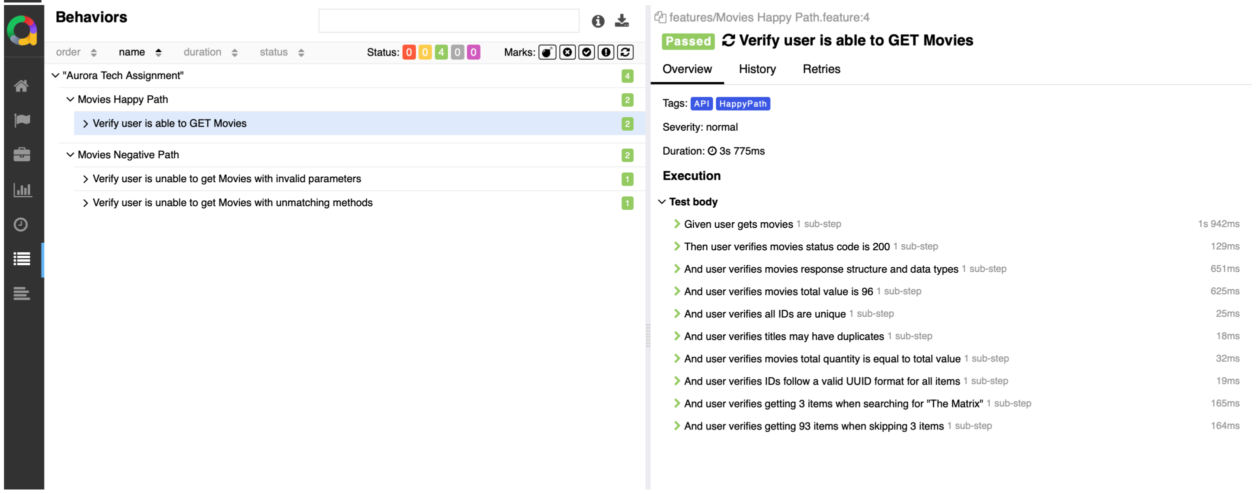
Show all

CATEGORIES 0 items total

Show all

EXECUTORS

There is no information about tests executors



Behaviors

order name duration status Status: 0 0 4 0 0 Marks:

"Aurora Tech Assignment" 4
Movies Happy Path 2
Verify user is able to GET Movies 2
Movies Negative Path 2
Verify user is unable to get Movies with invalid parameters 1
Verify user is unable to get Movies with unmatching methods 1

features/Movies Happy Path.feature:4

Passed Verify user is able to GET Movies

Overview History Retries

Tags: API HappyPath

Severity: normal

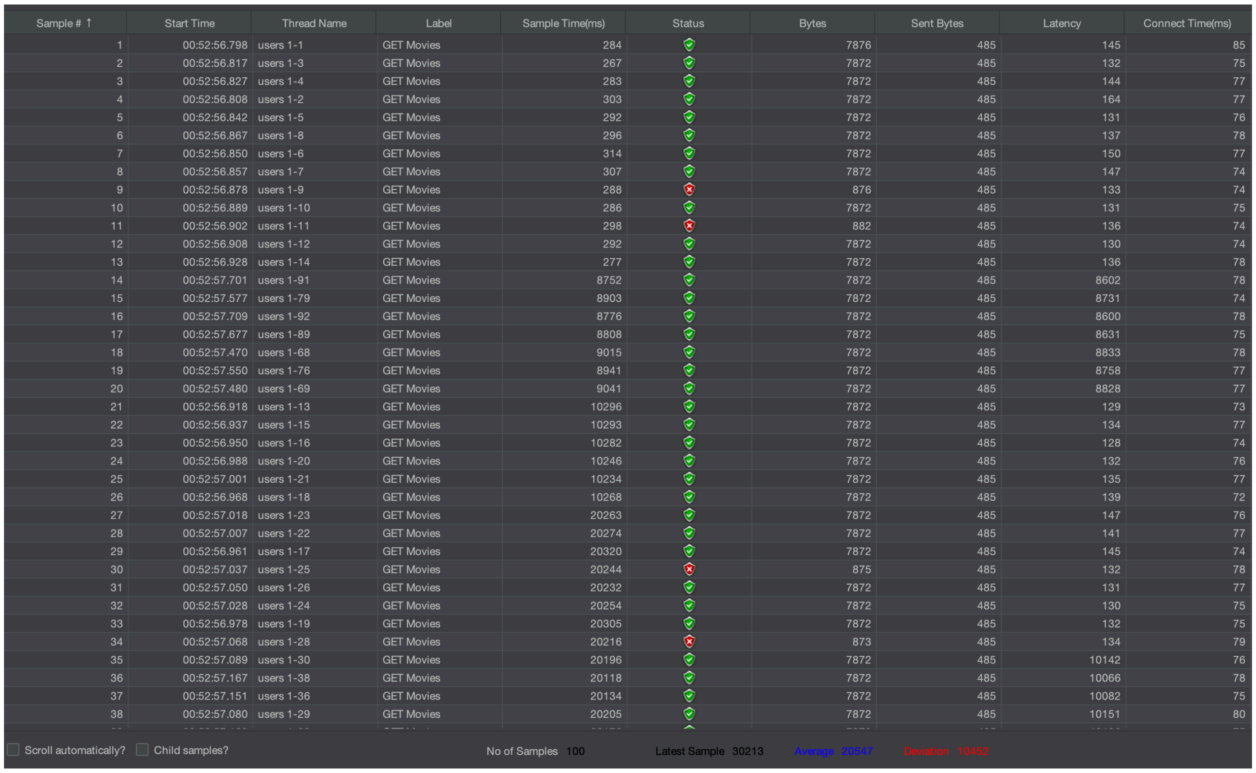
Duration: 3s 775ms

Execution

Test body

Given user gets movies 1 sub-step	1s 942ms
Then user verifies movies status code is 200 1 sub-step	129ms
And user verifies movies response structure and data types 1 sub-step	651ms
And user verifies movies total value is 96 1 sub-step	625ms
And user verifies all IDs are unique 1 sub-step	25ms
And user verifies titles may have duplicates 1 sub-step	18ms
And user verifies movies total quantity is equal to total value 1 sub-step	32ms
And user verifies IDs follow a valid UUID format for all items 1 sub-step	19ms
And user verifies getting 3 items when searching for "The Matrix" 1 sub-step	165ms
And user verifies getting 93 items when skipping 3 items 1 sub-step	164ms

Performance test of GET Movies endpoint

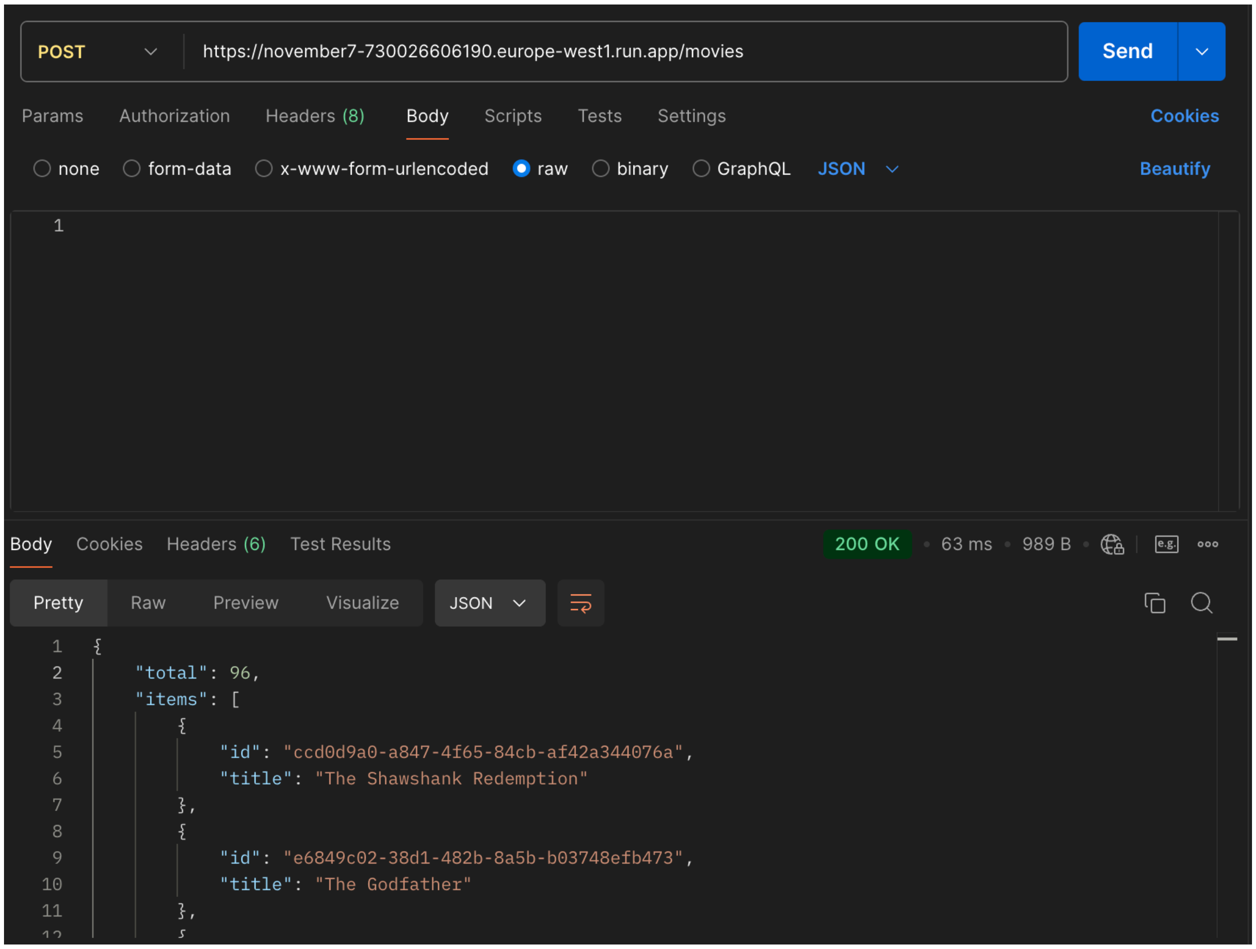


Sample # ↑	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	00:52:56.798	users 1-1	GET Movies	284	✓	7876	485	145	85
2	00:52:56.817	users 1-3	GET Movies	267	✓	7872	485	132	75
3	00:52:56.827	users 1-4	GET Movies	283	✓	7872	485	144	77
4	00:52:56.808	users 1-2	GET Movies	303	✓	7872	485	164	77
5	00:52:56.842	users 1-5	GET Movies	292	✓	7872	485	131	76
6	00:52:56.867	users 1-8	GET Movies	296	✓	7872	485	137	78
7	00:52:56.850	users 1-6	GET Movies	314	✓	7872	485	150	77
8	00:52:56.857	users 1-7	GET Movies	307	✓	7872	485	147	74
9	00:52:56.878	users 1-9	GET Movies	288	✗	876	485	133	74
10	00:52:56.889	users 1-10	GET Movies	286	✓	7872	485	131	75
11	00:52:56.902	users 1-11	GET Movies	298	✗	882	485	136	74
12	00:52:56.908	users 1-12	GET Movies	292	✓	7872	485	130	74
13	00:52:56.928	users 1-14	GET Movies	277	✓	7872	485	136	78
14	00:52:57.701	users 1-91	GET Movies	8752	✓	7872	485	8602	78
15	00:52:57.577	users 1-79	GET Movies	8903	✓	7872	485	8731	74
16	00:52:57.709	users 1-92	GET Movies	8776	✓	7872	485	8600	78
17	00:52:57.677	users 1-89	GET Movies	8808	✓	7872	485	8631	75
18	00:52:57.470	users 1-68	GET Movies	9015	✓	7872	485	8833	78
19	00:52:57.550	users 1-76	GET Movies	8941	✓	7872	485	8758	77
20	00:52:57.480	users 1-69	GET Movies	9041	✓	7872	485	8828	77
21	00:52:56.918	users 1-13	GET Movies	10296	✓	7872	485	129	73
22	00:52:56.937	users 1-15	GET Movies	10293	✓	7872	485	134	77
23	00:52:56.950	users 1-16	GET Movies	10282	✓	7872	485	128	74
24	00:52:56.988	users 1-20	GET Movies	10246	✓	7872	485	132	76
25	00:52:57.001	users 1-21	GET Movies	10234	✓	7872	485	135	77
26	00:52:56.968	users 1-18	GET Movies	10268	✓	7872	485	139	72
27	00:52:57.018	users 1-23	GET Movies	20263	✓	7872	485	147	76
28	00:52:57.007	users 1-22	GET Movies	20274	✓	7872	485	141	77
29	00:52:56.961	users 1-17	GET Movies	20320	✓	7872	485	145	74
30	00:52:57.037	users 1-25	GET Movies	20244	✗	875	485	132	78
31	00:52:57.050	users 1-26	GET Movies	20232	✓	7872	485	131	77
32	00:52:57.028	users 1-24	GET Movies	20254	✓	7872	485	130	75
33	00:52:56.978	users 1-19	GET Movies	20305	✓	7872	485	132	75
34	00:52:57.068	users 1-28	GET Movies	20216	✗	873	485	134	79
35	00:52:57.089	users 1-30	GET Movies	20196	✓	7872	485	10142	76
36	00:52:57.167	users 1-38	GET Movies	20118	✓	7872	485	10066	78
37	00:52:57.151	users 1-36	GET Movies	20134	✓	7872	485	10082	75
38	00:52:57.080	users 1-29	GET Movies	20205	✓	7872	485	10151	80

☐ Scroll automatically? ☐ Child samples? No of Samples 100 Latest Sample 30213 Average 20547 Deviation 10452

This report was generated by Jmeter. You can see the full report in 'GET Movies 100 requests results.csv' file. After sending 100 GET requests in 1 second we can see the results which show us that 9/100 requests fail which is actually critical for most modern applications. From my personal experience around 10% of all requests on this endpoint fail, independently of how many of them and in which timeframe we send them.

For example here I sent invalid request method through Postman and got 200 OK Status code and all the data I shouldn't receive:

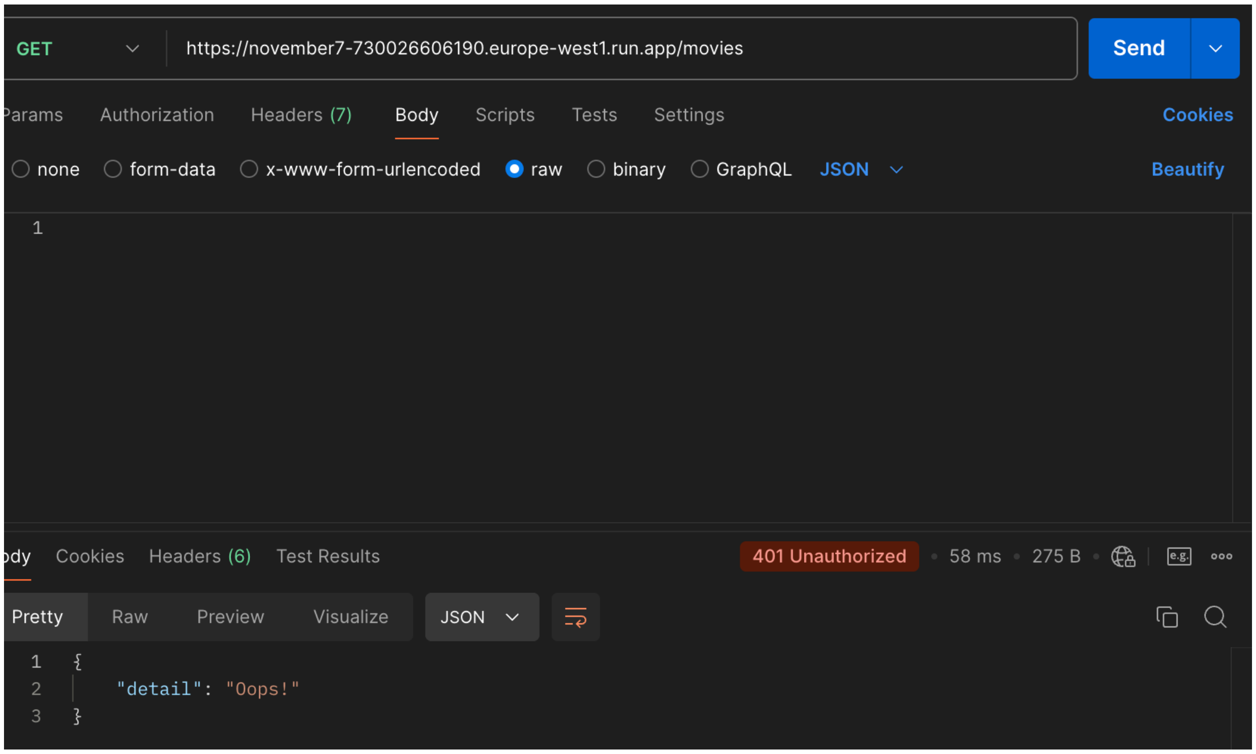

POST https://november7-730026606190.europe-west1.run.app/movies

200 OK • 63 ms • 989 B

```
{
    "total": 96,
    "items": [
        {
            "id": "ccd0d9a0-a847-4f65-84cb-af42a344076a",
            "title": "The Shawshank Redemption"
        },
        {
            "id": "e6849c02-38d1-482b-8a5b-b03748efb473",
            "title": "The Godfather"
        },
```

Such cases may be critical for application security since some users may not have the right to use GET method in the application but still be able to see the data.

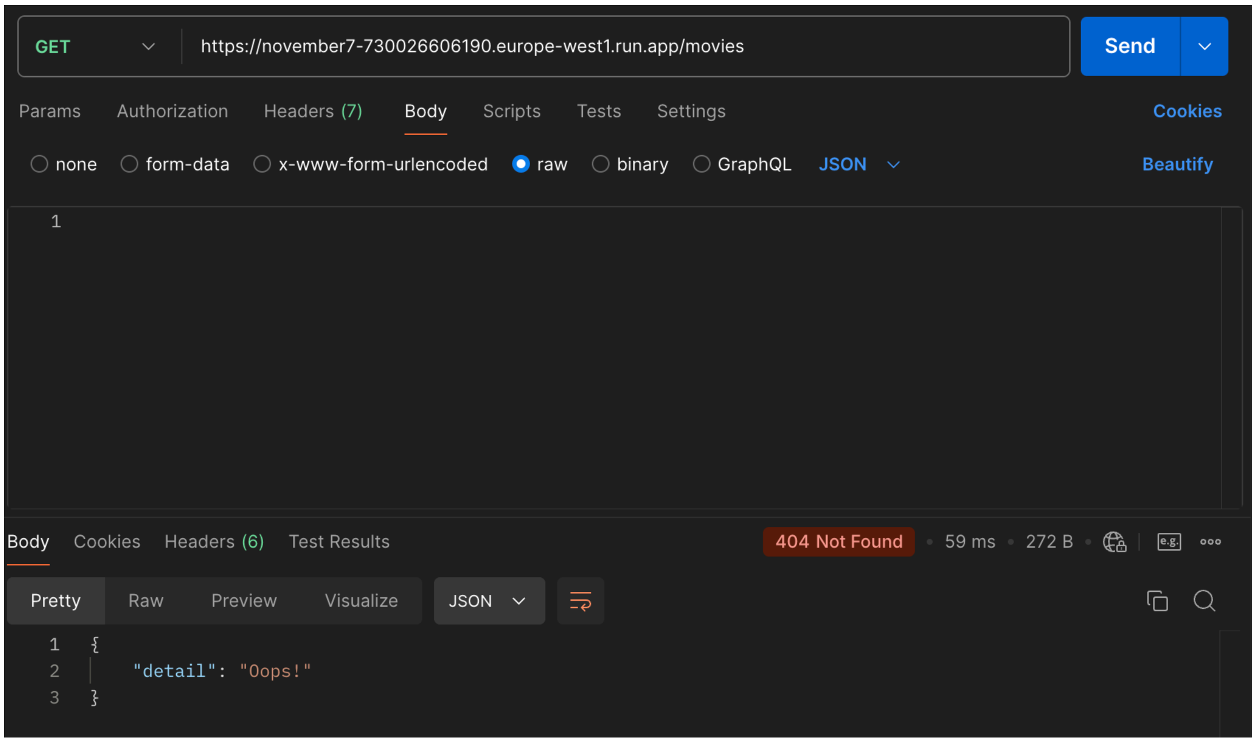
On the other hand there are cases with fully valid request but system decided not to respond correctly which gives us an understanding of system instability and ideas to improve:


GET https://november7-730026606190.europe-west1.run.app/movies

Body: raw, JSON

401 Unauthorized • 58 ms • 275 B

```
{
    "detail": "Oops!"
}
```


GET https://november7-730026606190.europe-west1.run.app/movies

Body: raw, JSON

404 Not Found • 59 ms • 272 B

```
{
    "detail": "Oops!"
}
```

Results

Let’s see what we have as result of this technical assignment:

- **Fully working, precise and scalable Test Automation Framework ready to be used for automating big corporate APIs, easy to start with, easy to maintain, ready for CI/CD, providing data-rich reports.**
- Response data is verified for accessibility, uniqueness, repeatability, structure, data types, parameter functionality, format, UUID format, total's value and the correctness of 'total' number.
- Test results of load testing shown us the obvious performance issues that would have been maximum priority in a real-life project.
- GET Movies endpoint can be tested in more details only in cases of access to Database or other REST methods.
- Detailed tech assignment report of the work done which can be used as a basis for further testing documentation.
- All the tests were executed manually and automated afterwards.