DP2 2023-2024
Analysis report D04

Acme Software Factory

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

Student #3:

- Heras Pérez, Raúl rauherper@alum.us.es

Other members:

- Flores de Francisco, Daniel danflode@alum.us.es
- Mellado Díaz, Luis luimeldia@alum.us.es
- Vento Conesa, Adriana adrvencon@alum.us.es
- Castillo Cebolla, Rafael rafcasceb@alum.us.es

GROUP C1.049

Version 1.0

19-05-24

Content Table

Abstract

In this report the different procedures and results from the formal testing applied to the different individual requirements regarding features will be presented. This includes the study of the performance testing.

Revision Table

Date	Version	Description of the changes	Sprint
18/05/2024	1.0	• Abstract • Introduction • Functional testing	4
19/05/2024	1.0	• Performance testing • Conclusion	4
25/05/2024	1.1	• Updated data • Updated graphs	4

Introduction

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

- Operations by developer on Training Modules.
- Operations by developers on Training Sessions.

This report will be presented in two main parts:

- Functional testing: in this section I will provide a description of all the tests done, along with their results and their success rate in covering the different instructions and, if applicable, any bugs and/or problems detected.

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

It is worth noting that, when referring to covering code, there are some instructions that would never be covered such as the default assertion of *assert object != null.*

Contents

Functional testing

Operations by developer on Training Modules

Test case 1: list

This test consists in clicking the button for listing the different pages of Training Modules corresponding to two different developers.

For hacking, we considered accessing the URL by a user with wrong role, in this case, with no authenticated user. The test was successful as it didn't give permission to see the objects.

It provided a coverage of 95.6 %, covering all instructions except the default assertion. No bugs were detected. However, after a revision with the lecturer, the *unbind* method was changed for efficiency improvements.

Test case 2: show

For this command, we selected several Modules of the listing of Developer 1 to see their details.

For hacking, we tried accessing a Module of Developer 1 without any authenticated user, with an user with the manager role and with another developer who cannot access to that information.

It provided a coverage of 95.8 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 3: create

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

We cannot perform any hacking regarding a create operation as it is a POST operation, and the framework only allows us to test this cases by means of GET requests.

It provided a coverage of 88.1 % at first, covering all instructions except the default assertion and a piece of dead code in the validator that was later deleted.
After solving this minor mistake, the coverage was of 93.8 %

Test case 4: update

For this command, we have updated the project with identifier 804. For each attribute we have checked the system rejects all different types of invalid data, including sending an empty form. Later, for each attribute, we have checked the system accepts all different types of valid data.

For hacking, we tried to update a published module by manipulating the URL, and the system denied it.

We also tried to hack the feature by manipulating the form with the *inspect element* function of the browser. We tried to modify the *creation moment* field, which is supposed to be *read-only*, and try to update the module. This change was successfully rejected by the system and no change was produced.

It provided a coverage of 93.3 %. Apart from not covering the default assertion, it did not covered a validation of the published attribute. This makes sense as the browser does not allow updating a published object.

Test case 5: delete

For this command we tried deleting the module with identifier 781, which cannot be deleted since it a published session. Then we tried deleting the module with identifier 802, which can be deleted.

For hacking, we tried to delete a published module by manipulating the URL, and the system denied it.

It provided a coverage of 93.1 %. Apart from not covering the default assertion, it did not covered a validation of the published attribute. This makes sense as the browser does not allow deleting a published object.

Test case 6: publish

For this command, we have tried to publish three different modules. First, we have tried to update a module which cannot be published since it doesn't have any modules. Then, we tried with a module which cannot be published since it has unpublished user stories. And, finally, we tried with a valid module. As the publish is also an update operation, I did a slightly more relaxed test in every update of the fields.

We cannot perform any hacking regarding a create operation as it is a POST operation, and the framework only allows us to test this cases by means of GET requests.

We also tried to hack the feature by manipulating the form with the *inspect element* function of the browser. We tried to modify the *creation moment* field, which is supposed to be *read-only*, and try to update the module. This change was successfully rejected by the system and no change was produced.

It provided a coverage of 94.1 %. Apart from not covering the default assertion, it did not covered a validation of the published attribute. This makes sense as the browser does not allow publishing a published object.

Operations by developer on Training Sessions.

Test case 1: list

For this command, we just selected the button for listing all training sessions for the different training modules of developer 1.

For hacking, we considered accessing the URL by a user with wrong role, in this case, with no authenticated user and with a manager. The test was successful as it didn't give permission to see the objects.

It provided a coverage of 93.5 %, covering all instructions except the default assertion.

Test case 2: show

For this command, we selected several sessions of the listing of the different modules of developer 1 to see their details.

For hacking, we tried accessing a session of developer 1 with an non authenticated user, a manager and another developer.

It provided a coverage of 94.9 %, covering all instructions but some logical exceptions: default assertions, and some combinations of the assertions. A bug where you could access to any session as long as you were a developer was found and solved.

Test case 3: create

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

We cannot perform any hacking regarding a create operation as it is a POST operation, and the framework only allows us to test this cases by means of GET requests.

It provided a coverage of 95.9 %, covering all instructions except a default assertion, which is logical. There were some errors detected with the validation the *startDate* and *endDate* parameters, which were solved.

Test case 4: update

For this command, we have updated a session. For each attribute we have checked the system rejects all different types of invalid data, including sending an empty form. Later, for each attribute, we have checked the system accepts all different types of valid data.

For hacking, we tried to update a published session by manipulating the URL, and the system denied it.

It provided a coverage of 96.1 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for the different attributes, for example, a published session will not access the update option.

Test case 5: delete

For this command we deleted a session. There is no restriction to test.

For hacking, we tried to delete a published session by manipulating the URL, and the system denied it.

It provided a coverage of 59.4%. This low number of coverage is explicable as the validation of the data does nothing, as there is no restrictions to validate. This leads to an impossible execution of the unbind method, so all of those lines of code are never executed.
After deleting the *unbind* method, we have a 88.9% coverage.

Test case 6: publish

For this command, we have tried to publish a session. Before correctly publishing it, we followed a similar approach to the update tests, since the publishing form also sends all attributes for them to be updated, but in a more relaxed way, checking only trivial error cases for each attribute instead of checking every case.

We cannot perform any hacking regarding a create operation as it is a POST operation, and the framework only allows us to test this cases by means of GET requests.

It provided a coverage of 94.8 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for the different attributes, for example, a published session will not access the update option.

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: Intel® Core™ i5-1155G7 CPU @2.50 GHz and 12 GB of RAM.

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

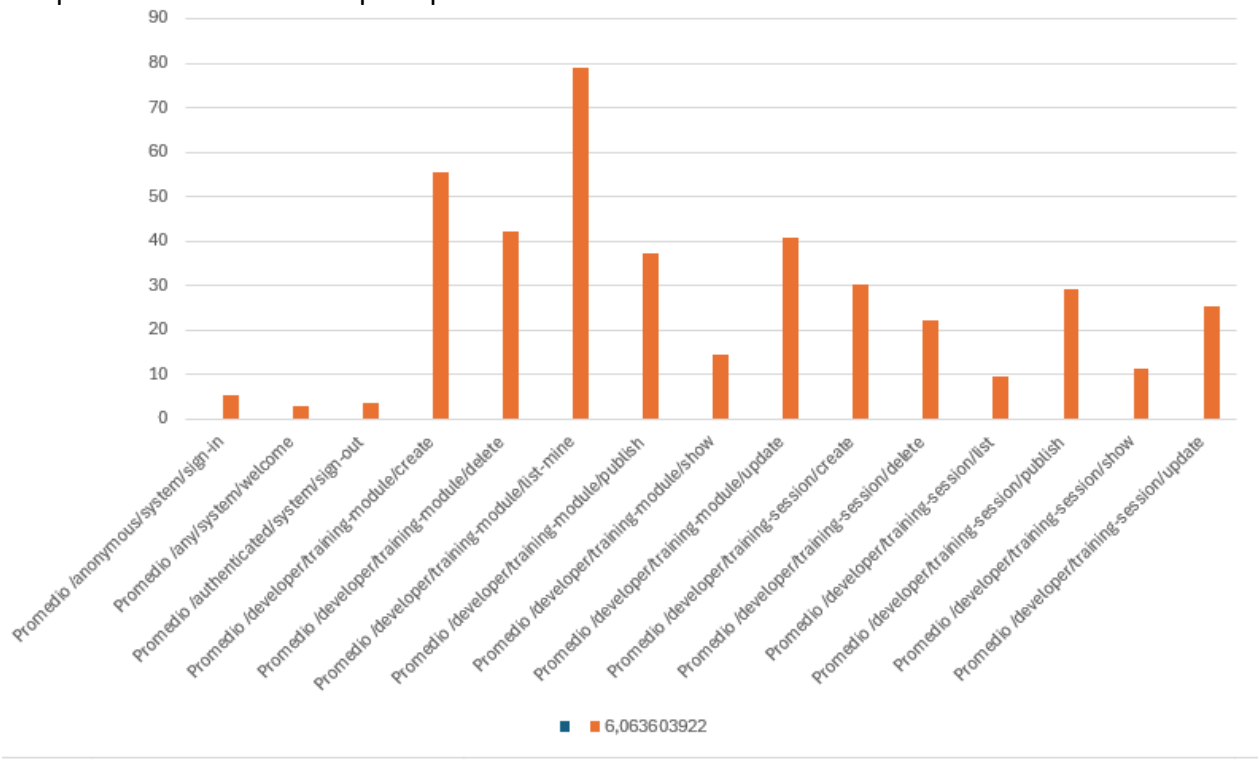

Image 1: Average response time per request path.

We can appreciate a particular spike in the list-mine feature. This can be due to the loading of all the modules that belong to a certain developer, but it is not a very high spike after all.
All the other features show a consistent and logical performance. Let us now calculate a confidence interval for the whole test suite.

			Interval(ms)	30,6474717	25,6745816
Media	28,16102662		Interval(s)	0,03064747	0,02567458
Error típico	1,265438787				
Mediana	24,00875				
Moda	30,8442				
Desviación estándar	27,83965331				
Varianza de la muestr	775,0462966				
Curtosis	13,9074384				
Coeficiente de asimetr	2,772521799				
Rango	244,7893				
Mínimo	1,7317				
Máximo	246,521				
Suma	13629,93688				
Cuenta	484				
Nivel de confianza(95,	2,486445032				

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, [25,6745816, 30,6474717]. 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 we don't have any to meet any performance requirement in this delivery, we are going to simulate a hypothesis contrast. The new data will be obtained my increasing a 10% the real testing data.

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

Media	28,16102662			Media	30,97712928	
Error típico	1,265438787			Error típico	1,391982666	
Mediana	24,00875			Mediana	26,409625	
Moda	30,8442			Moda	33,92862	
Desviación estándar	27,83965331			Desviación está	30,62361864	
Varianza de la muest	775,0462966			Varianza de la r	852,5509263	
Curtosis	13,9074384			Curtosis	15,29818224	
Coeficiente de asimet	2,772521799			Coeficiente de a	3,049773979	
Rango	244,7893			Rango	269,26823	
Mínimo	1,7317			Mínimo	1,90487	
Máximo	246,521			Máximo	271,1731	
Suma	13629,93688			Suma	14992,93057	
Cuenta	484			Cuenta	532,4	
Nivel de confianza(95	2,486445032			Nivel de confiar	2,735089536	
Interval(ms)	30,64747165	25,6745816		Interval(ms)	1,391982666	1,39198267
Interval(s)	0,030647472	0,02567458		Interval(s)	0,001391983	0,00139198

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

The results have naturally increased, which could be considered a downgrade. However, comparing the confidence intervals intuitively is not an easy task. For this purpose, we will make use of a Z-Test. Behold its results:

	133,2761	*146,60371*
Media	25,8497465	28,4347212
Varianza (co	28,1610266	30,9771293
Observacion	518	518
Diferencia hi	0	
z	-7,65045526	
P(Z<=z) una	9,992E-15	
Valor crítico	1,64485363	
Valor crítico	1,9984E-14	
Valor crítico	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.

The p-value is in range [0,00, alpha], so we compare both averages, and we see that, naturally, there has been an increase in the average of almost 3 units, so we can conclude that we have not improved the performance.

Conclusions

Our thorough testing approach has allowed us to catch and fix bugs in our code. We found and resolved some minor issues and understood better the functioning of the different functionalities. The final test results are very positive, with almost complete instruction coverage and excellent performance, including a great average response time

Bibliography

Intentionally blank.