DP2 2023-2024

Analysis report D04

Acme Software Factory

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

Student #1:

- Castillo Cebolla, Rafael rafcasceb@alum.us.es

Other members:

- Flores de Francisco, Daniel danflode@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 1.0

11-05-24

Content Table

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
10/05/2024	1.0	• Executive summary • Introduction • Functional testing	4
11/05/2024	1.0	• Performance testing • Conclusion • Bibliography	4

Introduction

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

- Operations by managers on Projects.
- Operations by managers on User stories.

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 project to serve the requests in the functional tests and a 95%-confidence hypothesis contrast.

Contents

Functional testing

Operations by manager on Projects

Test case 1: list-mine

For this command, we just selected the button for listing all projects for ten different managers.

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.

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 several projects of the listing of Manager 1 to see their details. Then we tried accessing the inexistent project of identifier -1.

For hacking, we tried accessing a project of Manger 1 with a wrong role and with Manager 2.

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 project. 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, the framework through web browser only supports to test GET hacking operations.

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

Test case 4: update

For this command, we have updated the project with identifier 153. 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, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 92.4 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story (a published user story cannot access the update function), since the framework from the web browser doesn't allow to. No bugs were detected.

Test case 5: delete

For this command we tried deleting the project with identifier 149, which cannot be deleted since it has child audits, child contracts, child sponsorships, child training modules. Then we tried deleting the project with identifier 150, which cannot be deleted since it is published. Lastly, we correctly deleted the project with identifier 153.

For hacking, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 91.6 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story (a published user story cannot access the delete function), since the framework from the web browser doesn't allow to. No bugs were detected.

Test case 6: publish

For this command, we have tried to publish three different projects. First, we have tried to update the project with identifier 153, which cannot be published since it doesn't have any user story. Then, we tried with the project with identifier 149, which cannot be published since it has unpublished user stories. And, finally, we tried with the project with identifier 151. 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. With this project we also tried publishing it with fatal errors.

For hacking, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 93.6 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story (a published user story cannot access the publishing function), since the framework from the web browser doesn't allow to. Besides, it didn't fully check either a two-components condition for same code checking, which by its particular definition, it wouldn't make sense to fulfil all its paths. No bugs were detected.

Operations by manager on User stories

Test case 1: list-mine

For this command, we just selected the button for listing all user stories for ten different managers.

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.

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-project

localhost:8082/acme-sf-d04/manager/user-story/list-for-project?projectId=151

For this command, we clicked on the button to see the user stories of three different projects: project with identifier 149, which had multiple children user stories, project with identifier 151, which had one child user story, and project with identifier 153, which had none. Then we tried accessing the inexistent project of identifier -1.

For hacking, we tried accessing the same user story listing with a wrong role and with Manager 2.

It provided a coverage of 95.5 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 3: show

For this command, we selected several user stories of the listing of Manager 1 to see their details. Then we tried accessing the inexistent project of identifier -1.

For hacking, we tried accessing a user story of Manager 1 with a wrong role and with Manager 2.

It provided a coverage of 96.5 %, 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 4: create

For this command, we have tried to create a new user story. 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, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 92.9 %, covering all instructions except a default assertion, which is logical. No bugs were detected.

Test case 5: update

For this command, we have updated the user story with identifier 631. 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, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 92.3 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story (a published user story cannot access the update function), since the framework from the web browser doesn't allow to. No bugs were detected.

Test case 6: delete

For this command we deleted the user story with identifier 631. There is no restriction to test.

For hacking, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 67.4 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story (a published user story cannot access the delete function), since the framework from the web browser doesn't allow to. Besides, it didn't check either the unbind of the service since from the framework it is never possible to cause failures in a safe delete of a user story. No bugs were detected.

Test case 7: publish

For this command, we have tried to publish the user story with identifier 631. 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.

For hacking, the framework through web browser only supports to test GET hacking operations.

It provided a coverage of 92.6 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story (a published user story cannot access the publishing function), since the framework from the web browser doesn't allow to. No bugs were detected.

Operations on ProjectUserStory intermediate table

Test case 1: create

For this command we tried linking projects to user stories. We checked that an association to a null project was rejected, that a published user story could link projects and that a published project could not be linked any further.

For hacking, we tried accessing the linking page of a user story of Manager 1 with a wrong role and with Manager 2.

It provided a coverage of 92.4 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story, since the framework from the web browser doesn't allow to. No bugs were detected.

Test case 2: delete

For this command we tried unlinking projects to user stories. We checked that unlinking a null project was rejected, that a published user story could unlink projects, that deleting a project deleted the association and that a published project could not be unlinked any further.

For hacking, we tried accessing the unlinking page of a user story of Manager 1 with a wrong role and with Manager 2.

It provided a coverage of 91.8 %. It covered all instructions but some logical exceptions: default assertions, and some combinations of the assertions for wrong user, null user story or published user story, since the framework from the web browser doesn't allow to. 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: Intel® Core™ i7-10750H CPU @2.60 GHz 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.

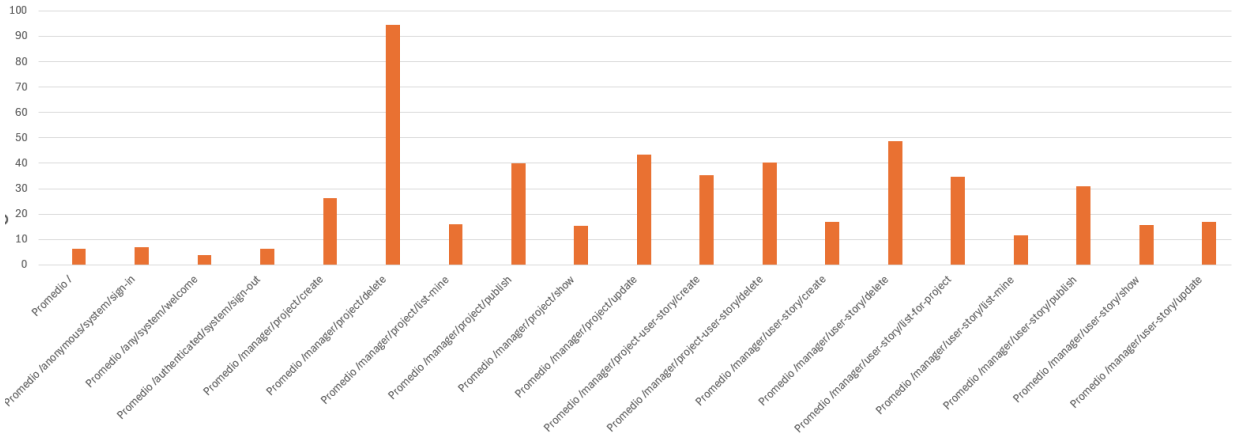

Image 1: Average response time per request path.

One feature is particularly highlighted above the others: the delete of a project. The entity Project has direct parent dependencies with five other entities: Audit, Contract, Sponsorship, TrainingModule and UserStory, which constraint its functionalities. A project cannot be deleted if it has a child of any of the first four previous entities, and in case of having a child user story, the intermediate table which links both instances should be deleted as well. Therefore, performing a delete of a project is not a simple task and must execute multiple queries and operations. Thus, it is logical that its response time is so elevated in comparison with the rest.

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

Columna1	
Media	15,95862208
Error típico	0,724455842
Mediana	9,7709
Moda	3,0004
Desviación estándar	20,21997655
Varianza de la muestra	408,8474517
Curtosis	18,76297591
Coeficiente de asimetría	3,41005332
Rango	215,9062
Mínimo	2,2201
Máximo	218,1263
Suma	12431,7666
Cuenta	779
Nivel de confianza(95,0%)	1,422119745

	Lower limit	Upper limit
Confidence interval (ms):	14,5365023	17,3807418
Confidence interval (s):	0,0145365	0,01738074

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, [14.5365023, 17,3807418]. 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.

Before				*After*		
Media	15,95862208			Media	17,55448428	
Error típico	0,724455842			Error típico	0,796901426	
Mediana	9,7709			Mediana	10,74799	
Moda	3,0004			Moda	3,30044	
Desviación estándar	20,21997655			Desviación estándar	22,2419742	
Varianza de la muestra	408,8474517			Varianza de la muestra	494,7054165	
Curtosis	18,76297591			Curtosis	18,76297591	
Coeficiente de asimetría	3,41005332			Coeficiente de asimetría	3,41005332	
Rango	215,9062			Rango	237,49682	
Mínimo	2,2201			Mínimo	2,44211	
Máximo	218,1263			Máximo	239,93893	
Suma	12431,7666			Suma	13674,94326	
Cuenta	779			Cuenta	779	
Nivel de confianza(95,0%)	1,422119745			Nivel de confianza(95,0%)	1,56433172	
	Lower limit	Upper limit			Lower limit	Upper limit
Confidence interval (ms):	14,53650233	17,3807418		Confidence interval (ms):	15,99015257	19,118816
Confidence interval (s):	0,014536502	0,01738074		Confidence interval (s):	0,015990153	0,01911882

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		
	Before	*After*
Media	15,95862208	17,55448428
Varianza (conocida)	408,8474517	494,7054165
Observaciones	779	779
Diferencia hipotética de las medias	0	
z	-1,481792303	
P(Z<=z) una cola	0,069197784	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,138395568	
Valor crítico de z (dos colas)	1,959963985	

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 0.138395568, which is considerably higher than alpha. This means that our supposed changes didn't result in any significant improvement. The sample times are different, but globally, they are considered the same.

Conclusions

This strict mechanism for formal testing has allowed us to exhaustively analyze our code in search for bugs in our code. We found two small details in an unbind method of a service which were not correctly functioning and proceeded to solve them. The final result of all tests is overwhelmingly positive, with an almost full instruction coverage (full within logic) and showing great performance results, with a great average response time.

Bibliography

Intentionally blank.