DP2 2023-2024

Testing report D04

Acme Software Factory

Repository: https://github.com/rafcasceb/Acme-SF-D04

Student #2:

- Flores de Francisco, Daniel danflode@alum.us.es

Other members:

Castillo Cebolla, Rafael	rafcasceb@alum.us.es
Heras Pérez, Raúl	rauherper@alum.us.es
Mellado Díaz, Luis	luimeldia@alum.us.es
Vento Conesa, Adriana	adrvencon@alum.us.es

GROUP C1.049

Version 1.2

25-05-24

Content Table

Contenido

Executive summary

This report will offer a detailed analysis of the testing procedure and results, including a section about functional testing and another one for performance testing.

We will follow a precise but accessible approach aiming to promote comprehension and assure a good final product.

Revision Table

Date	Version	Description of the changes	Sprint
11/05/2024	1.0	• Executive summary • Introduction • Functional testing	4
14/05/2024	1.0	• Performance testing • Conclusion • Bibliography	4
20/05/2024	1.1	• Remade after follow up	4
25/05/2024	1.2	• Revision after upgrading the framework	4

Introduction

This document will provide a detailed analysis of the testing procedure and results for the following features:

- Operations by clients on Contracts.
- Operations by clients on Progress Logs.

The content of a testing report is organized into two chapters:

- Functional testing: a listing with the test cases implemented, grouped by feature. For each test case, a succinct description plus a clear indication on how effective it was at detecting bugs are provided.

- Performance testing: it provides adequate charts, a 95%-confidence interval for the wall time taken by the application to serve the requests in the functional tests and a 95%- confidence hypothesis contrast.

Contents

Functional testing

Operations by client on Contracts

Test case 1: list-mine

For this command, we just selected the button for listing all contracts for two different clients

For hacking, we considered accessing the URL by a wrong role. Trying to access it with a good role but a wrong user doesn't make sense. Finally, trying to access with an anonymous user.

It provided a coverage of 93.1 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 2: show

For this command, we selected almost all contracts of the listing of Client1 to see their details.

For hacking, we tried accessing a contract of Client1 with a wrong role and with Client2. Finally, we tried accessing a contract with an anonymous user.

It provided a coverage of 96.6 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null contract (authorization), since the application from the web browser doesn't allow to. No bugs were detected.

Test case 3: create

For this command, we have tried to create a new contract. For each attribute we have checked the system rejects all different types of invalid data. Later, for each attribute, we have checked the system accepts all different types of valid data.

It provided a coverage of 95.7 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 4: update

For this command, we have updated a contract of the data base (id 359). For each attribute we have checked the system rejects all different types of invalid data. Later, for each attribute, we have checked the system accepts all different types of valid data.

For hacking, we tried to update a published contract, via URL.

It provided a coverage of 95.5 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null contract (authorization), since the application from the web browser doesn't allow to. For covering these instructions, we have to use external tools like Postman (post a null contract). No bugs were detected.

Test case 5: delete

For delete command we deleted some of Client1 contracts. No special cases in this feature. The unbind method was removed since it was useless in this case.

For hacking, we tried to delete a published contract, via URL.

It provided a coverage of 87.8 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null contract (authorization), since the application from the web browser doesn't allow to. For covering these instructions, we have to use external tools like Postman (delete a null contract). No bugs were detected.

Test case 6: publish

For this command, we have followed the same procedure as in the create, but instead create, we tried directly to publish the contract. For each attribute we have checked the system rejects all different types of invalid data. Later, for each attribute, we have checked the system accepts all different types of valid data. Finally we published a contract with valid data.

We also check that the amount of budget of contracts with the same associated project, does not exceed the total project cost. We tried:

- Trying to publish a contract that exceed the project cost
- Trying to publish a contract that does not exceed the project cost

For hacking, we tried to publish a published contract, via URL.

It provided a coverage of 96.1 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null contract (authorization), since the application from the web browser doesn't allow to. For covering these instructions, we have to use external tools like Postman (publish a null contract). A minor bug was detected, when the system compares the amount of budgets with the total project cost when any of them is null. It was fixed easily.

Operations by client on Progress logs

Test case 1: list-mine

For this command, we just selected the button for listing all progress logs for two different clients.

For hacking, we considered accessing the URL by a wrong role. Trying to access it with a good role but a wrong user doesn't make sense. Finally, trying to access with an anonymous user.

It provided a coverage of 92.6 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 2: list-for-contract

For this command, we clicked on the button to see the progress logs of almost all contracts: contracts with multiple children progress logs, contracts with one progress log, and contracts without any progress log.

For hacking, we tried accessing the same progress log listing with a wrong role, with Client2 and with an anonymous user.

It provided a coverage of 93.3%. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null progress log (authorization), since the application from the web browser doesn't allow to. No bugs were detected.

Test case 3: show

For this command, we selected almost all progress logs of the listing of Client1 to see their details.

For hacking, we tried accessing a progress log of Client1 with a wrong role and with Client2. Also with an anonymous user.

It provided a coverage of 95.2%. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null progress log (authorization), since the application from the web browser doesn't allow to. No bugs were detected.

Test case 4: create

For this command, we have tried to create a new progress log. For each attribute we have checked the system rejects all different types of invalid data. Later, for each attribute, we have checked the system accepts all different types of valid data.

It provided a coverage of 93.8 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 5: update

For this command, we have updated a progress log of the data base (id 597). For each attribute we have checked the system rejects all different types of invalid data. Later, for each attribute, we have checked the system accepts all different types of valid data.

For hacking, we tried to update a published progress log, via URL.

It provided a coverage of 94.0 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null progress log (authorization), since the application from the web browser doesn't allow to. For covering these instructions, we have to use external tools like Postman (post a null progress log). No bugs were detected.

Test case 6: delete

For this command we deleted almost all progress logs. There is no restriction to test. Note that there is no unbind method in this class, since it does not make sense to have it.

For hacking, we tried to delete a published progress log, via URL.

It provided a coverage of 91.7%. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null progress log (authorization), since the application from the web browser doesn't allow to. For covering these instructions, we have to use external tools like Postman (delete a null progress log). No bugs were detected.

Test case 7: publish

For this command, we have followed the same procedure as in the create, but instead of create it, we tried directly to publish the progress log. For each attribute we have checked the system rejects all different types of invalid data. Later, for each attribute, we have checked the system accepts all different types of valid data. Finally we published a progress log with valid data.

For hacking, we tried to publish a published progress log, via URL.

It provided a coverage of 94.1 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user and null progress log (authorization), since the application from the web browser doesn't allow to. For covering these instructions, we have to use external tools like Postman (publish a null progress log). No bugs were detected.

Performance testing

Let us present an analysis of the performance data obtained from the functional testing. Firstly, we will present the performance results of the tests and later we'll simulate a hypothesis contrast.

Performance data

These tests have been recorded in a computer with the following characteristics: AMD Ryzen® 5000 Series 7 CPU and 16 GB of RAM.

After the execution of the tests, the following graph has been generated, showing the average response time for each request path.

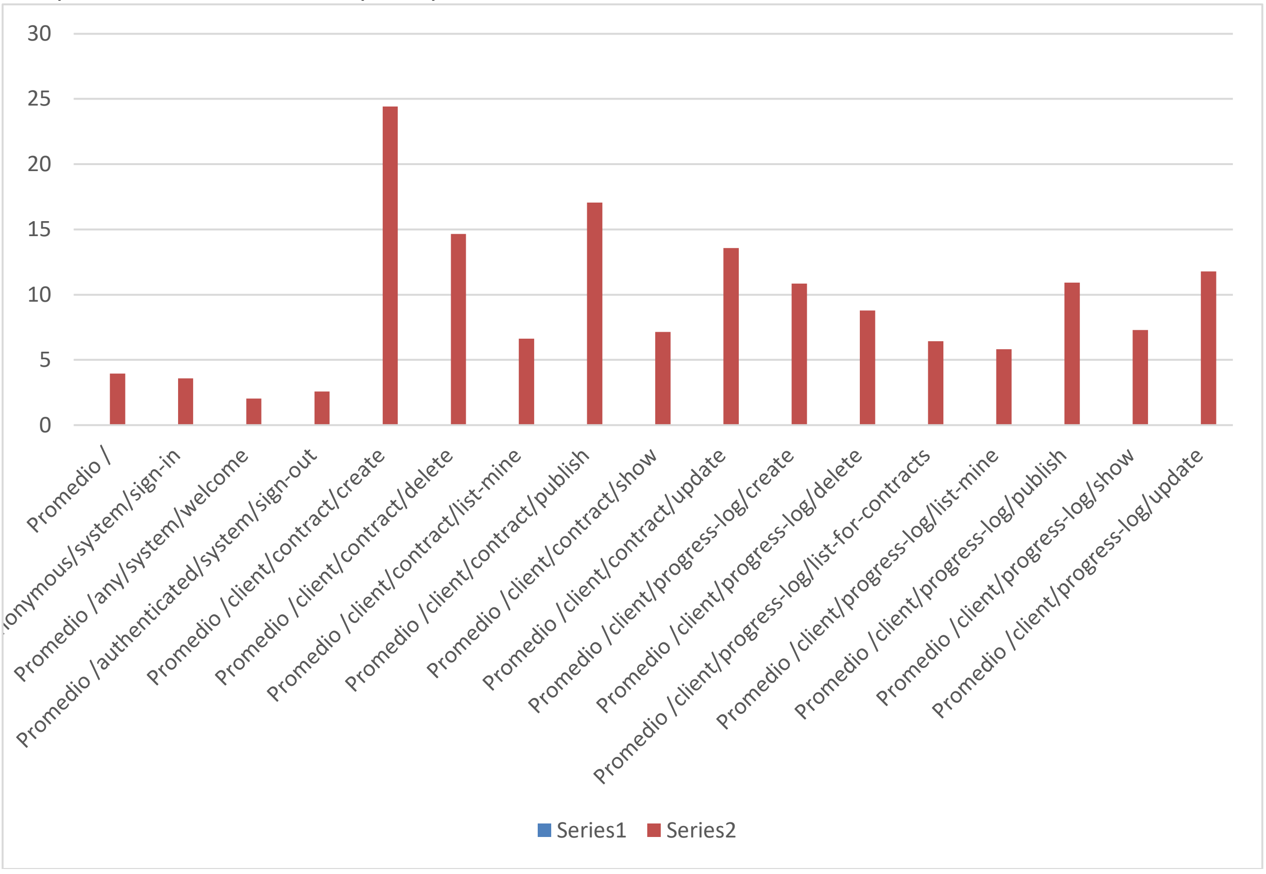
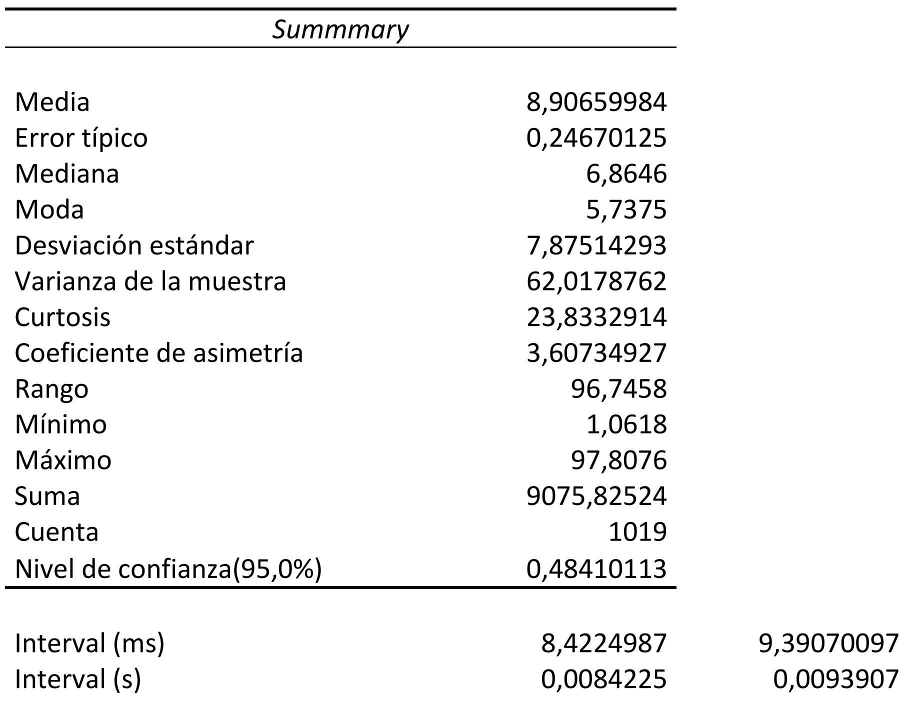


Image 1: Average response time per request path.

One feature is particularly highlighted above the others: the create of a contract. The reason of why this request is the MIR, is because it requires resources of other entities such as, Clients, Progress logs, Projects and even the Configuration class for validating the constraints. The execution path for this feature, is more complex than the rest of the features, and it is no coincidence that the second and third place are for the publish and update of a contract, which have in common all the validations for checking the constraints. Probably, the flow execution could be optimized for avoiding longer response time in these features.

All the other features show a consistent and logical performance. Let us now calculate a confidence interval for the whole test suite.



Summmary		
Media	8,90659984	
Error típico	0,24670125	
Mediana	6,8646	
Moda	5,7375	
Desviación estándar	7,87514293	
Varianza de la muestra	62,0178762	
Curtosis	23,8332914	
Coeficiente de asimetría	3,60734927	
Rango	96,7458	
Mínimo	1,0618	
Máximo	97,8076	
Suma	9075,82524	
Cuenta	1019	
Nivel de confianza(95,0%)	0,48410113	
Interval (ms)	8,4224987	9,39070097
Interval (s)	0,0084225	0,0093907

Image 2: Data analysis summary and confidence level.

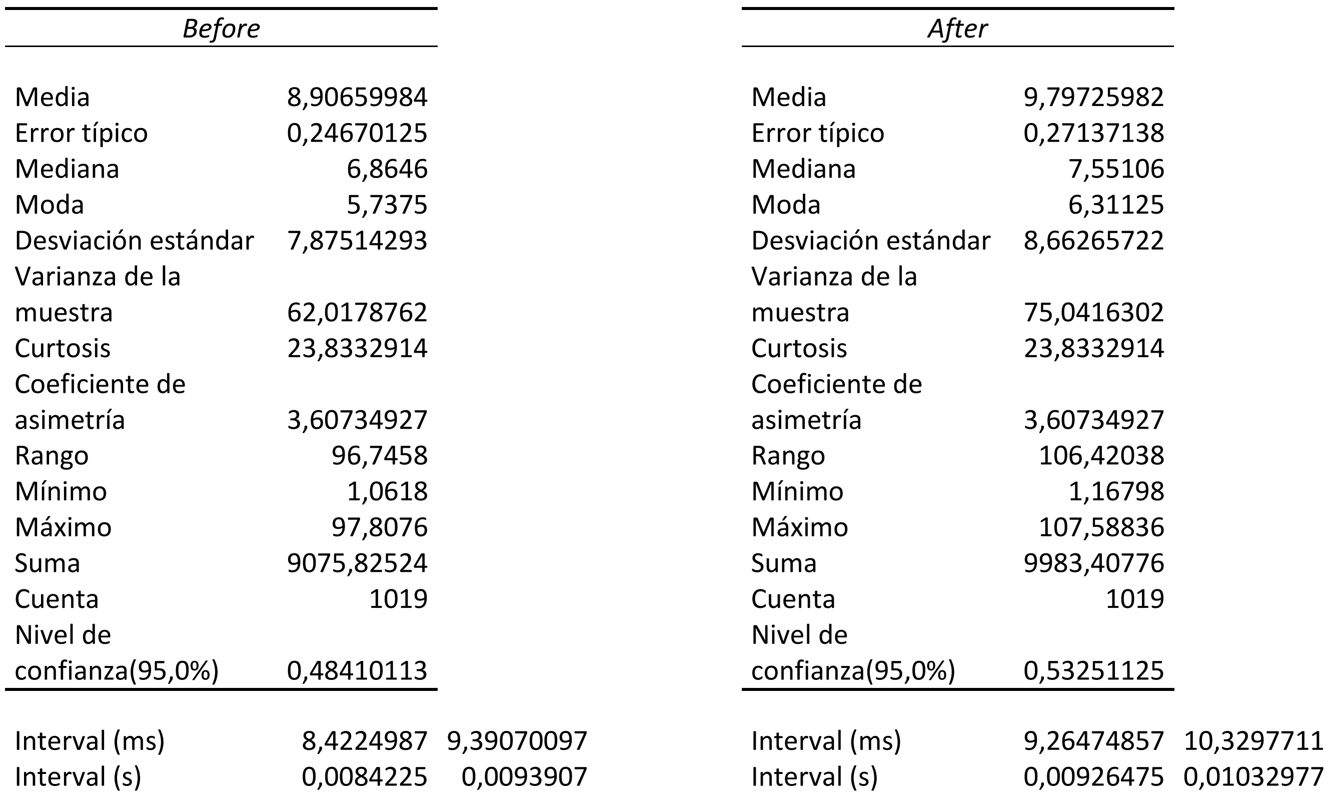
With this data analysis summary, we have computed the 95.0 % confidence interval of our data, which is, in milliseconds, [8,4224987, 9,39070097]. We can also see below the corresponding transformation to seconds.

In this project we don't have any performance requirement to which we can compare our confidence interval. However, in general terms, this could be considered an acceptable response time.

Hypothesis contrast

Since there are no specific performance requirements for this delivery, we will simulate a hypothesis test. The new data will be obtained by increasing the real testing data by 10%.

First, we will generate a data analysis summary for both performance samples and compare the results.



Before	
Media	8,90659984
Error típico	0,24670125
Mediana	6,8646
Moda	5,7375
Desviación estándar	7,87514293
Varianza de la muestra	62,0178762
Curtosis	23,8332914
Coeficiente de asimetría	3,60734927
Rango	96,7458
Mínimo	1,0618
Máximo	97,8076
Suma	9075,82524
Cuenta	1019
Nivel de confianza(95,0%)	0,48410113

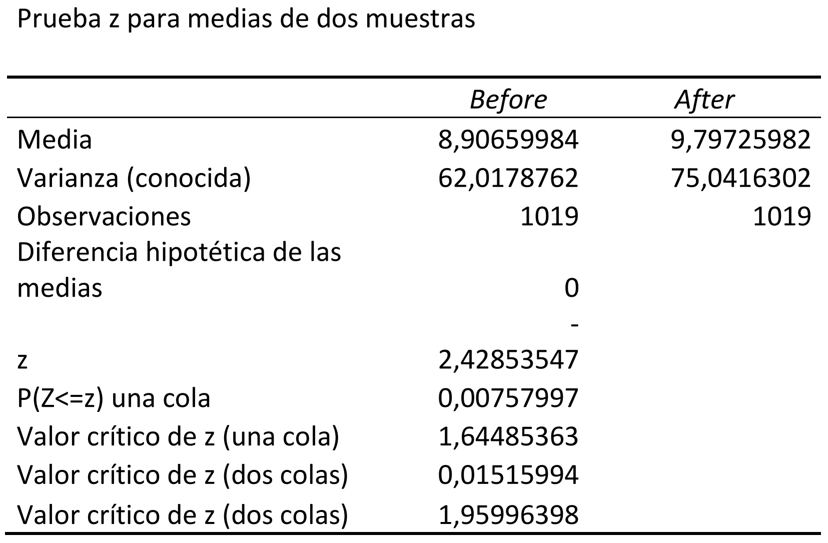
Interval (ms)	8,4224987	9,39070097
Interval (s)	0,0084225	0,0093907

After	
Media	9,79725982
Error típico	0,27137138
Mediana	7,55106
Moda	6,31125
Desviación estándar	8,66265722
Varianza de la muestra	75,0416302
Curtosis	23,8332914
Coeficiente de asimetría	3,60734927
Rango	106,42038
Mínimo	1,16798
Máximo	107,58836
Suma	9983,40776
Cuenta	1019
Nivel de confianza(95,0%)	0,53251125

Interval (ms)	9,26474857	10,3297711
Interval (s)	0,00926475	0,01032977

Image 3: Data analysis summary and confidence level for both samples.

The results have naturally increased, which might be seen as a regression. However, intuitively comparing the confidence intervals is challenging. To address this, we will use a Z-Test. Here are the results:



Prueba z para medias de dos muestras

	Before	*After*
Media	8,90659984	9,79725982
Varianza (conocida)	62,0178762	75,0416302
Observaciones	1019	1019
Diferencia hipotética de las medias	0	
z	-2,42853547	
P(Z<=z) una cola	0,00757997	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,01515994	
Valor crítico de z (dos colas)	1,95996398	

Image 4: Z-Test results between the two performance samples.

A Z-Test is based on a value called alpha, which is computed as 1 minus the confidence level percentage. In this case, alpha will be 0.05. To understand the result of the Z-Test, we need to compare the first two-tail p value (*Valor crítico de z (dos colas)* in Spanish) to alpha. If the p value is below alpha, then we proceed to compare the averages and see if the new average has decreased.

In our case, however, the p value is 0,01515994, which is lower than alpha. Now we can compare the average of the two samples, and realize that the average is higher than before. This means that our supposed changes didn't result in any significant improvement.

Conclusions

This rigorous formal testing mechanism has enabled us to thoroughly analyze our code for bugs. The final results of all tests are overwhelmingly positive, achieving nearly complete instruction coverage (fully within logical bounds) and demonstrating excellent performance, including an impressive response time.

Bibliography

Intentionally blank.