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

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 96.1 %, covering all instructions except the default assertion. No bugs were detected.

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.

It provided a coverage of 96.2 %, 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.

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 93.0 %. 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.

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 93.0 %. 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 the project with identifier 802, which cannot be published since it doesn't have any modules. Then, we tried with the module with identifier 797 , which cannot be published since it has unpublished user stories. And, finally, we tried with the project with identifier 781. This project was valid for publishing, however, 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.

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 94.7 %. 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.

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. 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 module of developer 1 with a wrong role.

It provided a coverage of 89.5 %, covering all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, since the framework 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 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 case45: update

For this command, we have updated the session with identifier 868. 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 %. 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 the session with identifier 821. There is no restriction to test.

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 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.

Test case 6: publish

For this command, we have tried to publish the session with identifier 868. 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 95.8 %. 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.

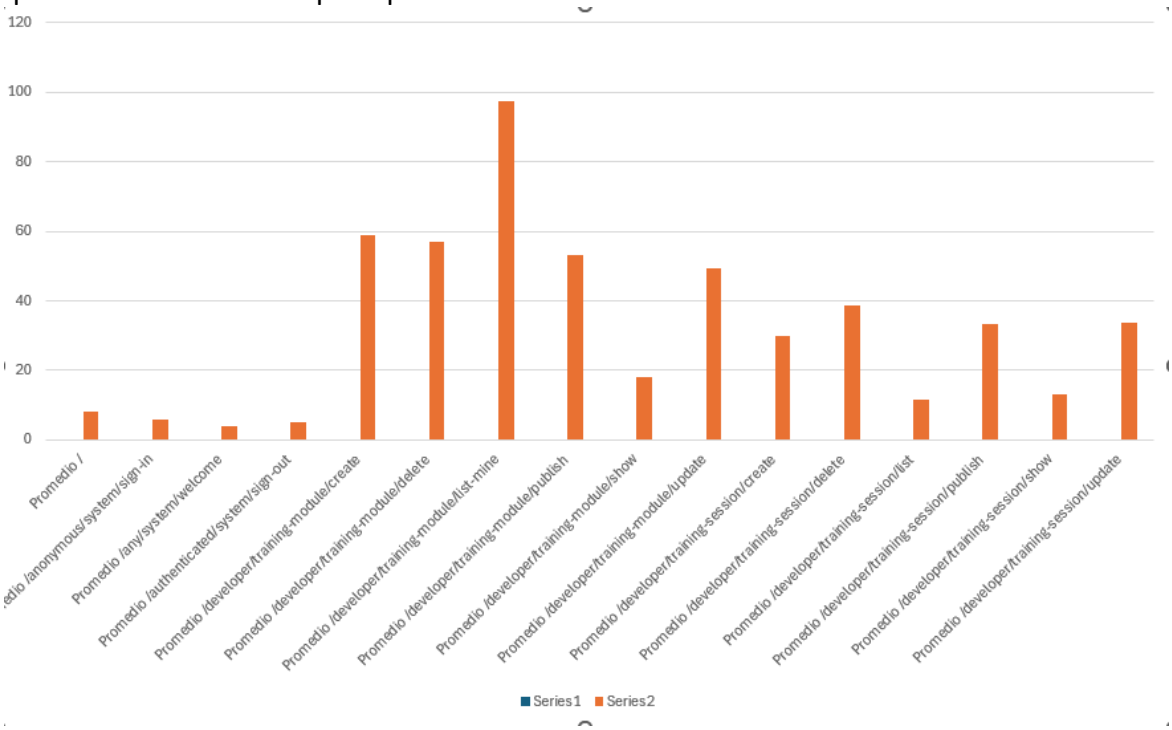

Image 1: Average response time per request path.

We can appreciate a particular spike in the list-mine feature. This can be due to the fact that there were done huge amounts of listing. In the list-mine test, all the pages of the listing of the modules for two different developers were performed. This is added to all the listings made in all the rest of the tests, including those involving the training sessions, as we have to access the list of training modules first.
All the other features show a consistent and logical performance. Let us now calculate a confidence interval for the whole test suite.

Columna1					
Media	33,3088372	Interval(ms)	36,2560939	30,3615805	
Error típico	1,49956532	Interval(s)	0,03625609	0,03036158	
Mediana	27,8195				
Moda	#N/D				
Desviación e	31,3835771				
Varianza de	984,928914				
Curtosis	4,9471764				
Coeficiente d	1,7419868				
Rango	222,0175				
Mínimo	2,46				
Máximo	224,4775				
Suma	14589,2707				
Cuenta	438				
Nivel de conf	2,94725668				

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, [30,36158054, 36,25609389]. 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.

Columna1				*Columna1*		
Media	33,3088372			Media	36,6397209	
Error típico	1,49956532			Error típico	1,64952185	
Mediana	27,8195			Mediana	30,60145	
Moda	#N/D			Moda	#N/D	
Desviación e	31,3835771			Desviación e	34,5219348	
Varianza de	984,928914			Varianza de	1083,42181	
Curtosis	4,9471764			Curtosis	5,44189404	
Coeficiente c	1,7419868			Coeficiente c	1,91618548	
Rango	222,0175			Rango	244,21925	
Mínimo	2,46			Mínimo	2,706	
Máximo	224,4775			Máximo	246,92525	
Suma	14589,2707			Suma	16048,1978	
Cuenta	438			Cuenta	481,8	
Nivel de conf	2,94725668			Nivel de conf	3,24198235	
Interval(ms)	36,2560939	30,3615805		Interval(ms)	39,8817033	33,3977386
Interval(s)	0,03625609	0,03036158		Interval(s)	0,0398817	0,03339774

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:

Prueba z para medias de dos muestras		
	33,3088372	*36,6397209*
Media	1377,62447	1515,38692
Varianza (co	984,928914	1083,42181
Observacion	12	12
Diferencia hi	0	
z	-10,4932346	
P(Z<=z) una	0	
Valor crítico	1,64485363	
Valor crítico	0	
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 0, so we compare both averages, and we see that, naturally, there has been an increase in the average of 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.