Student #3, Sprint 1: Analisis report

Group: C1.04.14

Repository: https://github.com/marizqlav/Acme-L3-D04

Student #1

Name: Domínguez-Adame, Alberto
email: albdomrui@alum.us.es

Student #2

Name: Herrera Ramírez, Ismael
email: ismherram@alum.us.es

Student #3

Name: Olmedo Marín, Marcos
email: marolmmar1@alum.us.es

Student #4

Name: Izquierdo Lavado, Mario
email: marizqlav @alum.us.es

Student #5

Name: Merino Palma, Alejandro
email: alemerpal@alum.us.es

Table of contents

Summary

Acme Life-Long Learning, Inc. (Acme L3, Inc. for short) is a company that specializes in helping learners get started on a variety of matters with the help of renowned lecturers. The goal of this project is to develop a WIS to help this organization manage their business.

Revision table

Number	Date	Description
1	15/03/2023	Full redaction of the document

Introduction

This document lists analysis records, each one including the following data: a verbatim copy of the requirement to which the record refers; detailed conclusions from the analysis and decisions made to mend the requirement; a link to the validation performed by a lecturer. .

This document has the following structure:

- Analysis report

Contents

Analysis records

All requirements not listed here didn't required any analysis to be done.

Requirement nº5: An audit is a document with auditing records regarding a published course. The system must store the following data about them: a code (pattern "[A-Z]{1,3}[0-9][0-9]{3}", not blank, unique), a conclusion (not blank, shorter than 101 characters), some strong points (not blank, shorter than 101 characters), some weak points (not blank, shorter than 101 characters), and a mark (computed as the mode of the marks in the corresponding auditing records; ties must be broken arbitrarily if necessary).

There were to design decisions regarding this requirement, both about the mark attribute. The first about if it should be transient or not, the first one about how to obtain the required auditingRecords.

About it being or not transient:

Decisions: Due to the mark attribute being an attribute derived from the related AuditingRecords of the Audit, we had to obtain it somehow from the Audit. Some options where considered:

- We could make the mark attribute a transient one and obtain it every time the user calls the function.
- We could make the mark attribute a serializable attribute and have it update every time a new AuditingRecord is added to the audit.

Conclusions: It was decided the former, since the method for obtaining the mark from the related AuditingRecords is not really performance heavy. This was verified by the teacher at laboratory class.

About how to obtain the auditingRecords.

Decisions: in order to calculate the mark, we need all auditingRecords related to the audit. For that:

- We could use a bidirectional relationship. This is the more intuitive and simple system.
- We could use complicated and queries so that the mark would be a property obtained via the auditService and it would not be included anyway in the entity.

Conclusion: after some talking with our lab teacher, they told us to not use OneToMany or bidirectional relationship in any case, so we used to second option, leaving mark as a transient attribute in our UML and creating a method in the AuditService class.

Requirement nº7: The system must handle auditor dashboards with the following data: total number of audits that they have written for theory and hand-on courses; average, deviation, minimum, and maximum number of auditing records in their audits; average, deviation, minimum, and maximum time of the period lengths in their auditing records.

All students had some doubts about how we should represent average, deviation, min and max values in all dashboards. We had some options:

- We could have all values with their names as an individual attribute. This would allow more flexibility and granularity when if we needed to use these values alone later. It would also be the more intuitive way.

- We could join related values in a map (average, deviation, min and max attributes together) since they are calculated together and they are likely to be required together.

It was decided that the average, deviation, minimum and maximum would be returned via a map with the “average”, “deviation”, “min” and “max” keys and their respective values. This was verified by the teacher at lab class.

Requirement nº4: There is a new project-specific role called auditor, which has the following profile data: firm (not blank, shorter than 76 characters), professional ID (not blank, shorter than 26 characters), a list of certifications (not blank, shorter than 101 characters), and an optional link with further information.

Decisions: we thought that the professional ID attribute would probably be a unique attribute among all auditors.

Conclusions: we ask out lab teacher and he confirmed this, so we made it unique.

Period datatype problems:

Decisions: Since there were a lot of classes that required a duration attribute, we thought some options to solve the problem all together. We first thought of using a simple Duration value with the Double type. Then we thought of using to attributes, firstDate and lastDate to be able to obtain the duration between the two. Finally, we thought of creating a custom datatype called Period which would contain this two dates so it would be easier to implement.

Conclusions: we thought the double type option wouldn’t be enough, and our lab teacher told us too. We also created a datatype called Period to substitute the duration attributes. However, we weren’t able to use them due to a framework error in the population process. More information about this can be found in the On your tutorials section of the subject by the name of **On framework parsing error (probably)**.

So we finally decided to use two simple dates attributes on each class requiring it.

Requirement nº13: Operations by any authenticated principals on audits. List the audits associated with a course.

Decisions: this one was really convoluted. I wanted to get a list with all audits from a course that could be selected through a droppable selector in the same page. The service was pretty straight forward. However, I didn't have an easy way to send the course (course id) through the url to the service. I considered some options:

- Just submitting the course with an acme:form. This was the first one that I thought and the most intuitive. It didn't work. This is because acme:form only submits with POST method and the list endpoint throws an error since it doesn't supports POST methods.

- Doing some js shenanigans. Since the acme framework is so limited, I tend to default to using custom html and js for everything it fails at the first. This would have implemented some complicated function which analyzed the selected value and reloaded a button url with this value. This was simple at concept but very hard to implement.

- Using custom tags and an option on acme:submit. So I just remembered that I could do that. Since the first approach didn't work as expected because the form used the POST method. So I just made a custom tag called custom:form-get which was basically the acme:form but with GET method and without the return button (because that looked bad). While researching for this I realized the acme:submit had an option for specifying the type of method. So I implemented both. This ended up being more simple than I thought. The only backside is that the get method messes up a bit the url, but it still works.

Conclusions: I used the last option since it worked just as expected and it was simpler than the second option. This didn't require to ask our teacher because it was pretty obvious when I figured it out.

Requirement nº14: Operations by auditors on audits.

Decisions: we faced this decision with a lot of entities. Should we allow to modify and / or set the code property? On one hand, that makes things harder to implement. On the other hand, is still a property like the other ones which may be wanted to change.

Also, there was a problem with highlighting the published audits in the list, which was not required but I thought it was a logically needed feature. This is basically the same problem with highlighting the auditing records so I'll explain it later.

Conclusions: we ended up concluding, after talking it with our lab teacher, that the code should be created by the application since it mostly works like a second id.

Requirement nº15: Operations by auditors on auditing records.

Decisions: first, I had a problem since I had to highlight the correction auditing records. Doing this on the form.jsp was pretty straight forward. However, doing this in the list.jsp not so much. I wanted to put in another color all correction audits. First I thought in using a jstl:if and check the correction property. However, acme doesn't allow to process each object of the list like a for loop, so I couldn't do that. I figured out to option:

- Not using acme:list and doing my own list with jstl fors and custom html. This would be the most ductile and flexible solution, but it would be the most complex one too.

- Processing the correction property from the service and converting it into some kind of mark that would show fine in a column. This wasn't what you would call pleasing to the eyes but it was the simplest and fastest option.

Another relevant decision was the confirmation button. Since acme doesn't implement an easy way of doing this (not that I know) I figured out two options:

- Creating a new service derived from create for confirming the correction auditing record. In this case I would redirect to this service from the correct service and ask to accept or decline, in which case to accept it would create the correction auditing record. This was the most "acme way" of doing it and it ensured that someone would have to confirm in case it wanted to correct.

- Just using a simple js script that, when the correct button was clicked, it showed to other buttons for accept and decline, having the accept button the true url to the correct service. This is simple and intuitive, but it allows a user to just write in the url and pass over the confirmation.

Conclusions: about the highlighting, since when I did this I couldn't spare much time, I decided to go with the second option which is ugly but still functional (which goes against my personal principles >:(and I'm still annoyed with it). I may implement the second option (or another one that I may find) in the next sprint in case it is requested.

About the confirmation, I chosed the last option since it was the simplest and I don't feel it's downside has much sense. Like, a confirmation is there in case you haven't noticed that the audit is published. You can pass over my confirmation by writing in the url, but by doing this and literally writing "correct" you are kind of presuppose to know that you are correcting, not just creating.

I also created a derived service from create for correcting the audits. I could have used the create service with a few ifs, but I wanted it to be more clear and less coupled.

Conclusion

In conclusion, this sprint didn’t required much analysis since the test were pretty straight forward to do.

Bibliography

Intentionally blank.