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 3](#_Toc163299977)

[Revision Table 4](#_Toc163299978)

[Introduction 5](#_Toc163299983)

[Contents 6](#_Toc163299984)

[Mandatory task D03-6: 6](#_Toc163299985)

[Mandatory task D03-07: 6](#_Toc163299986)

[Mandatory task D03-8: 7](#_Toc163299987)

[Extra task D03-17: 7](#_Toc163299988)

[Extra task D03-18: 7](#_Toc163299989)

[Extra task D03-19: 8](#_Toc163299990)

[Extra task D03-20: 8](#_Toc163299991)

[Extra task D03-21: 8](#_Toc163299993)

[Extra task D03-22: 8](#_Toc163299995)

[Conclusions 9](#_Toc163299997)

[Bibliography 10](#_Toc163299998)

# 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 |
|  | 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 nothing important. 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 nothing important. 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 nothing important. 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 completely covered all instructions except some default assertions, which is nothing important, the assertions for wrong user or null project since the framework doesn’t allow to, and the assertion of published project, since the publishing option is accessed through a button which simply doesn’t appear in the frontend if the condition is not met. 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 completely covered all instructions except some default assertions, which is nothing important, the assertions for wrong user or null project since the framework doesn’t allow to, and the assertion of published project, since the publishing option is accessed through a button which simply doesn’t appear in the frontend if the condition is not met. 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 completely covered all instructions except some default assertions, which is nothing important, the assertions for wrong user or null project since the framework doesn’t allow to; the assertion of published project, since the publishing option is accessed through a button which simply doesn’t appear in the frontend if the condition is not met; the assertion of errors in the published button, which is a Boolean checkbox, so could not have errors, and a two-components condition for same code checking, which by its 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 nothing important. 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 nothing important. 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 except, logically, a default assertion and a null user story and manager assertion, which is nothing important. 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 nothing important. 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 completely covered all instructions except some default assertions, which is nothing important, and the assertions for wrong user, null user story or published user story, since the framework doesn’t allow to (publishing option is accessed through a button which simply doesn’t appear in the frontend if user story is published). 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 completely covered all instructions except some default assertions, which is nothing important, and the assertions for wrong user, null user story or published user story, since the framework doesn’t allow to (publishing option is accessed through a button which simply doesn’t appear in the frontend if user story is published). 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 except some default assertions, which is nothing important, and the assertions for wrong user, null user story or published user story, since the framework doesn’t allow to (publishing option is accessed through a button which simply doesn’t appear in the frontend if user story is published). No bugs were detected.

### Operations on ProjectUserStory intermediate table

http://localhost:8082/acme-sf-d04/manager/user-story/show?id=631

http://localhost:8082/acme-sf-d04/manager/project/show?id=151

#### 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 with 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 with 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

Blablablalbla

# Conclusions

For the delivery number three, I have found some aspects in need of further analysis, mainly to comment about important decisions that had to be taken, more than contradictions or ambiguities.

# 

# Bibliography

Intentionally blank.